24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Magstar, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15 and 22 above.	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Magstar, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 22 above.	

Appendix A33
Invalidity of U.S. Patent 7,181,608 based on Magstar

27.1 a boot device..	Magstar, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1 above.	

27.2 a processor..	Magstar, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

27.3 cache memory; and.	Magstar, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.3 and 1.4 above.	

27.4 non-volatile memory for storing logic code for use by the processor,..	Magstar, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 and 1.3 above.	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Magstar, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Magstar, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.3 above.	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Magstar, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4 above.	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Magstar, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 4, 10, and 11 above.	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Magstar, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15 and 27 above.	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Magstar, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 27 above.	

Appendix A34
Invalidity of U.S. Patent 7,181,608 based on Mealey

The publication Mealey, B, IBM, An IP.com Prior Art Database Technical Disclosure, January, 1992 ("Mealey") invalidates claims 1-13, 15-16, 19-20, 22, 24-25, 27, and 29-30 of United States Patent No. 7,181,608 ("the '608 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '608 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix A34
Invalidity of U.S. Patent 7,181,608 based on Mealey

1 (Preamble) A method for providing accelerated loading of an operating system, comprising the steps of:	Mealey, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system, comprising the steps of:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.1 maintaining a list of boot data used for booting a computer system;	Mealey, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.2 initializing a central processing unit of the computer system;	Mealey, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.3 preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and	Mealey, as evidenced by the example citations below, discloses "preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.4 servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.	Mealey, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

2. The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.	Mealey, as evidenced by the example citations below, discloses "wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

3. The method of claim 1, wherein the preloading is performed by a data storage controller connected to the boot device.	Mealey, as evidenced by the example citations below, discloses "wherein the preloading is performed by a data storage controller connected to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the preloading is performed by a data storage controller connected to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

See Claims 1.3, and 1.4 above.

See also

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

4. The method of claim 1, further comprising updating the list of boot data.	Mealey, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

See Claim 1.1 above.

See also

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

5. The method of claim 4, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.	Mealey, as evidenced by the example citations below, discloses "wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1 and 4 above.	

6. The method of claim 4, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.	Mealey, as evidenced by the example citations below, discloses "wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1 and 4 above.	

7. (Preamble) A system for providing accelerated loading of an operating system of a host system comprising:	Mealey, as evidenced by the example citations below, discloses "a system for providing accelerated loading of an operating system of a host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system for providing accelerated loading of an operating system of a host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1 (Preamble) above.	

7.1 a digital signal processor (DSP) or controller;	Mealey, as evidenced by the example citations below, discloses "a digital signal processor (DSP) or controller."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a digital signal processor (DSP) or controller), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.2, 1.3, and 1.4 above.	

7.2 a cache memory device; and;	Mealey, as evidenced by the example citations below, discloses "a cache memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a cache memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

7.3.1 a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system	Mealey, as evidenced by the example citations below, discloses "a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1, 1.3, 2, 3, and 7.1 above.	

7.3.2 for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system	Mealey, as evidenced by the example citations below, discloses "for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3, and 1.4 above.	

7.3.3 and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system	Mealey, as evidenced by the example citations below, discloses "decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3, and 1.4 above.	

8. The system of claim 7, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.	Mealey, as evidenced by the example citations below, discloses "wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1, 1.3, 1.4, 2, 3, and 7 above.	

9.1 maintaining a list of application data associated with an application program;	Mealey, as evidenced by the example citations below, discloses "maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1, 2, and 8 above.	

9.2 preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and	Mealey, as evidenced by the example citations below, discloses "preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3, 2, and 8 above.	

9.3 servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.	Mealey, as evidenced by the example citations below, discloses "servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.".
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.4, 2, and 8 above.	

10. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.	Mealey, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image.
> Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

11. The method of claim 1, wherein the decompressing is provided by a data compression engine.	Mealey, as evidenced by the example citations below, discloses "decompressing is provided by a data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing is provided by a data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

12. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.	Mealey, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 10 and 11 above.	

13. The method of claim 1, wherein the compressed boot data is accessed via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the compressed boot data is accessed via direct memory access."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the compressed boot data is accessed via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4 above.	

15. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Mealey, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4 above.	

16. The method of claim 1, wherein a plurality of encoders are utilized to provide the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3, 1.4, and 15 above.	

19. The method of claim 7, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Mealey, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 15 above.	

20. The method of claim 7, wherein a plurality of encoders are utilized to provide the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 16 above.	

22.1 maintaining a list of boot data used for booting a computer system;.	Mealey, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

22.2 initializing a central processing unit of the computer system;.	Mealey, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

22.3 preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit;.	Mealey, as evidenced by the example citations below, discloses "preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

22.4 servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data	Mealey, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4 above.	

22.5 with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device.	Mealey, as evidenced by the example citations below, discloses "with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 4, 10 and 11 above.	

24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Mealey, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15 and 22 above.	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Mealey, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 22 above.	

27.1 a boot device..	Mealey, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1 above.	

27.2 a processor..	Mealey, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

27.3 cache memory; and.	Mealey, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4 above.	

27.4 non-volatile memory for storing logic code for use by the processor,..	Mealey, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 and 1.3 above.	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Mealey, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Mealey, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Mealey, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4 above.	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Mealey, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 4, 10, and 11 above.	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Mealey, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15 and 27 above.	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Mealey, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 27 above.	

Appendix A35
Invalidity of U.S. Patent 7,181,608 based on Menon

The publication Menon, A performance comparison of RAID-5 and log-structured arrays, IBM Almaden Research Center, ("Menon") invalidates claims 1-13, 15-16, 19-20, 22, 24-25, 27, and 29-30 of United States Patent No. 7,181,608 ("the '608 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '608 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix A35
Invalidity of U.S. Patent 7,181,608 based on Menon

1 (Preamble) A method for providing accelerated loading of an operating system, comprising the steps of:	Menon, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system, comprising the steps of:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "A related parameter is the compression ratio. Improved compression ratios help in two ways. First, the better the compression ratio the greater the free space and, hence, the better the performance. Second, the better the compression ratio, the better the cache hit Iatios. This second factor aEects performance quite significantly, particularly at low 1/0 rates where it has a bigger impact than the first factor."

Menon, 173.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

> "In understanding these comparisons, keep in nind the following assumptions: (1) the storing of old data for RAID- 5 and the LSA directory for LSA are both assurned to take 17% of the cache, so only 83?0 of the cache is used for other purposes (2) schemes that store compressed data in cache have an effectkcly larger cache (3) RAID-5 has flat skew within an array but has some skew between arrays, LSA has no skew within or across arrays (4) Only LSA gets better transfer times due to compressed data on disk (5) RAID-5 has better seek affinity than LSA."

Menon, 175.

1.1 maintaining a list of boot data used for booting a computer system;	Menon, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.2 initializing a central processing unit of the computer system;	Menon, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.3 preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and	Menon, as evidenced by the example citations below, discloses "preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "The RAID controller is assumed to have a read and write cache built from Non-Volatile Storage (NVS)."

Menon, 167."

> "In LSA, data is stored on disks in compressed form. After a piece of data is updated, it may not compress as well as it did before it was updated, so it may not fit back into the space that had been allocated for it bcfore the update."

Menon, 169.

> "The record is compressed as soon as it reaches the subsystem. Compressed records are stored in the controller cache."

Menon, 169.

> "We assume a Non-Volatile cache and the use of Fast Write (as for

RAID-5)."

Menon, 169.

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

> "On a write hit or miss, the data block is accepted from the system and placed in the Non-Volatile cache after which the write is considered 'done'."

Menon, 170.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

1.4 servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.	Menon, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

> "A related parameter is the compression ratio. Improved compression ratios help in two ways. First, the better the compression ratio the greater the free space and, hence, the better the performance. Second, the better the compression ratio, the better the cache hit Iatios. This second factor aEects performance quite significantly, particularly at low 1/0 rates where

it has a bigger impact than the first factor."

Menon, 173.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

> "In understanding these comparisons, keep in nind the following assumptions: (1) the storing of old data for RAID- 5 and the LSA directory for LSA are both assurned to take 17% of the cache, so only 83?0 of the cache is used for other purposes (2) schemes that store compressed data in cache have an effectkcly larger cache (3) RAID-5 has flat skew within an array but has some skew between arrays, LSA has no skew within or across arrays (4) Only LSA gets better transfer times due to compressed data on disk (5) RAID-5 has better seek affinity than LSA."

Menon, 175.

> "Our results for IMS workloads may be summarized as follows. RAID-5 with compression has better response time and throughput than RAID-5 without compression, so we should try to do the former whenever possible."

Menon, 176.

2. The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.	Menon, as evidenced by the example citations below, discloses "wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

3. The method of claim 1, wherein the preloading is performed by a data storage controller connected to the boot device.	Menon, as evidenced by the example citations below, discloses "wherein the preloading is performed by a data storage controller connected to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the preloading is performed by a data storage controller connected to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

See Claims 1.3, and 1.4 above.

See also

> "In this paper, we compare the performance of the well-known RAIDS arrays to that of log-structured arrays (LSA), on transaction-processing workloads. LSA borrows heavily from the log-structured file system (LFS) approach, but is executed in an outboard disk controller."

Menon, 167.

> "The RAID controller is assumed to have a read and write cache built from Non-Volatile Storage (NVS)."

Menon, 167.

> "The array uses Fast Write; When a disk block to be written is received, the block is first stored in 2 separate NVS memory locations in the array controller (to avoid single points of failure). At this point, the disk array controller signals successful completion of the write to the host. Disk blocks in array controller cache memory that need to be written to disk are called dirty. Disk blocks in cache memory that are identical to their counterparts on disk are called clean."

Menon, 167. *See also* Menon, 167-167 (Analysis of Cached RAIDs), 168 (Disk, Controller, and Channel Parameters).

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the

controller cache; the requested record alone is decompressed and sent to the host."
Menon, 169.

"The memory segment is a section of controller memory, logically organized as N + 1 segment-columns called memory segment -columns; N data memory segment-columns and 1 parity memory segment-column."
Menon, 170.

"As for RAID-5, we assume that the controller consists of multiple (p) processors."

Menon, 170.

4. The method of claim 1, further comprising updating the list of boot data.	Menon, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

5. The method of claim 4, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.	Menon, as evidenced by the example citations below, discloses "wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1 and 4 above.	

6. The method of claim 4, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.	Menon, as evidenced by the example citations below, discloses "wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 4 above.	

7. (Preamble) A system for providing accelerated loading of an operating system of a host system comprising:	Menon, as evidenced by the example citations below, discloses "a system for providing accelerated loading of an operating system of a host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system for providing accelerated loading of an operating system of a host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1 (Preamble) above.	

7.1 a digital signal processor (DSP) or controller;	Menon, as evidenced by the example citations below, discloses "a digital signal processor (DSP) or controller."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a digital signal processor (DSP) or controller), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

See Claims 1.2, 1.3, and 1.4 above.

See also

> "In this paper, we compare the performance of the well-known RAIDS arrays to that of log-structured arrays (LSA), on transaction-processing workloads. LSA borrows heavily from the log-structured file system (LFS) approach, but is executed in an outboard disk controller."

Menon, 167.

> "The RAID controller is assumed to have a read and write cache built from Non-Volatile Storage (NVS)."

Menon, 167.

> "The array uses Fast Write; When a disk block to be written is received, the block is first stored in 2 separate NVS memory locations in the array controller (to avoid single points of failure). At this point, the disk array controller signals successful completion of the write to the host. Disk blocks in array controller cache memory that need to be written to disk are called dirty. Disk blocks in cache memory that are identical to their counterparts on disk are called clean."

Menon, 167. *See also* Menon, 167-167 (Analysis of Cached RAIDs), 168 (Disk, Controller, and Channel Parameters).

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."
> Menon, 169.

> "The memory segment is a section of controller memory, logically organized as N + 1 segment-columns called memory segment -columns; N data memory segment-columns and 1 parity memory segment-column."
>
> Menon, 170.
>
> "As for RAID-5, we assume that the controller consists of multiple (p) processors."
>
> Menon, 170.

7.2 a cache memory device; and;	Menon, as evidenced by the example citations below, discloses "a cache memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a cache memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

7.3.1 a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system	Menon, as evidenced by the example citations below, discloses "a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 1.3, 2, 3, and 7.1 above.	

7.3.2 for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system	Menon, as evidenced by the example citations below, discloses "for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3, and 1.4 above.	

7.3.3 and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system	Menon, as evidenced by the example citations below, discloses "decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3, and 1.4 above.	

8. The system of claim 7, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.	Menon, as evidenced by the example citations below, discloses "wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 1.3, 1.4, 2, 3, and 7 above.	

9.1 maintaining a list of application data associated with an application program;	Menon, as evidenced by the example citations below, discloses "maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 2, and 8 above.	

9.2 preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and	Menon, as evidenced by the example citations below, discloses "preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3, 2, and 8 above.	

9.3 servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.	Menon, as evidenced by the example citations below, discloses "servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.".
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.4, 2, and 8 above.	

10. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.	Menon, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "The record is compressed as soon as it reaches the subsystem. Compressed records are stored in the controller cache."

Menon, 169.

> "In LSA, data is stored on disks in compressed form. After a piece of data is updated, it may not compress as well as it did before it was updated, so it may not fit back into the space that had been allocated for it bcfore the update."

Menon, 169.

11. The method of claim 1, wherein the decompressing is provided by a data compression engine.	Menon, as evidenced by the example citations below, discloses "decompressing is provided by a data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing is provided by a data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

12. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.	Menon, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 10 and 11 above.	

13. The method of claim 1, wherein the compressed boot data is accessed via direct memory access.	Menon, as evidenced by the example citations below, discloses "the compressed boot data is accessed via direct memory access."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the compressed boot data is accessed via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4 above.	

15. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Menon, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4 above.	

16. The method of claim 1, wherein a plurality of encoders are utilized to provide the compressed boot data.	Menon, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3, 1.4, and 15 above.	

19. The method of claim 7, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Menon, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 15 above.	

20. The method of claim 7, wherein a plurality of encoders are utilized to provide the compressed boot data.	Menon, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 16 above.	

22.1 maintaining a list of boot data used for booting a computer system;.	Menon, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

22.2 initializing a central processing unit of the computer system;.	Menon, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

22.3 preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit;.	Menon, as evidenced by the example citations below, discloses "preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

22.4 servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data	Menon, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4 above.	

22.5 with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device.	Menon, as evidenced by the example citations below, discloses "with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 4, 10 and 11 above.	

24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Menon, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15 and 22 above.	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Menon, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 22 above.	

27.1 a boot device..	Menon, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1 above.	

27.2 a processor..	Menon, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

27.3 cache memory; and.	Menon, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4 above.	

27.4 non-volatile memory for storing logic code for use by the processor,..	Menon, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 and 1.3 above.	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Menon, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Menon, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Menon, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4 above.	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Menon, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 4, 10, and 11 above.	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Menon, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15 and 27 above.	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Menon, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 27 above.	

Appendix A36
Invalidity of U.S. Patent 7,181,608 based on Rubini

The publication Rubini, Booting the Kernel, Linux Journal, Jan. 1997, ("Rubini") invalidates claims 1-13, 15-16, 19-20, 22, 24-25, 27, and 29-30 of United States Patent No. 7,181,608 ("the '608 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '608 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix A36
Invalidity of U.S. Patent 7,181,608 based on Rubini

1 (Preamble) A method for providing accelerated loading of an operating system, comprising the steps of:	Rubini, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system, comprising the steps of:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

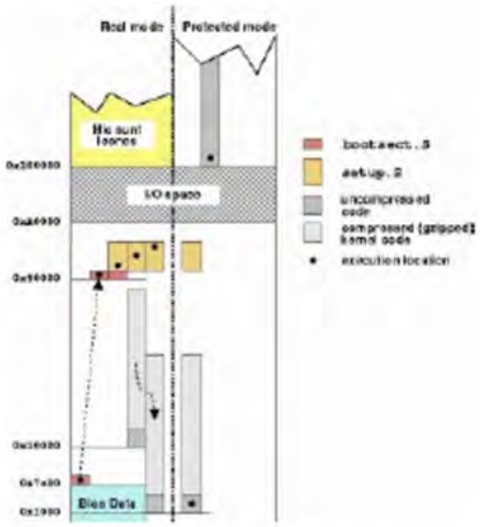


Figure 1. System Boot Data Map

Fig. 1.

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole

system." Rubini, 1.

1.1 maintaining a list of boot data used for booting a computer system;	Rubini, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory."

Rubini, 2.

1.2 initializing a central processing unit of the computer system;	Rubini, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards."

Rubini, 2.

1.3 preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and	Rubini, as evidenced by the example citations below, discloses "preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "To make things difficult, the PC firmware loads only half a kilobyte of code and establishes its own memory layout before loading this first sector. Whatever the boot media, the first sector of the boot partition is loaded into memory at the address 0x7c00, where execution begins. What happens at 0x7c00 depends on the boot loader being used; we examine three situations here: no boot-loader, LILO, Loadlin."

Rubini, 2.

> "Booting zImage and bzImage

> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.
>
> - The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
> - Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
> - Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
> - At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

> "The boot steps shown above rely on the assumption that the compressed kernel can fit in half a megabyte of space. While this is true most of the time, a system stuffed with device drivers might not fit into this space. For example, kernels used in installation disks can easily outgrow the available space. Some new method is needed to solve the problem—this new method is called bzImage and was introduced in kernel version

1.3.73.

A bzImage is generated by issuing make bzImage from the top level Linux source directory. This kind of kernel image boots similarly to zImage, with a few changes:

- When the system is loaded to address 0x10000, a little helper routine is called after loading each 64K data block. The helper routine moves the data block to high memory by using a special BIOS call. Only the newer BIOS versions implement this functionality, and so, make boot still builds the conventional zImage, though this may change in the near future.
- setup.S doesn't move the system back to 0x1000 (4K) but, after entering protected mode, jumps instead directly to address 0x100000 (1MB) where data has been moved by the BIOS in the previous step."

Rubini, 3-4.

"The rule for building the big compressed image can be read from Makefile; it affects several files in arch/i386/boot. One good point of bzImage is that when kernel/head.S is called, it doesn't notice the extra work, and everything goes forward as usual."

Rubini, 4.

1.4 servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.	Rubini, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path

names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

> "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."

Rubini, 4.

2. The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.	Rubini, as evidenced by the example citations below, discloses "wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

3. The method of claim 1, wherein the preloading is performed by a data storage controller connected to the boot device.	Rubini, as evidenced by the example citations below, discloses "wherein the preloading is performed by a data storage controller connected to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the preloading is performed by a data storage controller connected to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3, and 1.4 above.	

4. The method of claim 1, further comprising updating the list of boot data.	Rubini, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claim 1.1 above.

> "Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory."

Rubini, 2.

5. The method of claim 4, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.	Rubini, as evidenced by the example citations below, discloses "wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1 and 4 above.	

6. The method of claim 4, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.	Rubini, as evidenced by the example citations below, discloses "wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1 and 4 above.	

Appendix A36
Invalidity of U.S. Patent 7,181,608 based on Rubini

7. (Preamble) A system for providing accelerated loading of an operating system of a host system comprising:	Rubini, as evidenced by the example citations below, discloses "a system for providing accelerated loading of an operating system of a host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system for providing accelerated loading of an operating system of a host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1 (Preamble) above.	

7.1 a digital signal processor (DSP) or controller;	Rubini, as evidenced by the example citations below, discloses "a digital signal processor (DSP) or controller."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a digital signal processor (DSP) or controller), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.2, 1.3, and 1.4 above.	

7.2 a cache memory device; and;	Rubini, as evidenced by the example citations below, discloses "a cache memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a cache memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

7.3.1 a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system	Rubini, as evidenced by the example citations below, discloses "a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 1.3, 2, 3, and 7.1 above.	

7.3.2 for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system	Rubini, as evidenced by the example citations below, discloses "for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3, and 1.4 above.	

7.3.3 and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system	Rubini, as evidenced by the example citations below, discloses "decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3, and 1.4 above.	

8. The system of claim 7, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.	Rubini, as evidenced by the example citations below, discloses "wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 1.3, 1.4, 2, 3, and 7 above.	

9.1 maintaining a list of application data associated with an application program;	Rubini, as evidenced by the example citations below, discloses "maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 2, and 8 above.	

9.2 preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and	Rubini, as evidenced by the example citations below, discloses "preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3, 2, and 8 above.	

9.3 servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.	Rubini, as evidenced by the example citations below, discloses "servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.".
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.4, 2, and 8 above.	

10. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.	Rubini, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

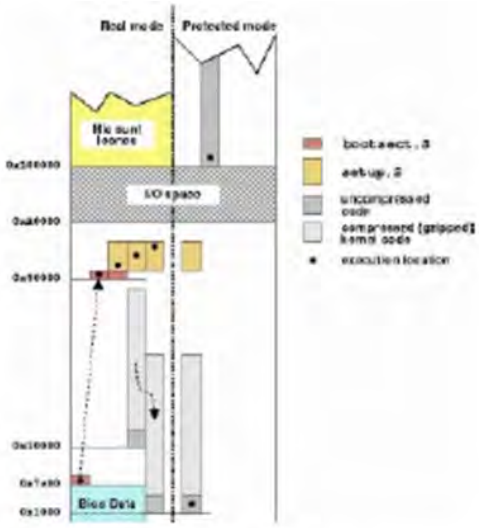


Figure 1. System Boot Data Map

Fig. 1.

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part

(or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

"The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory."

Rubini, 2.

"The boot steps shown above rely on the assumption that the compressed kernel can fit in half a megabyte of space. While this is true most of the time, a system stuffed with device drivers might not fit into this space. For example, kernels used in installation disks can easily outgrow the available space. Some new method is needed to solve the problem—this new method is called bzImage and was introduced in kernel version 1.3.73.

A bzImage is generated by issuing make bzImage from the top level Linux source directory. This kind of kernel image boots similarly to zImage, with a few changes:

- When the system is loaded to address 0x10000, a little helper routine is called after loading each 64K data block. The helper routine moves the data block to high memory by using a special BIOS call. Only the newer BIOS versions implement this functionality, and so, make boot still builds the conventional zImage, though this may change in the near future.
- setup.S doesn't move the system back to 0x1000 (4K) but, after entering protected mode, jumps instead directly to address 0x100000 (1MB) where data has been moved by the BIOS in the previous step."

Rubini, 3-4.

"The rule for building the big compressed image can be read from Makefile; it affects several files in arch/i386/boot. One good point of bzImage is that when kernel/head.S is called, it doesn't notice the extra work, and everything goes forward as usual."

Rubini, 4.

11. The method of claim 1, wherein the decompressing is provided by a data compression engine.	Rubini, as evidenced by the example citations below, discloses "decompressing is provided by a data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing is provided by a data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> "The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.
>
> - The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a

megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.

- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

  The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

"The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the

address 0x100000 (1MB). Several memory moves are needed to perform the task correctly." Rubini, 4.

12. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.	Rubini, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 10 and 11 above.	

13. The method of claim 1, wherein the compressed boot data is accessed via direct memory access.	Rubini, as evidenced by the example citations below, discloses "the compressed boot data is accessed via direct memory access."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the compressed boot data is accessed via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4 above.	

15. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Rubini, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4 above.	

16. The method of claim 1, wherein a plurality of encoders are utilized to provide the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3, 1.4, and 15 above.	

19. The method of claim 7, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Rubini, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 15 above.	

20. The method of claim 7, wherein a plurality of encoders are utilized to provide the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 16 above.	

22.1 maintaining a list of boot data used for booting a computer system;.	Rubini, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

22.2 initializing a central processing unit of the computer system;.	Rubini, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

22.3 preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit;.	Rubini, as evidenced by the example citations below, discloses "preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

22.4 servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data	Rubini, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4 above.	

22.5 with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device.	Rubini, as evidenced by the example citations below, discloses "with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 4, 10 and 11 above.	

24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Rubini, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15 and 22 above.	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Rubini, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 22 above.	

27.1 a boot device..	Rubini, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1 above.	

27.2 a processor..	Rubini, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

27.3 cache memory; and.	Rubini, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4 above.	

27.4 non-volatile memory for storing logic code for use by the processor,..	Rubini, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 and 1.3 above.	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Rubini, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Rubini, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Rubini, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4 above.	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Rubini, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 4, 10, and 11 above.	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Rubini, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15 and 27 above.	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Rubini, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 27 above.	

Appendix A37
Invalidity of U.S. Patent 7,181,608 based on Wynn

Wynn, et al. "The effect of compression on performance in a demand paging operating system," The Journal of Systems and Software (2000) ("Wynn Article") and Wynn, "The Effect of Compression on Performance in a Demand Paging Operating System," 1997 ("Wynn Thesis") (collectively, "Wynn"), alone or in combination, invalidate claims 1-13, 15-16, 19-20, 22, 24-25, 27, and 29-30 of United States Patent No. 7,181,608 ("the '608 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '608 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix A37
Invalidity of U.S. Patent 7,181,608 based on Wynn

1 (Preamble) A method for providing accelerated loading of an operating system, comprising the steps of:	Wynn, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system, comprising the steps of:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page so that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance overall is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

Appendix A37
Invalidity of U.S. Patent 7,181,608 based on Wynn

1.1 maintaining a list of boot data used for booting a computer system;	Wynn, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application

5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Article, 3.8

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9

1.2 initializing a central processing unit of the computer system;	Wynn, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page.

Wynn Article, Abstract

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Article, 3.8

Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in I 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.

We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

Wynn Article, 3.9

We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9

Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

Wynn Thesis, 5.10

1.3 preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and	Wynn, as evidenced by the example citations below, discloses "preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance overall is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

Based on these timings, it is clear that the time required to load a page with the

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

1.4 servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.	Wynn, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2

Wynn Claim 1.4

"servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

Based on these timings, it is clear that the time required to load a page with the

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

2. The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.	Wynn, as evidenced by the example citations below, discloses "wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 4.3

> Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

> Our second criterion, just as important as the first, is decompression rate. A high decompression rate is essential in order to improve the performance of the page fault path. The compression rate is of lessor importance because the compression will occur at link time. An application developer may see a negative performance impact when building his executable, however the user of the executable should receive a performance benefit. However, because we are interested in application run time, the speed of decompression is of much greater importance.

Wynn Article, 3.2

3.8. Test sequence

We created a test sequence which would approximate the sequence of events followed by a typical user. The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application and realtime video, are opened. The applications are executed, then the folders and applications are closed.

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application

Wynn Article, 3.8

4.3. Effect of page-based compression on performance

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 1.2

Wynn Claim 2

"The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."

> Applications load time is typically faster on operating systems that use virtual memory and demand paging. Present-day operating systems will, in general, not preload all of the code and data for an application at application load time. Smaller sections of code and data are loaded into memory as they are required by the application. The amount of code and data required at application load time is usually very much smaller than the total amount of code and data in the application. In fact, large sections of code and data may not be required by the application at all during a particular session.

Wynn Thesis, 2.1

3. The method of claim 1, wherein the preloading is performed by a data storage controller connected to the boot device.	Wynn, as evidenced by the example citations below, discloses "wherein the preloading is performed by a data storage controller connected to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the preloading is performed by a data storage controller connected to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.3, and 1.4 above.

When the number of virtual pages actively in use exceeds the number of page frames (pages of physical memory), it becomes necessary to remove some pages from physical memory to make room for new pages. Some operating systems, such as OSF/1 and 4.4BSD, maintain a separate process to select candidates for page replacement and perform the write to secondary storage [OSF93],[MCKU96], while others, such as OS/2, maintain a separate process to select candidates for page replacement, but defer the write to secondary storage until a page-fault occurs [DEIT92]. The separate process is commonly known as a page-stealer, a page-out daemon, or an ager.

Wynn Thesis, 2.3

In order to measure the effects of page-based compression on a demand-paging operating system, we modified the OS2KRNL (kernel) to maintain information on the pages which are being page-faulted into memory and the pages which are selected as victims for page-replacement.

Wynn Article, 3.6

4. The method of claim 1, further comprising updating the list of boot data.	Wynn, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claim 1.1 above.

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Claim 4

"The method of claim 1, further comprising updating the list of boot data."

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

Based on these timings, it is clear that the time required to load a page with the

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

5. The method of claim 4, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.	Wynn, as evidenced by the example citations below, discloses "wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1 and 4 above.	

6. The method of claim 4, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.	Wynn, as evidenced by the example citations below, discloses "wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1 and 4 above.	

7. (Preamble) A system for providing accelerated loading of an operating system of a host system comprising:	Wynn, as evidenced by the example citations below, discloses "a system for providing accelerated loading of an operating system of a host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system for providing accelerated loading of an operating system of a host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1 (Preamble) above.	

7.1 a digital signal processor (DSP) or controller;	Wynn, as evidenced by the example citations below, discloses "a digital signal processor (DSP) or controller."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a digital signal processor (DSP) or controller), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.2, 1.3, and 1.4 above.

When the number of virtual pages actively in use exceeds the number of page frames (pages of physical memory), it becomes necessary to remove some pages from physical memory to make room for new pages. Some operating systems, such as OSF/1 and 4.4BSD, maintain a separate process to select candidates for page replacement and perform the write to secondary storage [OSF93],[MCKU96], while others, such as OS/2, maintain a separate process to select candidates for page replacement, but defer the write to secondary storage until a page-fault occurs [DEIT92]. The separate process is commonly known as a page-stealer, a page-out daemon, or an ager.

Wynn Thesis, 2.3

In order to measure the effects of page-based compression on a demand-paging operating system, we modified the OS2KRNL (kernel) to maintain information on the pages which are being page-faulted into memory and the pages which are selected as victims for page-replacement.

Wynn Article, 3.6

7.2 a cache memory device; and;	Wynn, as evidenced by the example citations below, discloses "a cache memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a cache memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

7.3.1 a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system	Wynn, as evidenced by the example citations below, discloses "a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1, 1.3, 2, 3, and 7.1 above.	

7.3.2 for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system	Wynn, as evidenced by the example citations below, discloses "for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3, and 1.4 above.	

7.3.3 and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system	Wynn, as evidenced by the example citations below, discloses "decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3, and 1.4 above.	

8. The system of claim 7, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.	Wynn, as evidenced by the example citations below, discloses "wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1, 1.3, 1.4, 2, 3, and 7 above.

> We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 4.3

> Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

> Our second criterion, just as important as the first, is decompression rate. A high decompression rate is essential in order to improve the performance of the page fault path. The compression rate is of lessor importance because the compression will occur at link time. An application developer may see a negative performance impact when building his executable, however the user of the executable should receive a performance benefit. However, because we are interested in application run time, the speed of decompression is of much greater importance.

Wynn Article, 3.2

Appendix A37
Invalidity of U.S. Patent 7,181,608 based on Wynn

3.8. Test sequence

We created a test sequence which would approximate the sequence of events followed by a typical user. The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application and realtime video, are opened. The applications are executed, then the folders and applications are closed.

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application

Wynn Article, 3.8

4.3. Effect of page-based compression on performance

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based comp (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these the time required by the system to process the page faults during system boot is shown in Table 4. The actu required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The im point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in imately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The g Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve perfo at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 1.2

> Applications load time is typically faster on operating systems that use virtual memory and demand paging. Present-day operating systems will, in general, not preload all of the code and data for an application at application load time. Smaller sections of code and data are loaded into memory as they are required by the application. The amount of code and data required at application load time is usually very much smaller than the total amount of code and data in the application. In fact, large sections of code and data may not be required by the application at all during a particular session.

Wynn Thesis, 2.1

9.1 maintaining a list of application data associated with an application program;	Wynn, as evidenced by the example citations below, discloses "maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1, 2, and 8 above.	

9.2 preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and	Wynn, as evidenced by the example citations below, discloses "preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3, 2, and 8 above.	

9.3 servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.	Wynn, as evidenced by the example citations below, discloses "servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.".

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.4, 2, and 8 above.

We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 4.3

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

Our second criterion, just as important as the first, is decompression rate. A high decompression rate is essential in order to improve the performance of the page fault path. The compression rate is of lessor importance because the compression will occur at link time. An application developer may see a negative performance impact when building his executable, however the user of the executable should receive a performance benefit. However, because we are interested in application run time, the speed of decompression is of much greater importance.

Wynn Article, 3.2

3.8. Test sequence

We created a test sequence which would approximate the sequence of events followed by a typical user. The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application and realtime video, are opened. The applications are executed, then the folders and applications are closed.

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application

Wynn Article, 3.8

4.3. Effect of page-based compression on performance

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based comp (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these the time required by the system to process the page faults during system boot is shown in Table 4. The actu required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The im point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in imately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The g Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve perfo at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

We have found that the time saved due to compression offsets the additional time caused by the increase i of page faults, even in a memory constrained environment. Hence, page-based compression improves the per of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 1.2

> Applications load time is typically faster on operating systems that use virtual memory and demand paging. Present-day operating systems will, in general, not preload all of the code and data for an application at application load time. Smaller sections of code and data are loaded into memory as they are required by the application. The amount of code and data required at application load time is usually very much smaller than the total amount of code and data in the application. In fact, large sections of code and data may not be required by the application at all during a particular session.

Wynn Thesis, 2.1

10. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.	Wynn, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Article, 3.8

Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in I 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.

We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

Wynn Article, 3.9

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9

Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

Wynn Thesis, 5.10

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

11. The method of claim 1, wherein the decompressing is provided by a data compression engine.	Wynn, as evidenced by the example citations below, discloses "decompressing is provided by a data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing is provided by a data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page.

Wynn Article, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2

In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn — Claim 11

"The method of claim 1, wherein the decompressing is provided by a data compression engine."

Wynn Thesis, Abstract

OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

Based on these timings, it is clear that the time required to load a page with the

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

12. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.	Wynn, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 10 and 11 above.	

13. The method of claim 1, wherein the compressed boot data is accessed via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the compressed boot data is accessed via direct memory access."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the compressed boot data is accessed via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.3 and 1.4 above.

1.1. Motivation

As microprocessors increase in speed many traditional compute-constrained tasks are transformed into Input/Output (I/O) bound task. Access to secondary memory or direct access storage devices (DASD) is quickly becoming the primary bottleneck for many computer Systems. Engineers continue to make great strides in processor speed, the processing power of the Intel X86 family of processors has increased 335 times between 1978 and 1993 (Wirth, 1995). Such amazing increase in performance has not been seen in DASD, however. As the performance gap widens, system performance will tend to become bounded by I/O performance. This is similar to Amdahl's Law: The small DASD-access component of a problem will limit the speed-up attainable by increasing processor speed. Thus, it is becoming much more important to understand the dynamics of DASD performance if we are to improve overall system performance.

Wynn Article, 1.1

If the time required to read a compressed page from DASD and decompress it is less than the time required to read an uncompressed page from DASD, then page-based compression may improve the performance of a demand paging operating system. However, more memory is required for compressed page because an additional buffer is needed. In a memory constrained environment, removing a minimal amount of memory could cause performance degradation.

Wynn Article, 1.2

15. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Wynn, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.3 and 1.4 above.

5.1. Summary

An overview of compression and lossless and lossy algorithms was presented in Section 2. LZ77 and LZ78 and variants were explored in detail. The OS/2 operating system application loading and page-replacement policy, including aging and the page fault process were also described.

Our approach was described in Section 3. The Exepac compression format, which is based off LZ77 and variants, was described in detail. Modifications to utilities and to the operating system were listed, as well as the hardware setup, software setup and test sequence.

The results of this study were detailed in Section 4. First, the effect of page-based compression on the page fault rate are analyzed, then the effect of page-based compression on performance is analyzed. Finally, the effect of page-based compression on the scalability of the system is provided.

Wynn Article, 5.1

Section 3 describes the approach used in this study. The Exepack2 compression format, which is based off LZ77 and variants, is described in detail. Utility and operating system modifications are listed, as well as the hardware setup, software setup and test sequence.

Wynn Article, 1.2

Dictionary-based compression algorithms can have either a static dictionary or an adaptive dictionary. Most static dictionary-based compression schemes are built for a specific application, and are not general purpose. An example would be an inventory system with dictionary phrases like 'desk', 'chair', 'table' and 'vacuum cleaner'. The dictionary would be created before compression occurs. Static dictionary-based schemes can tune their dictionary in advance. More frequently used phrases like 'chair' would be assigned fewer bits, while less frequently used phrases like 'vacuum cleaner' would be assigned more bits. However, the dictionary must be available to both the encoder and the decoder. In some instances, this may mean transmitting or storing the dictionary along with the compressed data. If the data block is sufficiently large, then the overhead of storing the dictionary may be tolerable.

Adaptive dictionary-based compression algorithms start with either no dictionary or with a general default dictionary. As the compression algorithm sees the input data, it adds new phrases to the dictionary.

Most adaptive dictionary-based compression algorithms are based on two papers presented in 1977 and 1978 by Ziv and Lempel (1977, 1978). Ziv and Lempel (1977) paper describes a sliding-window technique (LZ77) in which the dictionary consists of the phrases found in a window, or section, of the previously seen data. The window 'slides' across the previously seen input data so that the most recently compressed data are always contained within the window. LZ77 also manages a look-ahead buffer, which can be viewed as a window into the input data containing the next characters to be encoded. As data are compressed, the uncompressed characters slide out of the look-ahead buffer and into the dictionary.

Tokens in the LZ77 compression format consist of three parts, an offset back into the dictionary, a phrase length, and the first character in the look-ahead buffer after the phrase. It is clear to see that pointers and phrases must be alternated.

Wynn Article, 2.1

Decoding with LZ77 is quite simple. For each token, the decoder uses the offset field to index into the window of previously decompressed text in the output buffer. It then copies the number of bytes specified by the length field from that location to the current location in the output buffer. Finally, the character is copied from the token to the output buffer.

The two major tunable parameters for LZ77 compression are the size of the dictionary and the size of the look-ahead buffer (the maximum phrase length). The average time to find the longest matching phrase is of the order of the product of the dictionary size and the phrase length. Thus, doubling the size of the dictionary or the look-ahead buffer has a similar doubling effect on the compression time. Decompression time is not significantly affected by increasing either the dictionary or the maximum phrase length.

The LZSS compression algorithm was a well-received extension of LZ77. LZSS has two primary enhancements to LZ77. First, LZSS adds a tree structure on top of the dictionary. If the tree is implemented as a binary tree, then the average time to find the longest matching string will be of the order of the base 2 logarithm of the dictionary size times the phrase length, which is far less than required by LZ77. Also, larger dictionary sizes can be used because doubling the size of the dictionary or the look-ahead buffer will cause a small impact to the compression time rather than doubling it.

Wynn Article, 2.1

As discussed in Chapter 3, compression techniques which remove repeating characters or strings, such as null suppression, work well on data but in general do not work well on code. Statistical modeling techniques do not have the decompression speed required at page fault time. The one choice that is left is dictionary-based compression. LZ77 and variants tend to have the faster decompression rates than LZ78 and variants.

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

This particular compression algorithm was chosen because it met all three of our criteria. First, the compression ratio averages 56 percent for pages of code and data, with slightly better compression for data than for code. Also, the decompression rates for LZ77 and derivatives is extremely high. Decompressing does not require a lot of processing, typically the algorithm will repeatedly set a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

Wynn Thesis, 5.2

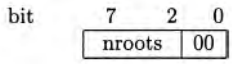
16. The method of claim 1, wherein a plurality of encoders are utilized to provide the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

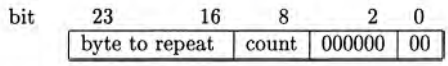
Wynn discloses this limitation:

See Claims 1.3, 1.4, and 15 above.

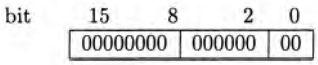
The nRoots token is indicated by a 00 prefix, and can take three forms. The first, and simplest form is a Pure Root token. Bits 2 through 7 specify the number of uncompressed bytes that follow the token header in the compressed data. This field is the nroots field. A value of zero is invalid, so up to 63 uncompressed bytes can be encoded by one token. The primary advantage of the Pure Root token is encoding runs of incompressible bytes, especially when compression is starting and the dictionary is nearly empty. The format of the Pure Root token is the following:



The second form of the nRoots token is a Repeating Bytes Root token, which is indicated by a zero value for bits 2 through 7, which is an invalid value for the Pure Roots token. The third byte in the token is the byte which will be repeated. The second byte of the token is the number of times to repeat the byte, which can be a value from 1 to 255. A value of zero is invalid. The Repeating Bytes Root token provides the advantage of run-length encoding when a character is repeated many times, as in zero-initialized data. The format of the Repeating Bytes Root token is the following:



The third form of the nRoots record is the End Code Root token, which is indicated by a zero value in bits 2 through 7 and a zero value for the second byte. Providing an end code allows more streamline (i.e., faster) decompression, without having to check for an end-of-data condition after each token. The End Code Root token is three bytes long, which is acceptable because there will be only one end code for each compressed block of data. The format of the End Code Root token is the following:



Wynn Article, 3.2

> The first field describes the number of times that the data is to be repeated. The data length field describes the number of bytes in the data which follows. The data bytes field is the data which is to be repeated. Iteration encoding differs from the simper run-length encoding in that iteration records can represent repeating patterns of bytes, while the simpler run-length encoding can only represent repeating bytes.

Wynn Thesis, 5.3

19. The method of claim 7, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Wynn, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 15 above.	

20. The method of claim 7, wherein a plurality of encoders are utilized to provide the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 16 above	

22.1 maintaining a list of boot data used for booting a computer system;.	Wynn, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above	

22.2 initializing a central processing unit of the computer system;.	Wynn, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above	

22.3 preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit;.	Wynn, as evidenced by the example citations below, discloses "preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above	

22.4 servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data	Wynn, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4 above	

22.5 with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device.	Wynn, as evidenced by the example citations below, discloses "with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 4, 10 and 11 above	

24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Wynn, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15 and 22 above	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Wynn, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 22 above	

27.1 a boot device..	Wynn, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1 above	

27.2 a processor..	Wynn, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above	

27.3 cache memory; and.	Wynn, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3 and 1.4 above	

27.4 non-volatile memory for storing logic code for use by the processor,..	Wynn, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 and 1.3 above	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Wynn, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Wynn, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Wynn, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4 above	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Wynn, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 4, 10, and 11 above	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Wynn, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15 and 27 above	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Wynn, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 27 above	

Appendix A38
Invalidity of U.S. Patent 7,181,608 based on Linux Kernel

Red Hat Linux 5.0, The Official Red Hat Linux Installation Guide (1995) ("Linux Redhat") and M. Beck, et. al, "Linux Kernel Internals" Addison Wesley Longman (1996) ("Beck") (collectively, "Linux Kernel"), alone or in combination, invalidate claims 1-13, 15-16, 19-20, 22, 24-25, 27, and 29-30 of United States Patent No. 7,181,608 ("the '608 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '608 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1 (Preamble) A method for providing accelerated loading of an operating system, comprising the steps of:	Linux Kernel, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system, comprising the steps of:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

1.1 maintaining a list of boot data used for booting a computer system;	Linux Kernel, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

```
rm -rf /lib/modules/2.0.29-old
mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.

- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

D.3 Installation Overview

Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

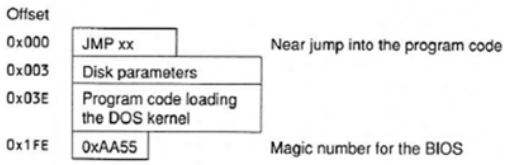


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

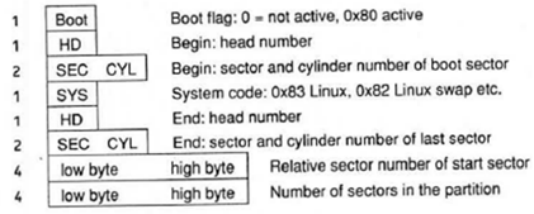


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1

The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

1.2 initializing a central processing unit of the computer system;	Linux Kernel, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

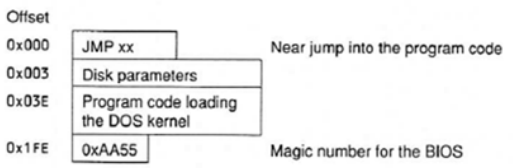


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

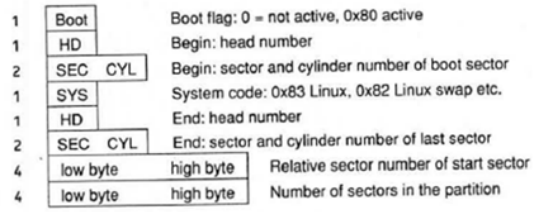


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

1.3 preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and	Linux Kernel, as evidenced by the example citations below, discloses "preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

```
rm -rf /lib/modules/2.0.29-old
mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.

- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

D.3 Installation Overview

Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word.*

The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

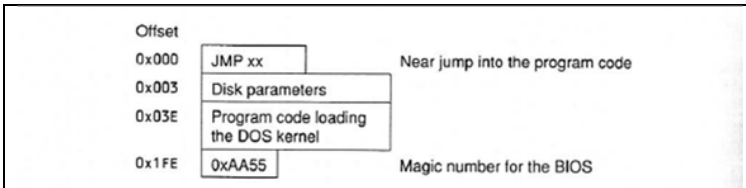


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

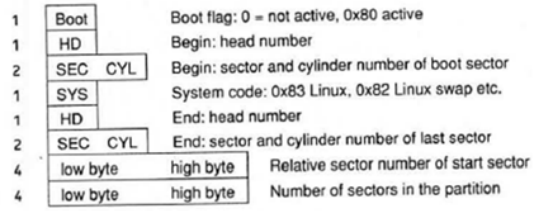


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1

The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

`boot=`*device* indicates which device (or which disk partition) shall contain the boot sector. If **`boot`** is not present, the boot sector is put on the current root device.

`compact` activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

1.4 servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.	Linux Kernel, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

Linux Kernel Claim 1.4

"servicing requests for boot data from the computer system using the preloaded boot data after completion of the initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache."

D.3 Installation Overview

Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

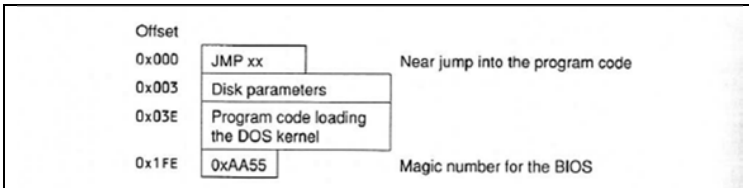


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

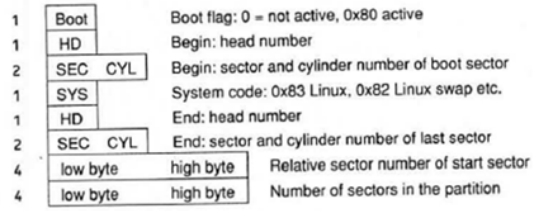


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1

The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

`boot=`*device* indicates which device (or which disk partition) shall contain the boot sector. If **`boot`** is not present, the boot sector is put on the current root device.

`compact` activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

2. The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

3. The method of claim 1, wherein the preloading is performed by a data storage controller connected to the boot device.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the preloading is performed by a data storage controller connected to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the preloading is performed by a data storage controller connected to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3, and 1.4 above.	

4. The method of claim 1, further comprising updating the list of boot data.	Linux Kernel, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claim 1.1 above.

See also

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

```
rm -rf /lib/modules/2.0.29-old
mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.

- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

In the text of each section, names of source text commands, variables and so on, are set in a **`display`** font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs **`GZIP.EXE`** and **`RAWRITE.EXE`**, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

5. The method of claim 4, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, boot data that comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1 and 4 above.	

6. The method of claim 4, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 4 above.	

7. (Preamble) A system for providing accelerated loading of an operating system of a host system comprising:	Linux Kernel, as evidenced by the example citations below, discloses "a system for providing accelerated loading of an operating system of a host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system for providing accelerated loading of an operating system of a host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1 (Preamble) above.	

7.1 a digital signal processor (DSP) or controller;	Linux Kernel, as evidenced by the example citations below, discloses "a digital signal processor (DSP) or controller."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a digital signal processor (DSP) or controller), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.2, 1.3, and 1.4 above.	

7.2 a cache memory device; and;	Linux Kernel, as evidenced by the example citations below, discloses "a cache memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a cache memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

7.3.1 a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system	Linux Kernel, as evidenced by the example citations below, discloses "a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 1.3, 2, 3, and 7.1 above.	

7.3.2 for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system	Linux Kernel, as evidenced by the example citations below, discloses "for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3, and 1.4 above.	

7.3.3 and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3, and 1.4 above.	

8. The system of claim 7, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 1.3, 1.4, 2, 3, and 7 above.	

9.1 maintaining a list of application data associated with an application program;	Linux Kernel, as evidenced by the example citations below, discloses "maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 2, and 8 above.	

9.2 preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and	Linux Kernel, as evidenced by the example citations below, discloses "preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3, 2, and 8 above.	

9.3 servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.	Linux Kernel, as evidenced by the example citations below, discloses "servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.".
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.4, 2, and 8 above.	

10. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.	Linux Kernel, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

D.3 Installation Overview

Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Linux Kernel Claim 10

"The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device"

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

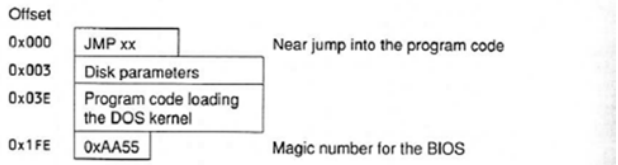


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

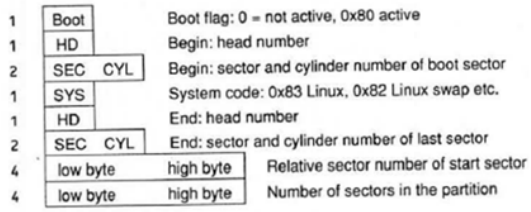


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1

The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

Linux Kernel Claim 10

"The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device"

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

11. The method of claim 1, wherein the decompressing is provided by a data compression engine.	Linux Kernel, as evidenced by the example citations below, discloses "decompressing is provided by a data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing is provided by a data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1

6.1.1 Building a modularized kernel

Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1

In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1

D.3 Installation Overview

Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3

D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2

D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

Linux Redhat, at Appendix D5.5

E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:

```
MILO> boot [-t file-system] device-name:file-name \
                [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

The actual compilation of the kernel now begins, with the simple call:

```
# make
```

After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2

For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

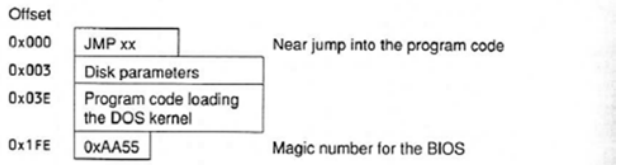


Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

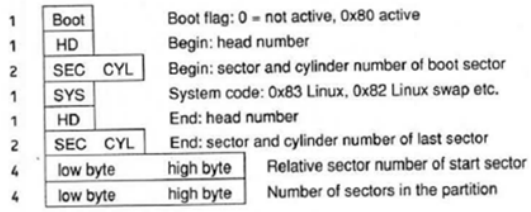


Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1

The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

12. The method of claim 1, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.	Linux Kernel, as evidenced by the example citations below, discloses "a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 10 and 11 above.	

13. The method of claim 1, wherein the compressed boot data is accessed via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the compressed boot data is accessed via direct memory access."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the compressed boot data is accessed via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4 above.	

15. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Linux Kernel, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claims 1.3 and 1.4 above.

See also

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

16. The method of claim 1, wherein a plurality of encoders are utilized to provide the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claims 1.3, 1.4, and 15 above.

See also

In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

19. The method of claim 7, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data..	Linux Kernel, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 15 above.	

20. The method of claim 7, wherein a plurality of encoders are utilized to provide the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4, 7, and 16 above	

22.1 maintaining a list of boot data used for booting a computer system;.	Linux Kernel, as evidenced by the example citations below, discloses "maintaining a list of boot data used for booting a computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above	

22.2 initializing a central processing unit of the computer system;.	Linux Kernel, as evidenced by the example citations below, discloses "initializing a central processing unit of the computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above	

22.3 preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit;.	Linux Kernel, as evidenced by the example citations below, discloses "preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading boot data in compressed form, based on the list of boot data, from a boot device into a cache memory prior to completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above	

22.4 servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data	Linux Kernel, as evidenced by the example citations below, discloses "servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for boot data from the computer system using the preloaded compressed boot data after completion of initialization of the central processing unit, wherein servicing requests comprises accessing the compressed boot data from the cache and decompressing the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4 above	

22.5 with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device.	Linux Kernel, as evidenced by the example citations below, discloses "with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, with a data compression engine and the data compression engine being operable to compress additional boot data and store the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 4, 10 and 11 above	

24. The method of claim 22, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15 and 22 above	

25. The method of claim 22, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data..	Linux Kernel, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 22 above	

27.1 a boot device..	Linux Kernel, as evidenced by the example citations below, discloses "a boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 above	

27.2 a processor..	Linux Kernel, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above	

27.3 cache memory; and.	Linux Kernel, as evidenced by the example citations below, discloses "cache memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, cache memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4 above	

27.4 non-volatile memory for storing logic code for use by the processor,..	Linux Kernel, as evidenced by the example citations below, discloses "non-volatile memory for storing logic code for use by the processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, non-volatile memory for storing logic code for use by the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 and 1.3 above	

27.5 the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system	Linux Kernel, as evidenced by the example citations below, discloses "the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the logic code being used for: maintaining a list associated with boot data, wherein the boot data is used in booting a first system;), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above	

27.6 preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system; and	Linux Kernel, as evidenced by the example citations below, discloses "preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading compressed boot data associated to the list into the cache memory prior to completion of initialization of a central processing unit of the first system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above	

27.7 servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit; and	Linux Kernel, as evidenced by the example citations below, discloses "servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing requests for the compressed boot data from the first system after completion of initialization of the central processing unit), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4 above	

27.8 a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device	Linux Kernel, as evidenced by the example citations below, discloses "a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for decompressing the compressed boot data accessed from the cache memory for use in responding to the servicing requests and for compressing additional boot data and storing the additional compressed boot data to the boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 4, 10, and 11 above	

29. The system of claim 27, wherein Lempel-Ziv is utilized by the data compression engine to compress the additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "Lempel-Ziv is utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv is utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15 and 27 above	

30. The system of claim 27, wherein a plurality of encoders are utilized by the data compression engine to compress the additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "a plurality of encoders are utilized by the data compression engine to compress the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized by the data compression engine to compress the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 27 above	

Appendix B1
Invalidity of U.S. Patent 8,090,936 based on Esfahani

U.S. Patent Nos. 6,434,695 to Esfahani ("Esfahani '695) and 6,732,265 to Esfahani ("Esfahani '265"), alone or in combination, invalidate claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

In addition, the prior art New World Mac operating system disclosed in Esfahani ("New World Mac System") invalidates claims 1-6, 8-9, 11-13, and 15-16 of the '936 Patent pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 because the system was on sale or in public use more than one year prior to the filing date of the '936 patent. Esfahani '695, Esfahani '265, and the New World Mac System are collectively referred to herein as "Esfahani."

Additionally, Apple intends to provide and seek discovery related to this system to obtain copies of relevant materials, including but not limited to, architectural documents, design documents, implementation documents, internal publications, patent applications and business records relating to this product and will supplement these contentions after those materials are discovered or received. Apple also reserves its rights to rely on any additional New World Mac System materials, either individually or collectively, as prior art publications to the '936 patent.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Esfahani, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

> A computer OS using a compressed ROM image in RAM is described. In brief, the low-level portion of an OS of a computer is designed to be separate from the intermediate-level portion of the OS. The low-level portion, which includes hardware-specific code, is stored in a relatively small Boot ROM, while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device. The mass storage device may be located remotely from the computer system, such as in a file server. Upon power-up or reset of the computer system, the code in the boot ROM is executed to read the compressed ROM image into RAM, i.e., system memory, of the computer system. The compressed image is then decompressed and executed as part of the boot sequence.

Esfahani 695, 2:54-67; *also* Esfahani '265, 2:58-3:4.

> Referring now to FIG. 4, in the improved OS, the low-level (hardware-specific) OS code 31 resides in firmware, in order to handle start-up activities of the computer system. This code fits into one, relatively small ROM, referred to as the Boot ROM 11. Thus, Boot ROM 11 includes all of the hardware-specific code and tables needed to start up the computer as well as to boot the OS and provide common hardware access services the OS might require. Note that the Boot ROM code is not specific to the MacOS or to any other OS. All higher-level software resides elsewhere, as will now be described.

> Prior to start-up, the mid-level portion 32 of OS 30 (which corresponds to part of the ToolBox ROM of earlier Macintosh computers) resides in compressed form in a file 40, referred to as Boot Info file 40.

Esfahani ’695, 5:7-23; *also* Esfahani ’265, 5:10-24.

> During start-up, the Boot Info file 40 is loaded into RAM 12, and the compressed mid-level OS 32 is decompressed. Hence, the mid-level OS 32 is essentially a compressed ROM image. The mid-level OS 32 is inserted into the memory map of the computer system as if it were firmware in ROM. That is, the ROM image can be write-protected in the memory map.

Esfahani ’695, 5:44-51; *also* Esfahani ’265, 5:48-54.

> Referring now to FIG. 5, the low-level OS portion 31 and the Boot Info file 40 are illustrated in greater detail. The low-level portion 31 stored in Boot ROM 11 contains the code needed to start up the computer, initialize and examine the hardware, provide a Device Tree (per Open Firmware) to describe the hardware, provide hardware access services, and transfer control to the OS. In one embodiment, the components of the low-level portion 31 include:
>
> > code for performing Power-On Self Test (POST), including code for performing diagnostics, generating a boot beep and an error beep;
> >
> > Open Firmware code;
> >
> > hardware-specific Mac OS drivers (“ndrv's”) that are needed at boot time (drivers needed at boot time, e.g.,video drivers, network drivers, or disk drivers, are loaded from the Device Tree);
> >
> > HardwareInit code (i.e., the lowest-level code for initializing the CPU, RAM, clock, system bus, etc.) without Mac OS-specific code;
> >
> > code for performing Run-Time Abstraction Services (RTAS). Certain hardware devices differ from machine to machine, but provide similar functions. RTAS provides such functions, including functions for accessing the real-time clock, NVRAM 20, restart, shutdown, and PCI configuration cycles. The I/O

> primitives for these functions in the ROM Image make use of RTAS.

Esfahani ’695, 6:29-54; *also* Esfahani ’265, 6:31-57.

> The Boot Info file 40 will now be described in greater detail. The Boot Info file 40 may be stored in the System Folder of the start-up volume. Alternatively, the Boot Info file 40 may be provided by a network server using, for example, the Bootstrap Protocol (BootP), which is described in B. Croft et al., Network Working Group Request for Comments (RFC) 951, September 1985. Referring to FIG. 5, the Boot Info file 40 includes Open Firmware-specific Mac OS code 52, referred to as the “Trampoline code”; an Open Firmware header 51, which includes a Forth script that performs operations necessary to the start-up of the OS, including validation tests and transfer of control to the Trampoline code; and a compressed ROM Image 53, which represents the mid-level portion 32 of the OS 30. The purpose of the header 51 is, generally, to specify the locations of the other components of the Boot Info file 40. The purpose of the Trampoline code 53 generally is to handle the transition between the Open Firmware code in the Boot Rom 11 and the ROM Image 52, as will be described in greater detail below.

Esfahani ’695, 7:5-24; *also* Esfahani ’265, 7:9:28.

> Note that in alternative embodiments of the OS 30, some or all of the above mentioned components of the ROM image 53 may be provided as separate, compressed elements. These separate elements may be embodied in separate Boot Info files or in a single Boot Info file. This approach would allow the OS components that are required for a given machine to be individually selected, decompressed as part of the ROM image, and used as part of the OS 30, while unnecessary components could be ignored.

Esfahani ’695, 7:43-51; *also* Esfahani ’265, 7:47-55.

> The Boot Info file 40 resides on the boot device (e.g., a disk, or on a network) and has a localizable name.

Esfahani ’695, 7:66-67; *also* Esfahani ’265, 8:3-4.

See also Esfahani, Figs. 1, 4, 5, 6A, and 6B.

1.2 initializing a central processing unit of said computer system;	Esfahani, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

> The low-level portion, including hardware-specific code, is stored in a relatively small read-only memory (ROM), while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device, which may be located remotely from the computer system. Upon power-up or reset of the computer system, the code in the ROM is executed to read the compressed ROM image into random access memory (RAM) of the computer system. The compressed image is then decompressed and executed as part of the boot sequence. Once decompressed, the portion of RAM storing the intermediate-level code is write-protected in the memory map, and the code in boot ROM is deleted from the memory map. Memory space in RAM that is allocated to the intermediate-level code but not used is returned to the operating system for use as part of system RAM.

Esfahani '695 [Abstract]

> During start-up, the Boot Info file 40 is loaded into RAM 12, and the compressed mid-level OS 32 is decompressed. Hence, the mid-level OS 32 is essentially a compressed ROM image. The mid-level OS 32 is inserted into the memory map of the computer system as if it were firmware in ROM. That is, the ROM image can be write-protected in the memory map.

Esfahani '695, 5:44-51; *also* Esfahani '265, 5:48-54.

> Referring now to FIG. 5, the low-level OS portion 31 and the Boot Info file 40 are illustrated in greater detail. The low-level portion 31 stored in Boot ROM 11 contains the code needed to start up the computer, initialize and examine the hardware, provide a Device Tree (per Open Firmware) to describe the hardware, provide hardware access services,

and transfer control to the OS. In one embodiment, the components of the low-level portion 31 include:

> code for performing Power-On Self Test (POST), including code for performing diagnostics, generating a boot beep and an error beep;
>
> Open Firmware code;
>
> hardware-specific Mac OS drivers ("ndrv's") that are needed at boot time (drivers needed at boot time, e.g.,video drivers, network drivers, or disk drivers, are loaded from the Device Tree);
>
> HardwareInit code (i.e., the lowest-level code for initializing the CPU, RAM, clock, system bus, etc.) without Mac OS-specific code;
>
> code for performing Run-Time Abstraction Services (RTAS). Certain hardware devices differ from machine to machine, but provide similar functions. RTAS provides such functions, including functions for accessing the real-time clock, NVRAM 20, restart, shutdown, and PCI configuration cycles. The I/O primitives for these functions in the ROM Image make use of RTAS.

Esfahani '695, 6:29-54; *also* Esfahani '265, 6:31-57.

> The Boot Info file 40 will now be described in greater detail. The Boot Info file 40 may be stored in the System Folder of the start-up volume. Alternatively, the Boot Info file 40 may be provided by a network server using, for example, the Bootstrap Protocol (BootP), which is described in B. Croft et al., Network Working Group Request for Comments (RFC) 951, September 1985. Referring to FIG. 5, the Boot Info file 40 includes Open Firmware-specific Mac OS code 52, referred to as the "Trampoline code"; an Open Firmware header 51, which includes a Forth script that performs operations necessary to the start-up of the OS, including validation tests and transfer of control to the Trampoline code; and a compressed ROM Image 53, which represents the mid-level portion 32 of the OS 30. The purpose of the header 51 is, generally, to specify the locations of the other components of the Boot Info file 40. The purpose of the Trampoline code 53 generally is to handle the transition between the Open Firmware code in the Boot Rom 11 and the ROM Image 52, as will be described in greater detail below.

Esfahani ’695, 7:5-24; *also* Esfahani ’265, 7:9:28.

> Overall Boot Process
>
> Referring now to FIGS. 5, 6A and 6B, the overall boot process of the improved OS 30 will now be described. In response to power on or reset at block 601 (FIG. 6A), the POST code executes (preliminary diagnostics, boot beep, initialization) at 602. At 603, the Open Firmware routine initializes and begins execution, including building the expanded Device Tree and the Interrupt Trees. At 604, the Open Firmware locates the Boot Info file 40, based on defaults and NVRAM settings, and loads the Boot Info file 40 into RAM 12. At 605, the Open Firmware executes the Forth script in the Boot Info file 40, which contains information about the rest of the file (offsets, etc.) and instructions to read both the Trampoline code 52 and the compressed ROM Image 53, and places them into a temporary place in RAM 12. At 606, the Open Firmware transfers control to the Trampoline code 52. At 607, the Trampoline code 52 decompresses the ROM Image 53 in RAM 12. In addition, the Trampoline code reallocates any unused memory space in the ROM Image 52; gathers information about the system from Open Firmware; creates data structures based on this information; terminates Open Firmware, and rearranges the contents of memory to an interim location in physical memory space. At 608, the Trampoline code 52 transfers control to the HardwareInit routine of the (now decompressed) ROM Image 53. At 609, the HardwareInit routine copies data structures to their correct places in memory and then calls the kernel of the OS. Next, at 610 the kernel fills in its data structures and then calls the 68K Emulator. At 611, the 68K Emulator initializes itself and then transfers control to the StartInit routine. At 612, the StartInit routine begins execution, initializing data structures and managers, and booting the MacOS (the high-level portion of the OS). Note that blocks 609 through 612, above, are specific to a Mac OS.

Esfahani ’695, 9:38-10:6; *also* Esfahani ’265, 8:42-9:10.

See also Esfahani, Figs. 1, 4, 5, 6A, and 6B.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Esfahani, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

> The low-level portion, including hardware-specific code, is stored in a relatively small read-only memory (ROM), while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device, which may be located remotely from the computer system. Upon power-up or reset of the computer system, the code in the ROM is executed to read the compressed ROM image into random access memory (RAM) of the computer system. The compressed image is then decompressed and executed as part of the boot sequence. Once decompressed, the portion of RAM storing the intermediate-level code is write-protected in the memory map, and the code in boot ROM is deleted from the memory map. Memory space in RAM that is allocated to the intermediate-level code but not used is returned to the operating system for use as part of system RAM.

Esfahani '695 [Abstract]

> A method and apparatus for use in booting a computer system are provided. The method includes loading a compressed image of a first portion of the OS of the computer system into a storage device of the computer system, which may be RAM, for example. The compressed image of the first portion of the OS is then decompressed and executed as part of the boot sequence of the computer system.

Esfahani '695 2:6-12; *also* Esfahani '265, 2:10-17.

> In particular embodiments, the first portion of the operating system may include an intermediate-level portion of the operating system, while a second, low-level portion of the operating system containing hardware specific aspects are stored in read-only memory. The process of

> transferring the first portion from non-volatile storage to volatile storage and decompressing the first portion may be initiated by the low-level portion stored in the read-only memory. Once decompressed, the portion of memory storing the first portion of the operating system may be write-protected, and this read-only memory portion may be mapped out of the address space of RAM used by the operating system.

Esfahani 695 2:13-25; *also* Esfahani '265, 2:18-29.

> A computer OS using a compressed ROM image in RAM is described. In brief, the low-level portion of an OS of a computer is designed to be separate from the intermediate-level portion of the OS. The low-level portion, which includes hardware-specific code, is stored in a relatively small Boot ROM, while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device. The mass storage device may be located remotely from the computer system, such as in a file server. Upon power-up or reset of the computer system, the code in the boot ROM is executed to read the compressed ROM image into RAM, i.e., system memory, of the computer system. The compressed image is then decompressed and executed as part of the boot sequence.

Esfahani 695, 2:54-67; *also* Esfahani '265, 2:58-3:4.

> Referring now to FIG. 4, in the improved OS, the low-level (hardware-specific) OS code 31 resides in firmware, in order to handle start-up activities of the computer system. This code fits into one, relatively small ROM, referred to as the Boot ROM 11. Thus, Boot ROM 11 includes all of the hardware-specific code and tables needed to start up the computer as well as to boot the OS and provide common hardware access services the OS might require. Note that the Boot ROM code is not specific to the MacOS or to any other OS. All higher-level software resides elsewhere, as will now be described.
>
> Prior to start-up, the mid-level portion 32 of OS 30 (which corresponds to part of the ToolBox ROM of earlier Macintosh computers) resides in compressed form in a file 40, referred to as Boot Info file 40.

Esfahani '695, 5:7-23; *also* Esfahani '265, 5:10-24.

> During start-up, the Boot Info file 40 is loaded into RAM 12, and the compressed mid-level OS 32 is decompressed. Hence, the mid-level OS 32 is essentially a compressed ROM image. The mid-level OS 32 is inserted into the memory map of the computer system as if it were

> firmware in ROM. That is, the ROM image can be write-protected in the memory map.

Esfahani ’695, 5:44-51; *also* Esfahani ’265, 5:48-54.

> Referring now to FIG. 5, the low-level OS portion 31 and the Boot Info file 40 are illustrated in greater detail. The low-level portion 31 stored in Boot ROM 11 contains the code needed to start up the computer, initialize and examine the hardware, provide a Device Tree (per Open Firmware) to describe the hardware, provide hardware access services, and transfer control to the OS. In one embodiment, the components of the low-level portion 31 include:
>
> > code for performing Power-On Self Test (POST), including code for performing diagnostics, generating a boot beep and an error beep;
> >
> > Open Firmware code;
> >
> > hardware-specific Mac OS drivers (“ndrv's”) that are needed at boot time (drivers needed at boot time, e.g.,video drivers, network drivers, or disk drivers, are loaded from the Device Tree);
> >
> > HardwareInit code (i.e., the lowest-level code for initializing the CPU, RAM, clock, system bus, etc.) without Mac OS-specific code;
> >
> > code for performing Run-Time Abstraction Services (RTAS). Certain hardware devices differ from machine to machine, but provide similar functions. RTAS provides such functions, including functions for accessing the real-time clock, NVRAM 20, restart, shutdown, and PCI configuration cycles. The I/O primitives for these functions in the ROM Image make use of RTAS.

Esfahani ’695, 6:29-54; *also* Esfahani ’265, 6:31-57.

> Note that in alternative embodiments of the OS 30, some or all of the above mentioned components of the ROM image 53 may be provided as separate, compressed elements. These separate elements may be embodied in separate Boot Info files or in a single Boot Info file. This approach would allow the OS components that are required for a given machine to be individually selected, decompressed as part of the ROM

image, and used as part of the OS 30, while unnecessary components could be ignored.

Esfahani ’695, 7:43-51; *also* Esfahani ’265, 7:47-55.

See also Esfahani, Figs. 1, 4, 5, 6A, and 6B.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Esfahani, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

> The low-level portion, including hardware-specific code, is stored in a relatively small read-only memory (ROM), while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device, which may be located remotely from the computer system. Upon power-up or reset of the computer system, the code in the ROM is executed to read the compressed ROM image into random access memory (RAM) of the computer system. The compressed image is then decompressed and executed as part of the boot sequence. Once decompressed, the portion of RAM storing the intermediate-level code is write-protected in the memory map, and the code in boot ROM is deleted from the memory map. Memory space in RAM that is allocated to the intermediate-level code but not used is returned to the operating system for use as part of system RAM.

Esfahani '695 [Abstract]

> A method and apparatus for use in booting a computer system are provided. The method includes loading a compressed image of a first portion of the OS of the computer system into a storage device of the computer system, which may be RAM, for example. The compressed image of the first portion of the OS is then decompressed and executed as part of the boot sequence of the computer system.

Esfahani '695 2:6-12; *also* Esfahani '265, 2:10-17.

> In particular embodiments, the first portion of the operating system may include an intermediate-level portion of the operating system, while a second, low-level portion of the operating system containing hardware specific aspects are stored in read-only memory. The process of transferring the first portion from non-volatile storage to volatile storage

> and decompressing the first portion may be initiated by the low-level portion stored in the read-only memory. Once decompressed, the portion of memory storing the first portion of the operating system may be write-protected, and this read-only memory portion may be mapped out of the address space of RAM used by the operating system.

Esfahani 695 2:13-25; *also* Esfahani ’265, 2:18-29.

> A computer OS using a compressed ROM image in RAM is described. In brief, the low-level portion of an OS of a computer is designed to be separate from the intermediate-level portion of the OS. The low-level portion, which includes hardware-specific code, is stored in a relatively small Boot ROM, while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device. The mass storage device may be located remotely from the computer system, such as in a file server. Upon power-up or reset of the computer system, the code in the boot ROM is executed to read the compressed ROM image into RAM, i.e., system memory, of the computer system. The compressed image is then decompressed and executed as part of the boot sequence.

Esfahani 695, 2:54-67; *also* Esfahani ’265, 2:58-3:4.

> Referring now to FIG. 4, in the improved OS, the low-level (hardware-specific) OS code 31 resides in firmware, in order to handle start-up activities of the computer system. This code fits into one, relatively small ROM, referred to as the Boot ROM 11. Thus, Boot ROM 11 includes all of the hardware-specific code and tables needed to start up the computer as well as to boot the OS and provide common hardware access services the OS might require. Note that the Boot ROM code is not specific to the MacOS or to any other OS. All higher-level software resides elsewhere, as will now be described.
>
> Prior to start-up, the mid-level portion 32 of OS 30 (which corresponds to part of the ToolBox ROM of earlier Macintosh computers) resides in compressed form in a file 40, referred to as Boot Info file 40.

Esfahani ’695, 5:7-23; *also* Esfahani ’265, 5:10-24.

> During start-up, the Boot Info file 40 is loaded into RAM 12, and the compressed mid-level OS 32 is decompressed. Hence, the mid-level OS 32 is essentially a compressed ROM image. The mid-level OS 32 is inserted into the memory map of the computer system as if it were firmware in ROM. That is, the ROM image can be write-protected in the memory map.

Esfahani ’695, 5:44-51; *also* Esfahani ’265, 5:48-54.

> Referring now to FIG. 5, the low-level OS portion 31 and the Boot Info file 40 are illustrated in greater detail. The low-level portion 31 stored in Boot ROM 11 contains the code needed to start up the computer, initialize and examine the hardware, provide a Device Tree (per Open Firmware) to describe the hardware, provide hardware access services, and transfer control to the OS. In one embodiment, the components of the low-level portion 31 include:
>
> > code for performing Power-On Self Test (POST), including code for performing diagnostics, generating a boot beep and an error beep;
> >
> > Open Firmware code;
> >
> > hardware-specific Mac OS drivers (“ndrv's”) that are needed at boot time (drivers needed at boot time, e.g.,video drivers, network drivers, or disk drivers, are loaded from the Device Tree);
> >
> > HardwareInit code (i.e., the lowest-level code for initializing the CPU, RAM, clock, system bus, etc.) without Mac OS-specific code;
> >
> > code for performing Run-Time Abstraction Services (RTAS). Certain hardware devices differ from machine to machine, but provide similar functions. RTAS provides such functions, including functions for accessing the real-time clock, NVRAM 20, restart, shutdown, and PCI configuration cycles. The I/O primitives for these functions in the ROM Image make use of RTAS.

Esfahani ’695, 6:29-54; *also* Esfahani ’265, 6:31-57.

> Note that in alternative embodiments of the OS 30, some or all of the above mentioned components of the ROM image 53 may be provided as separate, compressed elements. These separate elements may be embodied in separate Boot Info files or in a single Boot Info file. This approach would allow the OS components that are required for a given machine to be individually selected, decompressed as part of the ROM image, and used as part of the OS 30, while unnecessary components could be ignored.

Appendix B1
Invalidity of U.S. Patent 8,090,936 based on Esfahani

Esfahani ’695, 7:43-51; *also* Esfahani ’265, 7:47-55. *See also* Esfahani, Figs. 1, 4, 5, 6A, and 6B.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Esfahani, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

> The low-level portion, including hardware-specific code, is stored in a relatively small read-only memory (ROM), while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device, which may be located remotely from the computer system. Upon power-up or reset of the computer system, the code in the ROM is executed to read the compressed ROM image into random access memory (RAM) of the computer system. The compressed image is then decompressed and executed as part of the boot sequence. Once decompressed, the portion of RAM storing the intermediate-level code is write-protected in the memory map, and the code in boot ROM is deleted from the memory map. Memory space in RAM that is allocated to the intermediate-level code but not used is returned to the operating system for use as part of system RAM.

Esfahani '695 [Abstract]

> A method and apparatus for use in booting a computer system are provided. The method includes loading a compressed image of a first portion of the OS of the computer system into a storage device of the computer system, which may be RAM, for example. The compressed image of the first portion of the OS is then decompressed and executed as part of the boot sequence of the computer system.

Esfahani '695 2:6-12; *also* Esfahani '265, 2:10-17.

> In particular embodiments, the first portion of the operating system may include an intermediate-level portion of the operating system, while a

> second, low-level portion of the operating system containing hardware specific aspects are stored in read-only memory. The process of transferring the first portion from non-volatile storage to volatile storage and decompressing the first portion may be initiated by the low-level portion stored in the read-only memory. Once decompressed, the portion of memory storing the first portion of the operating system may be write-protected, and this read-only memory portion may be mapped out of the address space of RAM used by the operating system.

Esfahani 695 2:13-25; *also* Esfahani '265, 2:18-29.

> A computer OS using a compressed ROM image in RAM is described. In brief, the low-level portion of an OS of a computer is designed to be separate from the intermediate-level portion of the OS. The low-level portion, which includes hardware-specific code, is stored in a relatively small Boot ROM, while at least part of the intermediate-level portion is stored as a compressed ROM image on a disk or other mass storage device. The mass storage device may be located remotely from the computer system, such as in a file server. Upon power-up or reset of the computer system, the code in the boot ROM is executed to read the compressed ROM image into RAM, i.e., system memory, of the computer system. The compressed image is then decompressed and executed as part of the boot sequence.

Esfahani 695, 2:54-67; *also* Esfahani '265, 2:58-3:4.

> Note that in alternative embodiments of the OS 30, some or all of the above mentioned components of the ROM image 53 may be provided as separate, compressed elements. These separate elements may be embodied in separate Boot Info files or in a single Boot Info file. This approach would allow the OS components that are required for a given machine to be individually selected, decompressed as part of the ROM image, and used as part of the OS 30, while unnecessary components could be ignored.

Esfahani '695, 7:43-51; *also* Esfahani '265, 7:47-55.

> Overall Boot Process
>
> Referring now to FIGS. 5, 6A and 6B, the overall boot process of the improved OS 30 will now be described. In response to power on or reset at block 601 (FIG. 6A), the POST code executes (preliminary diagnostics, boot beep, initialization) at 602. At 603, the Open Firmware routine initializes and begins execution, including building the expanded Device Tree and the Interrupt Trees. At 604, the Open

> Firmware locates the Boot Info file 40, based on defaults and NVRAM settings, and loads the Boot Info file 40 into RAM 12. At 605, the Open Firmware executes the Forth script in the Boot Info file 40, which contains information about the rest of the file (offsets, etc.) and instructions to read both the Trampoline code 52 and the compressed ROM Image 53, and places them into a temporary place in RAM 12. At 606, the Open Firmware transfers control to the Trampoline code 52. At 607, the Trampoline code 52 decompresses the ROM Image 53 in RAM 12. In addition, the Trampoline code reallocates any unused memory space in the ROM Image 52; gathers information about the system from Open Firmware; creates data structures based on this information; terminates Open Firmware, and rearranges the contents of memory to an interim location in physical memory space. At 608, the Trampoline code 52 transfers control to the HardwareInit routine of the (now decompressed) ROM Image 53. At 609, the HardwareInit routine copies data structures to their correct places in memory and then calls the kernel of the OS. Next, at 610 the kernel fills in its data structures and then calls the 68K Emulator. At 611, the 68K Emulator initializes itself and then transfers control to the StartInit routine. At 612, the StartInit routine begins execution, initializing data structures and managers, and booting the MacOS (the high-level portion of the OS). Note that blocks 609 through 612, above, are specific to a Mac OS.

Esfahani ’695, 9:38-10:6; *also* Esfahani ’265, 8:42-9:10.

See also Esfahani, Figs. 1, 4, 5, 6A, and 6B.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Esfahani, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Esfahani, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Refer now to FIG. 2, which illustrates the structure of a traditional Macintosh OS. The OS 22 provides an interface between the hardware 23 of the computer system and the software applications 21. The OS includes the so-called MacOS 22A, which is the high-level portion of the OS 22. In this context, "high-level" refers to the portion of the OS 22 which is least hardware-specific and has the greatest degree of abstraction. Traditionally, this file would reside on disk or other mass storage device and would typically be the last portion of the OS22 to be invoked during the boot process. The (traditional Macintosh) OS22 also includes the so-called ToolBox ROM code 22B. The ToolBox ROM code 22B is firmware that resides in a ROM. The ToolBox ROM code 22B represents the middle- or intermediate-level and low-level portions of the OS22. In this context, "low-level" refers to the portion of the OS which is most hardware-specific and has the smallest degree of abstraction. The middle-level portion has degrees of hardware-dependence and abstraction lower than those of the high-level OS22A and greater than those of the low-level OS.

Esfahani, '695, 4:1-20; Esfahani '265, 4:5-25

See also Esfahani Fig. 2

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Esfahani, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Refer now to FIG. 2, which illustrates the structure of a traditional Macintosh OS. The OS 22 provides an interface between the hardware 23 of the computer system and the software applications 21. The OS includes the so-called MacOS 22A, which is the high-level portion of the OS 22. In this context, "high-level" refers to the portion of the OS 22 which is least hardware-specific and has the greatest degree of abstraction. Traditionally, this file would reside on disk or other mass storage device and would typically be the last portion of the OS22 to be invoked during the boot process. The (traditional Macintosh) OS22 also includes the so-called ToolBox ROM code 22B. The ToolBox ROM code 22B is firmware that resides in a ROM. The ToolBox ROM code 22B represents the middle- or intermediate-level and low-level portions of the OS22. In this context, "low-level" refers to the portion of the OS which is most hardware-specific and has the smallest degree of abstraction. The middle-level portion has degrees of hardware-dependence and abstraction lower than those of the high-level OS22A and greater than those of the low-level OS.

Esfahani, '695, 4:1-20; Esfahani '265, 4:5-25

See also Esfahani Fig. 2

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Esfahani, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> FIG. 1 illustrates a computer system 1 in which the present invention may be implemented. Note that while FIG. 1 illustrates the major components of a computer system, it is not intended to represent any particular architecture or manner of interconnecting the component; such details are not germane to the present invention. The computer system of FIG. 1 may be, for example, an Apple Macintosh computer, such as an Apple iMac computer. As shown, the computer system 1 of FIG. 1 includes a microprocessor 10, a read-only memory (ROM) 11, random access memory (RAM) 12, each connected to a bus system 18. The bus system 18 may include one or more buses connected to each other through various bridges, controllers and/or adapters, such as are well-known in the art. For example, the bus system may include a "system bus" that is connected through an adapter to one or more expansion buses, such as a Peripheral Component Interconnect (PCI) bus, or the like. Also coupled to the bus system 18 are a mass storage device 13, a display device 14, a keyboard 15, a pointing device 16, a communication device 17, and non-volatile RAM (NVRAM) 20. A cache memory 19 is coupled to the microprocessor 10.

Esfahani, '695, 3:1-22; Esfahani '265, 3:4-26

> Operation of the Trampoline code is described now in further detail with reference to FIGS. 7A through 7D. At 701, the debugging level for subsequent messages, if any, is determined. More specifically, a node in the Device Tree is checked for a specific property, and if the property exists, then its value is used to determine the debugging level. At 702, it

> is determined whether to create the read/write Debug ROM Image aperture (a specific level of debugging will cause the Debug ROM Image aperture to be created), and at 703 the presence of the appropriate copyright notice in the Boot ROM 11 is verified. At 704, the memory controller and the main memory node are located. At 705, the Trampoline code allocates RAM for the RTAS and instantiates the RTAS. At 706, an appropriate amount of RAM (e.g., 768K in one embodiment) is allocated for a Work Area that will contain the Device Tree and various other data structures. At this point, in certain embodiments, information on the hardware configuration of the system (e.g., processor, memory devices, etc.) may be gathered from the Device Tree and saved in one or more records. At 707, space is allocated in RAM for the ROM Image 53, and the ROM Image 53 is then decompressed from the Boot Info file into the allocated space.

Esfahani, ’695, 9:53-10:8; Esfahani ’265, 9:57-10:11

> At 712, the interrupt controller(s) are initialized, and at 713 the memory map is built. At 714, the ROM Image checksum is verified, and at 715, the physical addresses of the data structures in the Work Area are determined in preparation for moving these data structured to their proper locations. At 716, the Trampoline code causes Open Firmware to quiesce, while allowing Open Firmware to shut down any active hardware that might be performing DMA or interrupts. At 717, the Trampoline code switches from virtual to real mode, moves the ROM Image to its permanent location in RAM, and moves the contents of the Work Area to interim locations. At 718, control is transferred to the HardwareInit routine of the ROM Image. At 719, the Work Area data structures are moved to the their permanent locations, and at 720, control is transferred from the HardwareInit code to the kernel.

Esfahani, ’695, 10:21-35; Esfahani ’265, 10:25-40

See also Esfahani Fig. 1

6. The method of claim 1, further comprising updating the list of boot data.	Esfahani, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Referring now to FIG. 4, in the improved OS, the low-level (hardware-specific) OS code 31 resides in firmware, in order to handle start-up activities of the computer system. This code fits into one, relatively small ROM, referred to as the Boot ROM 11. Thus, Boot ROM 11 includes all of the hardware-specific code and tables needed to start up the computer as well as to boot the OS and provide common hardware access services the OS might require. Note that the Boot ROM code is not specific to the MacOS or to any other OS. All higher-level software resides elsewhere, as will now be described.
>
> Prior to start-up, the mid-level portion 32 of OS 30 (which corresponds to part of the ToolBox ROM of earlier Macintosh computers) resides in compressed form in a file 40, referred to as Boot Info file 40.

Esfahani '695, 5:7-23; *also* Esfahani '265, 5:10-24.

> During start-up, the Boot Info file 40 is loaded into RAM 12, and the compressed mid-level OS 32 is decompressed. Hence, the mid-level OS 32 is essentially a compressed ROM image. The mid-level OS 32 is inserted into the memory map of the computer system as if it were firmware in ROM. That is, the ROM image can be write-protected in the memory map.

Esfahani '695, 5:44-51; *also* Esfahani '265, 5:48-54.

> Referring now to FIG. 5, the low-level OS portion 31 and the Boot Info file 40 are illustrated in greater detail. The low-level portion 31 stored in Boot ROM 11 contains the code needed to start up the computer, initialize and examine the hardware, provide a Device Tree (per Open

> Firmware) to describe the hardware, provide hardware access services, and transfer control to the OS. In one embodiment, the components of the low-level portion 31 include:
>
> > code for performing Power-On Self Test (POST), including code for performing diagnostics, generating a boot beep and an error beep;
> >
> > Open Firmware code;
> >
> > hardware-specific Mac OS drivers ("ndrv's") that are needed at boot time (drivers needed at boot time, e.g.,video drivers, network drivers, or disk drivers, are loaded from the Device Tree);
> >
> > HardwareInit code (i.e., the lowest-level code for initializing the CPU, RAM, clock, system bus, etc.) without Mac OS-specific code;
> >
> > code for performing Run-Time Abstraction Services (RTAS). Certain hardware devices differ from machine to machine, but provide similar functions. RTAS provides such functions, including functions for accessing the real-time clock, NVRAM 20, restart, shutdown, and PCI configuration cycles. The I/O primitives for these functions in the ROM Image make use of RTAS.

Esfahani '695, 6:29-54; *also* Esfahani '265, 6:31-57.

> The Boot Info file 40 will now be described in greater detail. The Boot Info file 40 may be stored in the System Folder of the start-up volume. Alternatively, the Boot Info file 40 may be provided by a network server using, for example, the Bootstrap Protocol (BootP), which is described in B. Croft et al., Network Working Group Request for Comments (RFC) 951, September 1985. Referring to FIG. 5, the Boot Info file 40 includes Open Firmware-specific Mac OS code 52, referred to as the "Trampoline code"; an Open Firmware header 51, which includes a Forth script that performs operations necessary to the start-up of the OS, including validation tests and transfer of control to the Trampoline code; and a compressed ROM Image 53, which represents the mid-level portion 32 of the OS 30. The purpose of the header 51 is, generally, to specify the locations of the other components of the Boot Info file 40. The purpose of the Trampoline code 53 generally is to handle the transition between the Open Firmware code in the Boot Rom 11 and the ROM Image 52, as will be described in greater detail below.

Esfahani ’695, 7:5-24; *also* Esfahani ’265, 7:9:28.

> Note that in alternative embodiments of the OS 30, some or all of the above mentioned components of the ROM image 53 may be provided as separate, compressed elements. These separate elements may be embodied in separate Boot Info files or in a single Boot Info file. This approach would allow the OS components that are required for a given machine to be individually selected, decompressed as part of the ROM image, and used as part of the OS 30, while unnecessary components could be ignored.

Esfahani ’695, 7:43-51; *also* Esfahani ’265, 7:47-55.

> The Boot Info file 40 resides on the boot device (e.g., a disk, or on a network) and has a localizable name.

Esfahani ’695, 7:66-67; *also* Esfahani ’265, 8:3-4.

See also Esfahani, Figs. 1, 4, 5, 6A, and 6B.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Esfahani, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The ROM Image 53 of the improved OS 30 is similar to the ToolBox ROM code of the earlier Macintosh OS, in that it has a similar layout and contains many of the same components. The image may be compressed using a known compression algorithm, such as LZSS, for example. The ROM Image 53 may also be encoded, if desired.

Esfahani, '695, 8:32-37; Esfahani '265, 8:36-41

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Esfahani, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The ROM Image 53 of the improved OS 30 is similar to the ToolBox ROM code of the earlier Macintosh OS, in that it has a similar layout and contains many of the same components. The image may be compressed using a known compression algorithm, such as LZSS, for example. The ROM Image 53 may also be encoded, if desired.

Esfahani, '695, 8:32-37; Esfahani '265, 8:36-41

11.1. a processor;	Esfahani, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Esfahani, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Esfahani, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Esfahani discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> In particular embodiments, the first portion of the operating system may include an intermediate-level portion of the operating system, while a second, low-level portion of the operating system containing hardware specific aspects are stored in read-only memory. The process of transferring the first portion from non-volatile storage to volatile storage and decompressing the first portion may be initiated by the low-level portion stored in the read-only memory. Once decompressed, the portion of memory storing the first portion of the operating system may be write-protected, and this read-only memory portion may be mapped out of the address space of RAM used by the operating system.

Esfahani, '695, 2:14-25; Esfahani '265, 2:17-29

> FIG. 1 illustrates a computer system 1 in which the present invention may be implemented. Note that while FIG. 1 illustrates the major components of a computer system, it is not intended to represent any particular architecture or manner of interconnecting the component; such details are not germane to the present invention. The computer

> system of FIG. 1 may be, for example, an Apple Macintosh computer, such as an Apple iMac computer. As shown, the computer system 1 of FIG. 1 includes a microprocessor 10, a read-only memory (ROM) 11, random access memory (RAM) 12, each connected to a bus system 18. The bus system 18 may include one or more buses connected to each other through various bridges, controllers and/or adapters, such as are well-known in the art. For example, the bus system may include a "system bus" that is connected through an adapter to one or more expansion buses, such as a Peripheral Component Interconnect (PCI) bus, or the like. Also coupled to the bus system 18 are a mass storage device 13, a display device 14, a keyboard 15, a pointing device 16, a communication device 17, and non-volatile RAM (NVRAM) 20. A cache memory 19 is coupled to the microprocessor 10.

Esfahani, '695, 3:1-22; Esfahani '265, 3:4-26

> The Boot Info file 40 resides on the boot device (e.g., a disk, or on a network) and has a localizable name. Identification information that leads to the file's path may be stored in NVRAM 20 or another suitable location, and the search algorithm for a usable Boot Info file parallels the search mechanism across SCSI, ATA, etc., used in the earlier Macintosh OS's StartSearch routine. By default, the Boot Info file 40 is located by using the current, active System folder's dirID (directory ID) in the boot block of each partition of the Hierarchical File System (HFS) and then searching for a file with a predetermined file type. Searching by file type is done to allow localization of the file.

Esfahani, '695, 7:66-8:10; Esfahani '265, 8:3-14

See also Esfahani Fig. 1

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Esfahani, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Esfahani, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Esfahani, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Esfahani, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Esfahani, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Esfahani, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Esfahani discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B2
Invalidity of U.S. Patent 8,090,936 based on Lillich

U.S. Patent Nos. 5,619,698 to Lillich ("Lillich '698) and 5,790,856 to Lillich ("Lillich '856"), alone or in combination, invalidate claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

In addition, the prior art 68k operating system disclosed in Lillich (see Lillich '856, 1:20-5:55, and Lillich '698, 1:10-5:30) ("68k System") invalidates claims 1-6, 8-9, 11-13, and 15-16 of the '936 Patent pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 because the system was on sale or in public use more than one year prior to the filing date of the '936 patent. Lillich '698, Lillich '856, and the 68k System are collectively referred to herein as "Lillich."

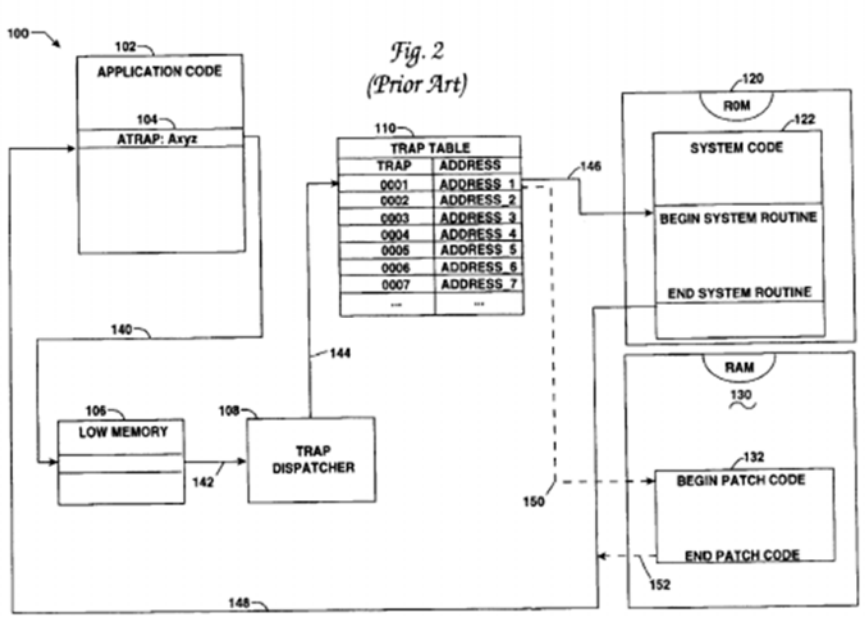
Additionally, Apple intends to provide and seek discovery related to this system to obtain copies of relevant materials, including but not limited to, architectural documents, design documents, implementation documents, internal publications, patent applications and business records relating to this product and will supplement these contentions after those materials are discovered or received. Apple also reserves its rights to rely on any additional 68k System materials, either individually or collectively, as prior art publications to the '936 patent.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Lillich, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this claim limitation:



Lillich, Fig. 2

> "With reference to FIG. 2. the well known 68K patching paradigm 100 will be described. By way of background. The 68K patching paradigm 100 is implemented by versions of the Macintosh® Operating system

> designed for the Motorola 68K series of microprocessors. Which operating system is hereinafter referred to as the "68K operating system." The paradigm 100 is implemented on a computer system such as computer system 50 of FIG. 1. wherein a CPU 52 includes one of the Motorola 68K series microprocessors or microcontrollers as well as other components necessary to properly interface with and control other devices coupled with the CPU 52."

Lillich '856, 1:40-51; *also* Lillich '698, 2:1-12.

> "The system routines for the 68K operating system reside mainly in ROM. However. to provide flexibility for any subsequent development. application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM.

Lillich '856, 1:52-58, *also* Lillich '698, 2:14:20.

> "While the operating system routines reside mainly in ROM 120 (in their original state) information regarding the locations of the operating system routines is encoded in compressed form within ROM 120. Upon system start up. this information is decompressed and the trap table 110 is formed in RAM 130."

Lillich '856, 1:66-2:4, *also* Lillich, 2:28-33.

1.2 initializing a central processing unit of said computer system;	Lillich, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this claim limitation:

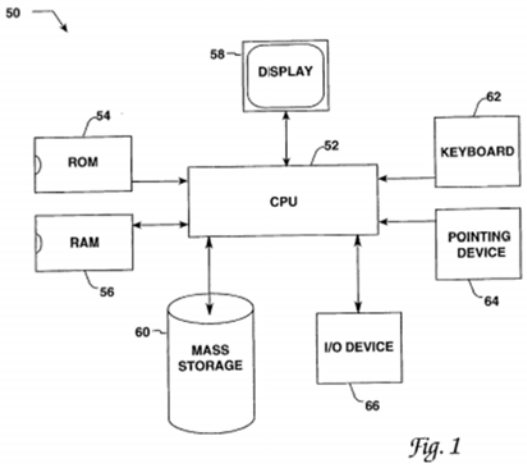


Fig. 1

Lillich, Fig. 1.

> "By way of example, a representative computer system 50 is illustrated schematically in FIG. 1."

Lillich '856, 8:44-46; *also* Lillich '698, 9:39-41.

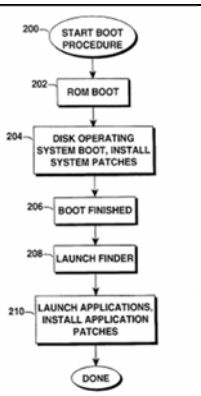


Fig. 3
(Prior Art)

Lillich ’698, Fig 3.

> “Next, in reference to FIG. 3, one method for installing patches in the 68K operating system will be described. In an initial step 200, the computer boot process is started. Then, in a step 202, the ROM boot is performed. In ROM boot step 202, initialization procedures such as expanding the trap table from ROM into RAM are performed. Next, in a step 204, the disk operating system boot is performed. At this point any system patches are also installed.”

Lillich ’698, 3:52-59.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Lillich, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this claim limitation:

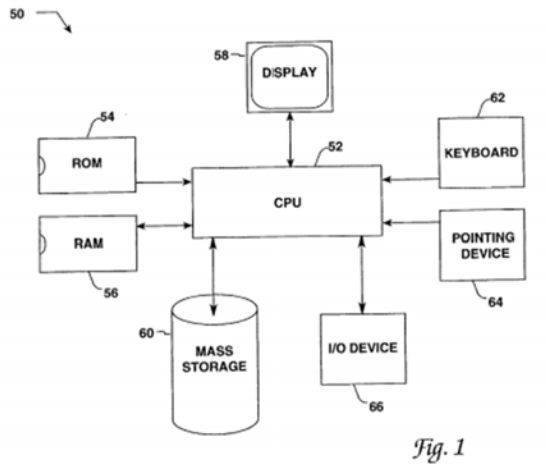


Fig. 1

Lillich, Fig. 1.

> "By way of example, a representative computer system 50 is illustrated schematically in FIG. 1."

Lillich '856, 8:44-46; *also* Lillich '698, 9:39-41.

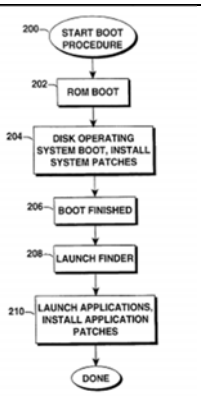


Fig. 3
(Prior Art)

Lillich ’698, Fig 3.

> “Next, in reference to FIG. 3, one method for installing patches in the 68K operating system will be described. In an initial step 200, the computer boot process is started. Then, in a step 202, the ROM boot is performed. In ROM boot step 202, initialization procedures such as expanding the trap table from ROM into RAM are performed. Next, in a step 204, the disk operating system boot is performed. At this point any system patches are also installed.”

Lillich ’698, 3:52-59.

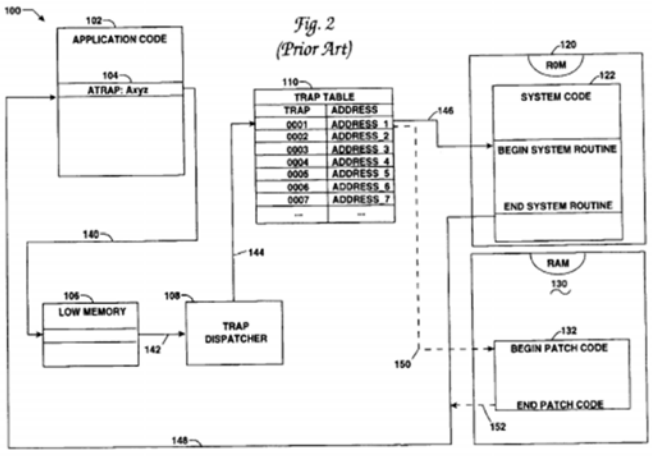


Fig. 2
(Prior Art)

Lillich, Fig. 2

Lillich — Claim 1.3

“preloading said at least a portion of said boot data in compressed form from said boot device to a memory;

> "With reference to FIG. 2. the well known 68K patching paradigm 100 will be described. By way of background. The 68K patching paradigm 100 is implemented by versions of the Macintosh® Operating system designed for the Motorola 68K series of microprocessors. Which operating system is hereinafter referred to as the "68K operating system." The paradigm 100 is implemented on a computer system such as computer system 50 of FIG. 1. wherein a CPU 52 includes one of the Motorola 68K series microprocessors or microcontrollers as well as other components necessary to properly interface with and control other devices coupled with the CPU 52."

Lillich '856, 1:40-51; *also* Lillich '698, 2:1-12.

> "The system routines for the 68K operating system reside mainly in ROM. However. to provide flexibility for any subsequent development. application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM.

Lillich '856, 1:52-58, *also* Lillich '698, 2:14:20.

> "While the operating system routines reside mainly in ROM 120 (in their original state) information regarding the locations of the operating system routines is encoded in compressed form within ROM 120. Upon system start up. this information is decompressed and the trap table 110 is formed in RAM 130."

Lillich '856, 1:66-2:4, *also* Lillich, 2:28-33.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Lillich, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this claim limitation:

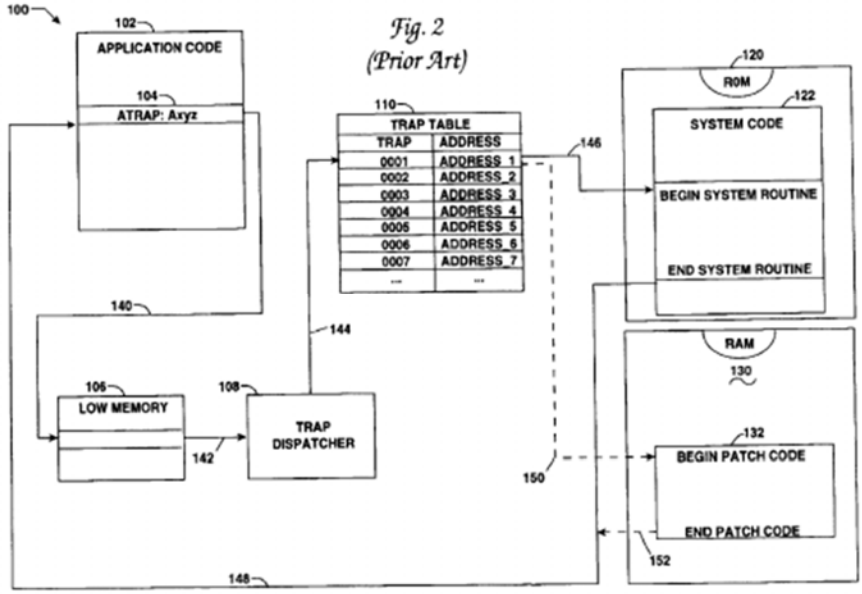


Fig. 2
(Prior Art)

Lillich, Fig. 2

> "With reference to FIG. 2. the well known 68K patching paradigm 100 will be described. By way of background. The 68K patching paradigm 100 is implemented by versions of the Macintosh® Operating system designed for the Motorola 68K series of microprocessors. Which operating system is hereinafter referred to as the "68K operating system." The paradigm 100 is implemented on a computer system such as computer system 50 of FIG. 1. wherein a CPU 52 includes one of the Motorola 68K series microprocessors or microcontrollers as well as

other components necessary to properly interface with and control other devices coupled with the CPU 52."

Lillich '856, 1:40-51; *also* Lillich '698, 2:1-12.

"The system routines for the 68K operating system reside mainly in ROM. However. to provide flexibility for any subsequent development. application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM.

Lillich '856, 1:52-58, *also* Lillich '698, 2:14:20.

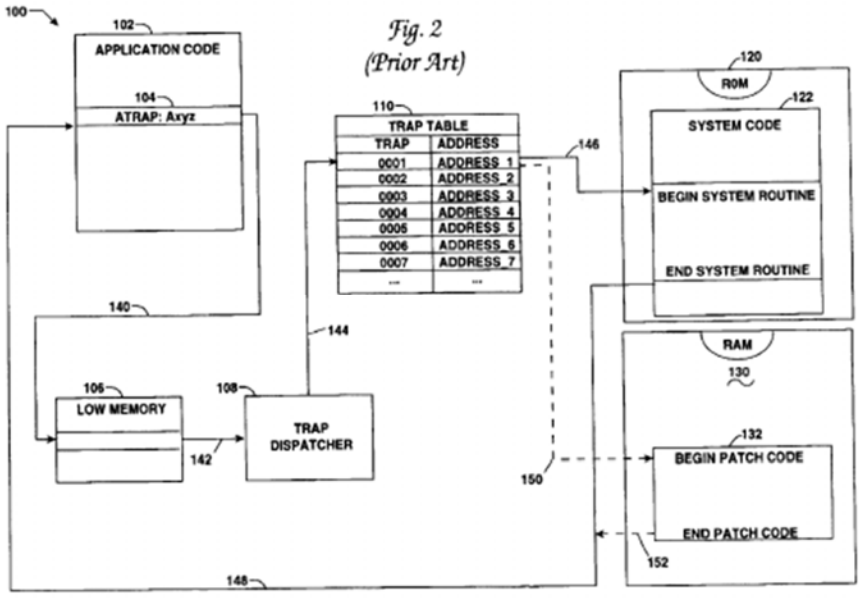
"While the operating system routines reside mainly in ROM 120 (in their original state) information regarding the locations of the operating system routines is encoded in compressed form within ROM 120. Upon system start up. this information is decompressed and the trap table 110 is formed in RAM 130."

Lillich '856, 1:66-2:4, *also* Lillich, 2:28-33.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Lillich, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this claim limitation:



Lillich, Fig. 2

> "With reference to FIG. 2. the well known 68K patching paradigm 100 will be described. By way of background. The 68K patching paradigm 100 is implemented by versions of the Macintosh® Operating system designed for the Motorola 68K series of microprocessors. Which operating system is hereinafter referred to as the "68K operating system." The paradigm 100 is implemented on a computer system such

> as computer system 50 of FIG. 1. wherein a CPU 52 includes one of the Motorola 68K series microprocessors or microcontrollers as well as other components necessary to properly interface with and control other devices coupled with the CPU 52."

Lillich '856, 1:40-51; *also* Lillich '698, 2:1-12.

> "The system routines for the 68K operating system reside mainly in ROM. However. to provide flexibility for any subsequent development. application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM.

Lillich '856, 1:52-58, *also* Lillich '698, 2:14:20.

> "While the operating system routines reside mainly in ROM 120 (in their original state) information regarding the locations of the operating system routines is encoded in compressed form within ROM 120. Upon system start up. this information is decompressed and the trap table 110 is formed in RAM 130."

Lillich '856, 1:66-2:4, *also* Lillich, 2:28-33.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Lillich, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Lillich, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The system routines for the 68K operating system reside mainly in ROM. However, to provide flexibility for any subsequent development, application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM. This indirect mechanism permits the ROM addressing of system routines to vary, or to be replaced by patch routines, without affecting the operation of applications which utilize the system routines.

Lillich '856, 1:52-61; *also* Lillich '698, 2:14-23

> Additionally, when new applications are launched they too can load new versions of individual routines into RAM and then patch the trap table in order to redirect any calls to the original system routine to the new versions.

Lillich '856, 2:60-64; *also* Lillich '698, 3:34-37

> In the world of Macintosh®, DLLs can be loosely divided into three different categories: applications, import libraries, and extensions. Typically, applications are fragments which have a user interface and are designed to function interactively with the user. Often applications do not export symbols to other fragments but rather serve as a root fragments, providing some root functionality and importing other necessary functionality.

> . . .
>
> Once an application has been launched and the binding manager has completed preparation, the end result is a new, self-sufficient executable process with an associated data closure. The closure contains a root fragment as well as instances of all the import libraries required to resolve all the import symbols present in both the root fragment and the import libraries. In explanation, the root fragment is the fragment which includes some root functionality as well as a list of import symbols.

Lillich ’856, 3:40-4:9

> As discussed previously, new versions of individual system routines may be loaded into RAM and the trap table patched in order to redirect any calls to the original system routine to the new versions. After this is complete, the operating system boot procedure is complete in a step 206. In a next step 208, the application "Finder" is launched. As will be appreciated by those skilled in the art, the Finder is the primal application which performs critical tasks such as displaying the Macintosh® desktop and launching other applications at the request of the user. Once the Finder is running, in a step 210 other applications may be launched and in turn any patches necessary for the other applications may be installed.

Lillich ’698, 3:60-4:5

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Lillich, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The system routines for the 68K operating system reside mainly in ROM. However, to provide flexibility for any subsequent development, application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM. This indirect mechanism permits the ROM addressing of system routines to vary, or to be replaced by patch routines, without affecting the operation of applications which utilize the system routines.

Lillich '856, 1:52-61; *also* Lillich '698, 2:14-23

> Additionally, when new applications are launched they too can load new versions of individual routines into RAM and then patch the trap table in order to redirect any calls to the original system routine to the new versions.

Lillich '856, 2:60-64; *also* Lillich '698, 3:34-37

> In the world of Macintosh®, DLLs can be loosely divided into three different categories: applications, import libraries, and extensions. Typically, applications are fragments which have a user interface and are designed to function interactively with the user. Often applications do not export symbols to other fragments but rather serve as a root

> fragments, providing some root functionality and importing other necessary functionality.
>
> . . .
>
> Once an application has been launched and the binding manager has completed preparation, the end result is a new, self-sufficient executable process with an associated data closure. The closure contains a root fragment as well as instances of all the import libraries required to resolve all the import symbols present in both the root fragment and the import libraries. In explanation, the root fragment is the fragment which includes some root functionality as well as a list of import symbols.

Lillich ’856, 3:40-4:9

> As discussed previously, new versions of individual system routines may be loaded into RAM and the trap table patched in order to redirect any calls to the original system routine to the new versions. After this is complete, the operating system boot procedure is complete in a step 206. In a next step 208, the application "Finder" is launched. As will be appreciated by those skilled in the art, the Finder is the primal application which performs critical tasks such as displaying the Macintosh® desktop and launching other applications at the request of the user. Once the Finder is running, in a step 210 other applications may be launched and in turn any patches necessary for the other applications may be installed.

Lillich ’698, 3:60-4:5

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Lillich, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> As will be appreciated by those skilled in the art, CPU 52 includes a microprocessor and any additional circuitry and/or device drivers necessary to control the computer system. For instance, the CPU 52 may include a keyboard controller which provides an interface between the microprocessor and the keyboard 62. ROM 64 is typically persistent memory accessible by the CPU 52 which contains the operating system instructions either in an executable format or in a compressed format which is expanded when the computer system 50 boots. RAM 56 is typically transient memory and is used as "scratch pad" memory by the operating system and/or any applications implemented on the computer system 50. For example, if a portion of the operating system present in ROM 64 is in compressed format, it may be expanded and stored into RAM 56.

Lillich '856, 8:53-67; *also* Lillich '698, 9:48-63

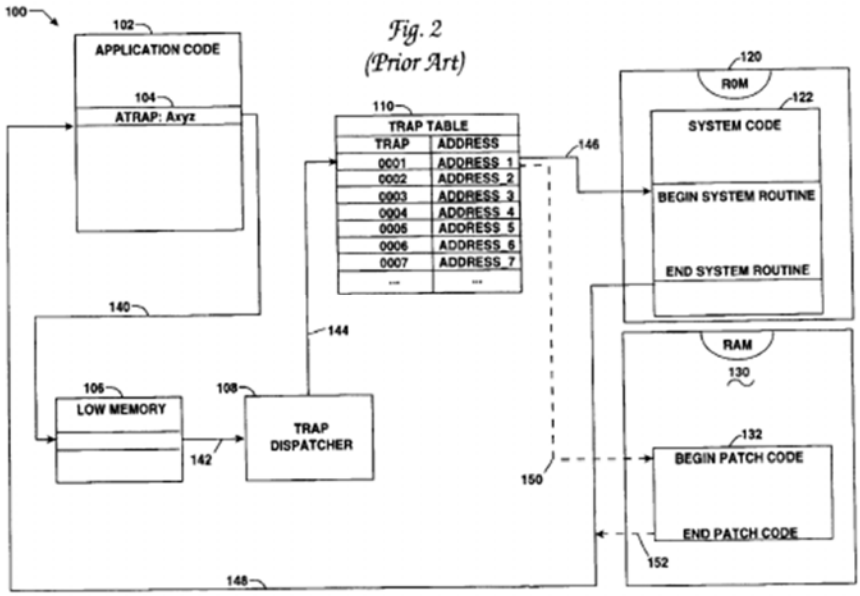
6. The method of claim 1, further comprising updating the list of boot data.	Lillich, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also



Lillich, Fig. 2

> "With reference to FIG. 2. the well known 68K patching paradigm 100 will be described. By way of background. The 68K patching paradigm 100 is implemented by versions of the Macintosh® Operating system designed for the Motorola 68K series of microprocessors. Which operating system is hereinafter referred to as the "68K operating system." The paradigm 100 is implemented on a computer system such as computer system 50 of FIG. 1. wherein a CPU 52 includes one of the Motorola 68K series microprocessors or microcontrollers as well as

other components necessary to properly interface with and control other devices coupled with the CPU 52."

Lillich '856, 1:40-51; *also* Lillich '698, 2:1-12.

"The system routines for the 68K operating system reside mainly in ROM. However. to provide flexibility for any subsequent development. application code written for execution within the 68K operating system must be kept free of any specific ROM addresses. For this reason, all calls to system routines are passed indirectly through a trap table resident in RAM.

Lillich '856, 1:52-58, *also* Lillich '698, 2:14:20.

"While the operating system routines reside mainly in ROM 120 (in their original state) information regarding the locations of the operating system routines is encoded in compressed form within ROM 120. Upon system start up. this information is decompressed and the trap table 110 is formed in RAM 130."

Lillich '856, 1:66-2:4, *also* Lillich, 2:28-33.

"Because the trap table 110 is resident in RAM 130. individual entries in the trap table 110 can be changed to point to addresses other than the original ROM addresses. This allows the system routines to be replaced or augmented by patches. At startup time the system can load new versions of individual routines (e.g. from the System file or from a floppy disk) into RAM and then patch the trap table in order to redirect any calls to the original system routine to the new versions."

Lillich '856, 2:52-60, *also* Lillich '698, 3:26-34.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Lillich, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Lillich, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Lillich, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Lillich, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Lillich, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lillich discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> Mass storage device 60 is coupled with CPU 52 and may be any mass storage device such as a hard disk system, a floptical disk system, a tape drive or the like. Mass storage device 60 generally includes code fragments such as applications, import libraries, and extensions which are not currently in use by the system. I/O device 66 is coupled to the CPU 52 and may be a network card, a printer port, modem, etc. Additionally there may be a multiplicity of I/O devices such as I/O device 66 coupled to the computer system 50. Design and construction of computer systems such as computer system 50 will be well known to those skilled in the art.

Lillich '856, 9:7-18; *also* Lillich '698, 10:3-14

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Lillich, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Lillich, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Lillich, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Lillich, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Lillich, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Lillich, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lillich discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B3
Invalidity of U.S. Patent 8,090,936 based on Ballard

U.S. Patent No. 5,933,630 to Ballard ("Ballard") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B3
Invalidity of U.S. Patent 8,090,936 based on Ballard

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Ballard, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

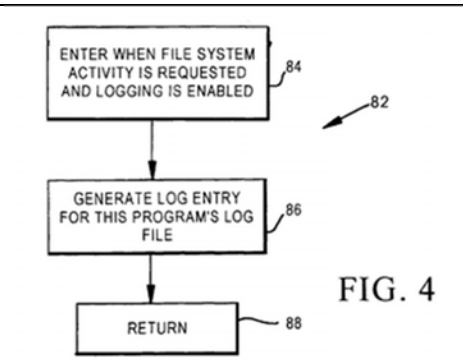


FIG. 4

Ballard, Fig. 4

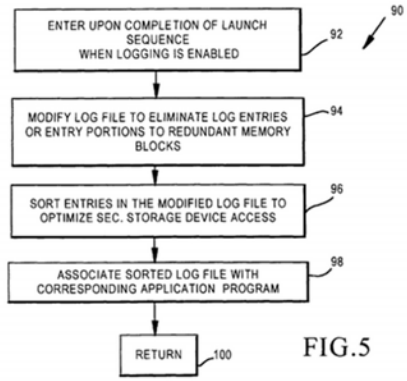


FIG.5

Ballard, Fig. 5

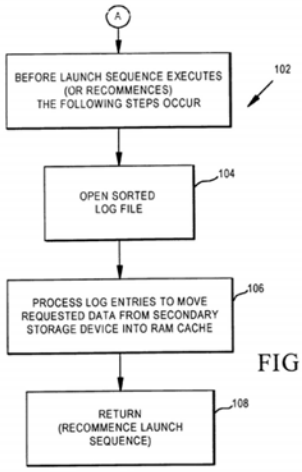


FIG 6

Ballard, Fig. 6

Ballard **Claim 1.1**

"maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

> It is common for an application program for a personal computer to be stored in multiple files on the secondary storage device. There often is an executable file, a preferences file and many other files. Some programs include a data base file or a default data file. During a launch of the program multiple files are opened and select portions are moved from the secondary storage device into the primary storage device. When purchasing a computer program the packaging often specifies the amount of RAM address space (i.e., primary storage device address space) required to be allocated to the program while active. By active it is meant that the program has been launched and is currently processing or is currently able to accept input commands.

Ballard, 1:51-63

> According to another aspect of the invention, the launch access log is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. According to one approach the disk access log entries are sorted by physical address. According to another approach the disk access log entries are grouped by file, then organized by logical address within each group. The processed log file then is stored with the application program.

Ballard, 2:22-30

> The primary storage device 14 typically is a storage device having a faster access time than that of the secondary storage device. An exemplary primary storage device 14 is random access memory (RAM). All or a portion of the RAM serves as a RAM cache 15. Portions of a computer program and/or data files are loaded into the RAM cache 15 to speed up execution of the program and processing of data. Mass produced computer software typically include specifications requiring a minimum amount of RAM required to run the program on a given computer system. During a launch sequence for starting such a computer program, portions of the program are copied from the secondary storage device into RAM.
>
> A launch sequence as used herein means the sequence of steps executed by the computer system during the launch time which pertain to starting up a given computer program and getting the computer ready to accept input commands for the computer program. A launch sequence is executed for a computer program. Different computer programs have different launch sequences. Steps included in a launch sequence include copying portions of the computer program being launched from the secondary storage device to the primary storage device. Other steps may include allocating a port or device to serve as an input source and/or output receptor. The method of this invention for accelerating a program

launch is directed to improving the speed for accessing the secondary storage device during a launch sequence.

Ballard, 3:40-67

The log file corresponds to a specific computer program--the one being launched that triggered such log file to be created. When multiple programs are being launched at the same time, a log file is created for each such program. The interrupt service routine then sets a flag at step 78 to indicate that logging is enabled for such program. At step 80 the program returns. If the trigger for calling the routine 70 was for a secondary storage device access, then the steps at FIG. 4 also are performed before returning.

Log File System Activity

Once the launch of a computer program is detected and a log file is opened, all file system activity is monitored. Specifically, for each operating system call to the file system the call is analyzed to determine to which application being launched, if any, does the call pertain. Exemplary operating system calls to the file system are OPEN, READ, WRITE, and CLOSE. Referring to FIG. 4 routine 82 is entered at step 84 when both file system activity is detected and logging is enabled. If the call pertains to a computer program being launched, then an entry is appended to the appropriate log file (step 86). The routine 82 then returns at step 88.

The log entry includes a file identifier, the logical address(es) specified in the call and the time of access (e.g., system time; index value). Alternatively, the physical memory address(es) corresponding to the logical address(es) are stored in the log entry. In some embodiments, the operating system has already caused the physical addresses to be generated. If not, then the physical addresses are derived from the logical addresses using the operating system's file allocation table to translate the logical address into the physical address.

A logical address (also referred to as a virtual address) is the address which the computer program uses to access memory. A memory management unit translates this address into a physical address before the actual memory is read or written. A physical address is a memory location on the secondary storage device 16.

Ballard, 5:1-37

Detect Launch Completion

When completion of a launch sequence occurs and logging is enabled, then routine 90 is executed. Referring to FIG. 5, the routine enters at step 92. Conventional operating systems have a specific function that is called when a program is ready for normal execution. Under the Macintosh operating system, the function "Get Next Event" is called. For a computer program running under such operating system, such function is called by the computer program when the launch sequence is complete. According to one embodiment of this invention, step 92 is implemented by an interrupt service routine which is called whenever a computer program calls such function. The interrupt service routine clears the logging enabled flag for the corresponding application program.

In an alternative embodiment, access activity to the secondary storage device 16 is monitored to determine when activity has ceased for a threshold length of time (e.g., 3 seconds). Alternatively or in addition activity is monitored to determine when activity has gone below a threshold data throughput rate (e.g., 50 kilobytes per second) for a threshold period of time (e.g., 5 seconds). When there is insufficient activity for such threshold time, then the program launch is considered to be complete. The routine 90 then is executed.

Modify and Sort Log File

Once the launch sequence is determined to have been completed, then the log entry order is re-organized. The purpose is to eliminate redundant accesses to the same memory block and to optimize access time for the secondary storage device. If accesses occur faster, then the launch time (i.e., time elapsed from start to finish of launch sequence is less) is reduced. Thus, the launch of the computer program is accelerated.

Referring to FIG. 5, at step 94 the log file is processed to eliminate log entries or log entry portions to redundant memory blocks.

Ballard, 5:55-6:23

Note in this example that the log includes redundant entries. Entry h2 is a subset of entry d. Entry i2 is the same as entry f. Entry l2 is the same as entry a. It is expected that a redundant access request results from a computer program launch sequence access specifying a different address in the same block as previously accessed.

Frequently the log file will include an entry specifying a single address. Even though only one address is specified, an entire memory block will

be accessed (read or written). This is because the minimum file allocation unit is a memory block cluster. Thus, the smallest portion of the computer program than can be accessed is the cluster size (i.e., cluster size). Entries in the log file are tested at step 94 to identify any redundant accesses to portions of the computer program. To determine whether a later entry is an access to a redundant portion of the computer program, the cluster address for the access is identified. The cluster address is either stored in the log file or is derived from the address stored in the log file. For a FAT drive the cluster size is stored in the drive's boot sector. The boot sector includes information about the layout of the file system used for the drive partition. Following the boot sector are several reserved sectors. Following the reserved sectors is the file allocation table (FAT). Following the FAT are one or more backup copies of the FAT. Following the backup copies is the root directory. The size of the root directory is specified in the boot sector. After the root directory are the user's file and directory area. This is the area divided into clusters. Thus, from the information in the boot sector the starting address of cluster space is determined, along with the size of a cluster. Thus, the address boundaries for each cluster are known.

The address for any given cluster x is given by Ax+B, where A and B are constants determined from the boot partition. Thus, for any given file system call the cluster boundaries for such address are determinable. The log file stores either the accessed address, the cluster number (i.e., x) or the cluster address (e.g., start address, end address or some other identifying address) for each cluster accessed.

Ballard, 6:44-7:14

Following is a description of the redundancy testing. Consider the following access sequence: address 139, address 119, address 110, addresses 120-133, and addresses 138-140. Also consider that the memory block and cluster size is 10 addresses, and that blocks are located at 100-109, 110-119, 120-129, 130-139, 140-149. When address 139 is accessed the contents within the block of addresses 130-139 is accessed. For an embodiment which stores a starting address of the cluster as the cluster address, the log entry includes address 130. When address 119 is accessed the contents within the block of addresses 110-119 is accessed. The log entry for such access includes address 110. The next access in the launch sequence specifies address 110, causing the block 110-119 to be accessed. The cluster address is 110 which is already in the log. This access is a redundant access to a cluster already specified. The redundant access is eliminated by deleting the log entry for address 110. The next access specifies addresses 120-133. This access spans two clusters 120-129 and 130-139. The accesses are logged separately. The log entry for

> cluster address 120 is not redundant. The log entry for cluster address 130, however, is redundant because the first log entry for address 139 encompasses the memory block of addresses 130-139. To eliminate the redundancy, the log entry for address 139 is removed. The next access specifies addresses 139-140. This access also spans two clusters with two log entries. The first of the two is for cluster 130-139. This is a redundant entry and thus is removed from the log fie. The second of the two entries is for cluster 140-149. This is a nonredundant access. When the end of the log file is reached then redundancy testing (step 94) is complete.
>
> In many instances the redundant accesses are eliminated from consideration by a cache operation performed by the computer instead of by step 94. Specifically, when caching is performed the second access to the same cluster will not result in a call to the hard drive because the data is in the cache. The cache will satisfy the request. Requests satisfied by the cache are ignored for purposes of creating a log of accesses. Thus, redundant entries do not get logged.

Ballard, 7:15-53.

See also Ballard, Tables B-C

1.2 initializing a central processing unit of said computer system;	Ballard, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

> The log file corresponds to a specific computer program--the one being launched that triggered such log file to be created. When multiple programs are being launched at the same time, a log file is created for each such program. The interrupt service routine then sets a flag at step 78 to indicate that logging is enabled for such program. At step 80 the program returns. If the trigger for calling the routine 70 was for a secondary storage device access, then the steps at FIG. 4 also are performed before returning.
>
> Log File System Activity
>
> Once the launch of a computer program is detected and a log file is opened, all file system activity is monitored. Specifically, for each operating system call to the file system the call is analyzed to determine to which application being launched, if any, does the call pertain. Exemplary operating system calls to the file system are OPEN, READ, WRITE, and CLOSE. Referring to FIG. 4 routine 82 is entered at step 84 when both file system activity is detected and logging is enabled. If the call pertains to a computer program being launched, then an entry is appended to the appropriate log file (step 86). The routine 82 then returns at step 88.
>
> The log entry includes a file identifier, the logical address(es) specified in the call and the time of access (e.g., system time; index value). Alternatively, the physical memory address(es) corresponding to the logical address(es) are stored in the log entry. In some embodiments, the operating system has already caused the physical addresses to be generated. If not, then the physical addresses are derived from the logical addresses using the operating system's file allocation table to translate the logical address into the physical address.

A logical address (also referred to as a virtual address) is the address which the computer program uses to access memory. A memory management unit translates this address into a physical address before the actual memory is read or written. A physical address is a memory location on the secondary storage device 16.

Ballard, 5:1-37

Detect Launch Completion

When completion of a launch sequence occurs and logging is enabled, then routine 90 is executed. Referring to FIG. 5, the routine enters at step 92. Conventional operating systems have a specific function that is called when a program is ready for normal execution. Under the Macintosh operating system, the function "Get Next Event" is called. For a computer program running under such operating system, such function is called by the computer program when the launch sequence is complete. According to one embodiment of this invention, step 92 is implemented by an interrupt service routine which is called whenever a computer program calls such function. The interrupt service routine clears the logging enabled flag for the corresponding application program.

In an alternative embodiment, access activity to the secondary storage device 16 is monitored to determine when activity has ceased for a threshold length of time (e.g., 3 seconds). Alternatively or in addition activity is monitored to determine when activity has gone below a threshold data throughput rate (e.g., 50 kilobytes per second) for a threshold period of time (e.g., 5 seconds). When there is insufficient activity for such threshold time, then the program launch is considered to be complete. The routine 90 then is executed.

Modify and Sort Log File

Once the launch sequence is determined to have been completed, then the log entry order is re-organized. The purpose is to eliminate redundant accesses to the same memory block and to optimize access time for the secondary storage device. If accesses occur faster, then the launch time (i.e., time elapsed from start to finish of launch sequence is less) is reduced. Thus, the launch of the computer program is accelerated.

Referring to FIG. 5, at step 94 the log file is processed to eliminate log

entries or log entry portions to redundant memory blocks.

Ballard, 5:55-6:23

> Note in this example that the log includes redundant entries. Entry h2 is a subset of entry d. Entry i2 is the same as entry f. Entry l2 is the same as entry a. It is expected that a redundant access request results from a computer program launch sequence access specifying a different address in the same block as previously accessed.
>
> Frequently the log file will include an entry specifying a single address. Even though only one address is specified, an entire memory block will be accessed (read or written). This is because the minimum file allocation unit is a memory block cluster. Thus, the smallest portion of the computer program than can be accessed is the cluster size (i.e., cluster size). Entries in the log file are tested at step 94 to identify any redundant accesses to portions of the computer program. To determine whether a later entry is an access to a redundant portion of the computer program, the cluster address for the access is identified. The cluster address is either stored in the log file or is derived from the address stored in the log file. For a FAT drive the cluster size is stored in the drive's boot sector. The boot sector includes information about the layout of the file system used for the drive partition. Following the boot sector are several reserved sectors. Following the reserved sectors is the file allocation table (FAT). Following the FAT are one or more backup copies of the FAT. Following the backup copies is the root directory. The size of the root directory is specified in the boot sector. After the root directory are the user's file and directory area. This is the area divided into clusters. Thus, from the information in the boot sector the starting address of cluster space is determined, along with the size of a cluster. Thus, the address boundaries for each cluster are known.
>
> The address for any given cluster x is given by Ax+B, where A and B are constants determined from the boot partition. Thus, for any given file system call the cluster boundaries for such address are determinable. The log file stores either the accessed address, the cluster number (i.e., x) or the cluster address (e.g., start address, end address or some other identifying address) for each cluster accessed.

Ballard, 6:44-7:14

> Following is a description of the redundancy testing. Consider the following access sequence: address 139, address 119, address 110, addresses 120-133, and addresses 138-140. Also consider that the memory block and cluster size is 10 addresses, and that blocks are located

> at 100-109, 110-119, 120-129, 130-139, 140-149. When address 139 is accessed the contents within the block of addresses 130-139 is accessed. For an embodiment which stores a starting address of the cluster as the cluster address, the log entry includes address 130. When address 119 is accessed the contents within the block of addresses 110-119 is accessed. The log entry for such access includes address 110. The next access in the launch sequence specifies address 110, causing the block 110-119 to be accessed. The cluster address is 110 which is already in the log. This access is a redundant access to a cluster already specified. The redundant access is eliminated by deleting the log entry for address 110. The next access specifies addresses 120-133. This access spans two clusters 120-129 and 130-139. The accesses are logged separately. The log entry for cluster address 120 is not redundant. The log entry for cluster address 130, however, is redundant because the first log entry for address 139 encompasses the memory block of addresses 130-139. To eliminate the redundancy, the log entry for address 139 is removed. The next access specifies addresses 139-140. This access also spans two clusters with two log entries. The first of the two is for cluster 130-139. This is a redundant entry and thus is removed from the log fie. The second of the two entries is for cluster 140-149. This is a nonredundant access. When the end of the log file is reached then redundancy testing (step 94) is complete.
>
> In many instances the redundant accesses are eliminated from consideration by a cache operation performed by the computer instead of by step 94. Specifically, when caching is performed the second access to the same cluster will not result in a call to the hard drive because the data is in the cache. The cache will satisfy the request. Requests satisfied by the cache are ignored for purposes of creating a log of accesses. Thus, redundant entries do not get logged.

Ballard, 7:15-53.

See also Ballard, Abstract, Tables B-C, Figs. 4-6.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Ballard, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> It is common for an application program for a personal computer to be stored in multiple files on the secondary storage device. There often is an executable file, a preferences file and many other files. Some programs include a data base file or a default data file. During a launch of the program multiple files are opened and select portions are moved from the secondary storage device into the primary storage device. When purchasing a computer program the packaging often specifies the amount of RAM address space (i.e., primary storage device address space) required to be allocated to the program while active. By active it is meant that the program has been launched and is currently processing or is

currently able to accept input commands.

Ballard, 1:51-63

> According to another aspect of the invention, the launch access log is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. According to one approach the disk access log entries are sorted by physical address. According to another approach the disk access log entries are grouped by file, then organized by logical address within each group. The processed log file then is stored with the application program.

Ballard, 2:22-30

> The primary storage device 14 typically is a storage device having a faster access time than that of the secondary storage device. An exemplary primary storage device 14 is random access memory (RAM). All or a portion of the RAM serves as a RAM cache 15. Portions of a computer program and/or data files are loaded into the RAM cache 15 to speed up execution of the program and processing of data. Mass produced computer software typically include specifications requiring a minimum amount of RAM required to run the program on a given computer system. During a launch sequence for starting such a computer program, portions of the program are copied from the secondary storage device into RAM.
>
> A launch sequence as used herein means the sequence of steps executed by the computer system during the launch time which pertain to starting up a given computer program and getting the computer ready to accept input commands for the computer program. A launch sequence is executed for a computer program. Different computer programs have different launch sequences. Steps included in a launch sequence include copying portions of the computer program being launched from the secondary storage device to the primary storage device. Other steps may include allocating a port or device to serve as an input source and/or output receptor. The method of this invention for accelerating a program launch is directed to improving the speed for accessing the secondary storage device during a launch sequence.

Ballard, 3:40-67

> Following is a description of the redundancy testing. Consider the following access sequence: address 139, address 119, address 110, addresses 120-133, and addresses 138-140. Also consider that the memory block and cluster size is 10 addresses, and that blocks are located at 100-109, 110-119, 120-129, 130-139, 140-149. When address

> 139 is accessed the contents within the block of addresses 130-139 is accessed. For an embodiment which stores a starting address of the cluster as the cluster address, the log entry includes address 130. When address 119 is accessed the contents within the block of addresses 110-119 is accessed. The log entry for such access includes address 110. The next access in the launch sequence specifies address 110, causing the block 110-119 to be accessed. The cluster address is 110 which is already in the log. This access is a redundant access to a cluster already specified. The redundant access is eliminated by deleting the log entry for address 110. The next access specifies addresses 120-133. This access spans two clusters 120-129 and 130-139. The accesses are logged separately. The log entry for cluster address 120 is not redundant. The log entry for cluster address 130, however, is redundant because the first log entry for address 139 encompasses the memory block of addresses 130-139. To eliminate the redundancy, the log entry for address 139 is removed. The next access specifies addresses 139-140. This access also spans two clusters with two log entries. The first of the two is for cluster 130-139. This is a redundant entry and thus is removed from the log fie. The second of the two entries is for cluster 140-149. This is a nonredundant access. When the end of the log file is reached then redundancy testing (step 94) is complete.
>
> In many instances the redundant accesses are eliminated from consideration by a cache operation performed by the computer instead of by step 94. Specifically, when caching is performed the second access to the same cluster will not result in a call to the hard drive because the data is in the cache. The cache will satisfy the request. Requests satisfied by the cache are ignored for purposes of creating a log of accesses. Thus, redundant entries do not get logged.

Ballard, 7:15-53.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Ballard, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> A general purpose personal computer typically has many interactive application computer programs installed. A user is able to start-up multiple programs. With regard to an interactive computer program, the term "launch time", as used herein, means the time from when a processor receives a command to start the computer program until the time that the computer program is ready to accept input commands (e.g., user interface commands, batch-entry commands). The term "launch" as used herein means the process performed during the launch time to start up the computer program and get the computer ready to accept input commands for the computer progra[m].

Ballard, 1:38-50

> According to another aspect of the invention, the launch access log is processed by identifying all the file portions accessed during the launch,

> eliminating any duplicate cluster accesses, then sorting the remaining accesses. According to one approach the disk access log entries are sorted by physical address. According to another approach the disk access log entries are grouped by file, then organized by logical address within each group. The processed log file then is stored with the application program.

Ballard, 2:22-30

> The primary storage device 14 typically is a storage device having a faster access time than that of the secondary storage device. An exemplary primary storage device 14 is random access memory (RAM). All or a portion of the RAM serves as a RAM cache 15. Portions of a computer program and/or data files are loaded into the RAM cache 15 to speed up execution of the program and processing of data. Mass produced computer software typically include specifications requiring a minimum amount of RAM required to run the program on a given computer system. During a launch sequence for starting such a computer program, portions of the program are copied from the secondary storage device into RAM.
>
> A launch sequence as used herein means the sequence of steps executed by the computer system during the launch time which pertain to starting up a given computer program and getting the computer ready to accept input commands for the computer program. A launch sequence is executed for a computer program. Different computer programs have different launch sequences. Steps included in a launch sequence include copying portions of the computer program being launched from the secondary storage device to the primary storage device. Other steps may include allocating a port or device to serve as an input source and/or output receptor. The method of this invention for accelerating a program launch is directed to improving the speed for accessing the secondary storage device during a launch sequence.

Ballard, 3:40-67

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Ballard, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> It is common for an application program for a personal computer to be stored in multiple files on the secondary storage device. There often is an executable file, a preferences file and many other files. Some programs include a data base file or a default data file. During a launch of the program multiple files are opened and select portions are moved from the secondary storage device into the primary storage device. When purchasing a computer program the packaging often specifies the amount of RAM address space (i.e., primary storage device address space) required to be allocated to the program while active. By active it is meant that the program has been launched and is currently processing or is currently able to accept input commands.

Ballard, 1:51-63

> According to another aspect of the invention, the launch access log is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. According to one approach the disk access log entries are sorted by physical address. According to another approach the disk access log entries are grouped by file, then organized by logical address within each group. The processed log file then is stored with the application program.

Ballard, 2:22-30

> The primary storage device 14 typically is a storage device having a faster access time than that of the secondary storage device. An exemplary primary storage device 14 is random access memory (RAM). All or a portion of the RAM serves as a RAM cache 15. Portions of a computer program and/or data files are loaded into the RAM cache 15 to speed up execution of the program and processing of data. Mass produced computer software typically include specifications requiring a minimum amount of RAM required to run the program on a given computer system. During a launch sequence for starting such a computer program, portions of the program are copied from the secondary storage device into RAM.
>
> A launch sequence as used herein means the sequence of steps executed by the computer system during the launch time which pertain to starting up a given computer program and getting the computer ready to accept input commands for the computer program. A launch sequence is executed for a computer program. Different computer programs have different launch sequences. Steps included in a launch sequence include copying portions of the computer program being launched from the secondary storage device to the primary storage device. Other steps may include allocating a port or device to serve as an input source and/or output receptor. The method of this invention for accelerating a program launch is directed to improving the speed for accessing the secondary storage device during a launch sequence.

Ballard, 3:40-67

> Following is a description of the redundancy testing. Consider the following access sequence: address 139, address 119, address 110, addresses 120-133, and addresses 138-140. Also consider that the memory block and cluster size is 10 addresses, and that blocks are located at 100-109, 110-119, 120-129, 130-139, 140-149. When address 139 is accessed the contents within the block of addresses 130-139 is accessed. For an embodiment which stores a starting address of the

cluster as the cluster address, the log entry includes address 130. When address 119 is accessed the contents within the block of addresses 110-119 is accessed. The log entry for such access includes address 110. The next access in the launch sequence specifies address 110, causing the block 110-119 to be accessed. The cluster address is 110 which is already in the log. This access is a redundant access to a cluster already specified. The redundant access is eliminated by deleting the log entry for address 110. The next access specifies addresses 120-133. This access spans two clusters 120-129 and 130-139. The accesses are logged separately. The log entry for cluster address 120 is not redundant. The log entry for cluster address 130, however, is redundant because the first log entry for address 139 encompasses the memory block of addresses 130-139. To eliminate the redundancy, the log entry for address 139 is removed. The next access specifies addresses 139-140. This access also spans two clusters with two log entries. The first of the two is for cluster 130-139. This is a redundant entry and thus is removed from the log fie. The second of the two entries is for cluster 140-149. This is a nonredundant access. When the end of the log file is reached then redundancy testing (step 94) is complete.

In many instances the redundant accesses are eliminated from consideration by a cache operation performed by the computer instead of by step 94. Specifically, when caching is performed the second access to the same cluster will not result in a call to the hard drive because the data is in the cache. The cache will satisfy the request. Requests satisfied by the cache are ignored for purposes of creating a log of accesses. Thus, redundant entries do not get logged.

Ballard, 7:15-53.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Ballard, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Ballard, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> A general purpose personal computer typically has many interactive application computer programs installed. A user is able to start-up multiple programs. With regard to an interactive computer program, the term "launch time", as used herein, means the time from when a processor receives a command to start the computer program until the time that the computer program is ready to accept input commands (e.g., user interface commands, batch-entry commands). The term "launch" as used herein means the process performed during the launch time to start up the computer program and get the computer ready to accept input commands

for the computer program.

It is common for an application program for a personal computer to be stored in multiple files on the secondary storage device. There often is an executable file, a preferences file and many other files. Some programs include a data base file or a default data file. During a launch of the program multiple files are opened and select portions are moved from the secondary storage device into the primary storage device. When purchasing a computer program the packaging often specifies the amount of RAM address space (i.e., primary storage device address space) required to be allocated to the program while active. By active it is meant that the program has been launched and is currently processing or is currently able to accept input commands.

Ballard, 1:38-63

The processor 12 serves to execute an operating system and one or more application computer programs. In some embodiments there are multiple processors for executing the operating system and application programs. System utilities and/or operating system extension programs also are executed according to some computer system 10 embodiments. Conventional operating systems include DOS, Windows, Windows NT, MacOS, OS/2 and various UNIX-based operating systems. The display device 18 and input devices 20 enable interaction between a user and the computer system 10. The computer system 10 in the process of executing the operating system and zero or more computer programs defines an operating environment for a user to interact with the computer, operating system and executing computer program. The display device 18 serves as an output device. Exemplary display devices include a CRT monitor or flat panel display. The user inputs commands and data to the computer system 10 via the input devices. Exemplary input devices include a keyboard, a pointing device and a clicking device. Data also is input to the computer via transportable disks or through I/O ports (not shown).

Ballard, 3:10-30.

Log File System Activity

Once the launch of a computer program is detected and a log file is opened, all file system activity is monitored. Specifically, for each operating system call to the file system the call is analyzed to determine to which application being launched, if any, does the call pertain. Exemplary operating system calls to the file system are OPEN, READ, WRITE, and CLOSE. Referring to FIG. 4 routine 82 is entered at step 84 when both file system activity is detected and logging is enabled. If the

call pertains to a computer program being launched, then an entry is appended to the appropriate log file (step 86). The routine 82 then returns at step 88.

The log entry includes a file identifier, the logical address(es) specified in the call and the time of access (e.g., system time; index value). Alternatively, the physical memory address(es) corresponding to the logical address(es) are stored in the log entry. In some embodiments, the operating system has already caused the physical addresses to be generated. If not, then the physical addresses are derived from the logical addresses using the operating system's file allocation table to translate the logical address into the physical address.

A logical address (also referred to as a virtual address) is the address which the computer program uses to access memory. A memory management unit translates this address into a physical address before the actual memory is read or written. A physical address is a memory location on the secondary storage device 16.

Ballard, 5:1-37

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Ballard, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> A general purpose personal computer typically has many interactive application computer programs installed. A user is able to start-up multiple programs. With regard to an interactive computer program, the term "launch time", as used herein, means the time from when a processor receives a command to start the computer program until the time that the computer program is ready to accept input commands (e.g., user interface commands, batch-entry commands). The term "launch" as used herein means the process performed during the launch time to start up the computer program and get the computer ready to accept input commands

for the computer program.

It is common for an application program for a personal computer to be stored in multiple files on the secondary storage device. There often is an executable file, a preferences file and many other files. Some programs include a data base file or a default data file. During a launch of the program multiple files are opened and select portions are moved from the secondary storage device into the primary storage device. When purchasing a computer program the packaging often specifies the amount of RAM address space (i.e., primary storage device address space) required to be allocated to the program while active. By active it is meant that the program has been launched and is currently processing or is currently able to accept input commands.

Ballard, 1:38-63

The processor 12 serves to execute an operating system and one or more application computer programs. In some embodiments there are multiple processors for executing the operating system and application programs. System utilities and/or operating system extension programs also are executed according to some computer system 10 embodiments. Conventional operating systems include DOS, Windows, Windows NT, MacOS, OS/2 and various UNIX-based operating systems. The display device 18 and input devices 20 enable interaction between a user and the computer system 10. The computer system 10 in the process of executing the operating system and zero or more computer programs defines an operating environment for a user to interact with the computer, operating system and executing computer program. The display device 18 serves as an output device. Exemplary display devices include a CRT monitor or flat panel display. The user inputs commands and data to the computer system 10 via the input devices. Exemplary input devices include a keyboard, a pointing device and a clicking device. Data also is input to the computer via transportable disks or through I/O ports (not shown).

Ballard, 3:10-30.

Log File System Activity

Once the launch of a computer program is detected and a log file is opened, all file system activity is monitored. Specifically, for each operating system call to the file system the call is analyzed to determine to which application being launched, if any, does the call pertain. Exemplary operating system calls to the file system are OPEN, READ, WRITE, and CLOSE. Referring to FIG. 4 routine 82 is entered at step 84 when both file system activity is detected and logging is enabled. If the

call pertains to a computer program being launched, then an entry is appended to the appropriate log file (step 86). The routine 82 then returns at step 88.

The log entry includes a file identifier, the logical address(es) specified in the call and the time of access (e.g., system time; index value). Alternatively, the physical memory address(es) corresponding to the logical address(es) are stored in the log entry. In some embodiments, the operating system has already caused the physical addresses to be generated. If not, then the physical addresses are derived from the logical addresses using the operating system's file allocation table to translate the logical address into the physical address.

A logical address (also referred to as a virtual address) is the address which the computer program uses to access memory. A memory management unit translates this address into a physical address before the actual memory is read or written. A physical address is a memory location on the secondary storage device 16.

Ballard, 5:1-37

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Ballard, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Ballard, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

Ballard discloses this limitation:

> Launch time for a computer program is reduced by logging hard disk accesses during an initial launch, then processing the log file to accelerate subsequent launches. The log file is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. The disk access log entries are sorted by physical address or are grouped by file, then organized by logical address within each group. The processed log file is stored with the application program. When the application program is launched thereafter, the processed log file is accessed first. All the disk accesses in the log file are performed moving all the data into RAM cache. When the program launch resumes, the launch occurs faster because all the data is already in cache.

Ballard, Abstract

> According to another aspect of the invention, the launch access log is processed by identifying all the file portions accessed during the launch, eliminating any duplicate cluster accesses, then sorting the remaining accesses. According to one approach the disk access log entries are sorted by physical address. According to another approach the disk access log entries are grouped by file, then organized by logical address within each group. The processed log file then is stored with the application program.

Ballard, 2:22-30

> Following is a description of the redundancy testing. Consider the following access sequence: address 139, address 119, address 110,

addresses 120-133, and addresses 138-140. Also consider that the memory block and cluster size is 10 addresses, and that blocks are located at 100-109, 110-119, 120-129, 130-139, 140-149. When address 139 is accessed the contents within the block of addresses 130-139 is accessed. For an embodiment which stores a starting address of the cluster as the cluster address, the log entry includes address 130. When address 119 is accessed the contents within the block of addresses 110-119 is accessed. The log entry for such access includes address 110. The next access in the launch sequence specifies address 110, causing the block 110-119 to be accessed. The cluster address is 110 which is already in the log. This access is a redundant access to a cluster already specified. The redundant access is eliminated by deleting the log entry for address 110. The next access specifies addresses 120-133. This access spans two clusters 120-129 and 130-139. The accesses are logged separately. The log entry for cluster address 120 is not redundant. The log entry for cluster address 130, however, is redundant because the first log entry for address 139 encompasses the memory block of addresses 130-139. To eliminate the redundancy, the log entry for address 139 is removed. The next access specifies addresses 139-140. This access also spans two clusters with two log entries. The first of the two is for cluster 130-139. This is a redundant entry and thus is removed from the log fie. The second of the two entries is for cluster 140-149. This is a nonredundant access. When the end of the log file is reached then redundancy testing (step 94) is complete.

In many instances the redundant accesses are eliminated from consideration by a cache operation performed by the computer instead of by step 94. Specifically, when caching is performed the second access to the same cluster will not result in a call to the hard drive because the data is in the cache. The cache will satisfy the request. Requests satisfied by the cache are ignored for purposes of creating a log of accesses. Thus, redundant entries do not get logged.

The modified log file next is sorted at step 96. In one embodiment the log entries are grouped according to the file. Each access resulting in a log entry specifies a file and an address. All entries for a given file are grouped together, then arranged within the group by logical address. The groups are arranged chronologically according to which file is specified first in the log file. Alternatively other criteria for ordering the groups is used. In an alternative embodiment instead of grouping the entries by file, all the log entries are sorted according to physical address to optimize access time to the secondary storage device 16. In one embodiment the log entries define a queue processed according to the methods disclosed in U.S. patent application Ser. No. 08/656,372 filed May 31, 1996 for "Estimating Access Time for Hard Drive I/O

> Requests." The contents of such application are incorporated herein by reference and made a part hereof. The log entries are rearranged in an order to optimize access time as described therein.

Ballard, 7:15-8:4.

See also Ballard, 5:1-7:14, Figs. 4-6

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Ballard, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Ballard, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Ballard, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Ballard, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Ballard, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ballard discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> The secondary storage device 16 serves as a permanent storage memory for one or more computer programs 24 to be executed by the processor 12. The secondary storage device 16 also stores data files for use with the application computer programs. Exemplary secondary storage devices include a hard disk drive, floppy disk drive, CD-ROM drive, bernoulli disk drive or other drive system for accessing permanent or replaceable disks, such as floppy disks, magnetic disks, magneto-optical disks, or optical disks.

Ballard, 3:31-39

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Ballard, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Ballard, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Ballard, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Ballard, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Ballard, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Ballard, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ballard discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B4
Invalidity of U.S. Patent 8,090,936 based on Feigenbaum

U.S. Patent No. 5,307,497 to Feigenbaum ("Feigenbaum") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B4
Invalidity of U.S. Patent 8,090,936 based on Feigenbaum

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Feigenbaum, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

> It is also known that the speed at which a ROM can be accessed is several orders of magnitude faster than the speed at which a hard disk or a floppy diskette can be accessed. The invention takes advantage of this known speed difference by storing portions of DOS in a ROM where such portions can be accessed at a speed much faster than the speed at which DOS could be accessed if such portions of DOS were stored on a hard disk or floppy diskette. However, the invention is more than simply substituting a ROM for a disk because of several problems. A first thought that might occur to many persons is that by storing DOS in a ROM, DOS can then be executed directly from the ROM. However, while certain portions of DOS, such as commands in the COMMAND.COM program, can be executed directly from ROM, other portions, such as IBMBIO.COM and IBMDOS.COM are altered and cannot be used in a read only device which precludes any alteration.

Feigenbaum, 1:40-57

1.2 initializing a central processing unit of said computer system;	Feigenbaum, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

> A data processing system, such as a personal computer, contains bootable DOS programs that are stored in a ROM as an alternate file system in which the files are stored in packed format. When the system is powered on, the programs are rapidly booted up or loaded from ROM into RAM and executed to "instantly" (as it appears to the user) place the system in operation.

Feigenbaum, Abstract

> Referring now to the drawings, and first to FIG. 1, the invention is embodied in a personal computer system 10, and resides in the manner in which such system is programmed and operated. It is to be appreciated that such computers are complex and include many components and data processing devices, such as device controllers and adapters, which have been omitted from the drawings for simplicity of illustration. The description provided herein is limited to only those items which are useful in understanding the invention. System 10 includes a microprocessor 12, such as an Intel 80286 microprocessor, which is commercially available and functions in a known manner to execute programs stored in a RAM 14 and a ROM 16. Such ROM preferably comprises a plurality of ROM units which together form ROM 16 and provide sufficient storage capacity for all of the stored information described in more detail below.

Feigenbaum, 4:10-27

> With reference to FIG. 2, a memory map 39 is illustrated based on the full address space of sixteen megabytes (MB) using the twenty-four bit addressing capability of microprocessor 12. The lowest 640 kilobytes (KB) are assigned to RAM 14 and form a storage area, known as the DOS address space, for containing DOS programs in the lowest portion thereof and application programs in the upper or top portion thereof. Addresses immediately below the one megabyte (MB) region are assigned to that portion of ROM 20 containing

> POST 38 and BIOS 40. A video buffer occupies the space immediately above the DOS address space beginning at hex address A0000 H. ROM DOS 34 occupies a 256KB region at the top of the 16 MB address space beginning at hex address FC0000 H. Thus, the lowest one MB region is assigned to the same addresses and functions as such region is used in the prior art, while ROM DOS 34 is assigned to an address space which is not immediately addressable by DOS nor application programs running in the DOS address space. The remaining regions are unused memory addresses. Further, the low 1MB address range is accessible in the real mode of operation of microprocessor 12, while the address range above the real mode range is accessible in protected mode.

Feigenbaum, 5:26-49

> The steps are performed by microprocessor 12 operating under program control to control the operation of the various components within system 10.

Feigenbaum, 8:34-37

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Feigenbaum, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

> A data processing system, such as a personal computer, contains bootable DOS programs that are stored in a ROM as an alternate file system in which the files are stored in packed format. When the system is powered on, the programs are rapidly booted up or loaded from ROM into RAM and executed to "instantly" (as it appears to the user) place the system in operation.

Feigenbaum, Abstract

> When such personal computers are powered up, a power on self test (POST) routine or program is first executed, such program being stored in ROM. Upon the successful completion of such test, portions of DOS are read into the system memory from a hard disk or a floppy diskette and the system is booted up and initialized. The test, booting and initialization process can take many seconds. The present invention is directed to an improvement by which such process is significantly shortened in time. It is also common to start a personal computer by turning on not only a system unit but also a display, and the invention has an objective of accomplishing the start up process so that the system is available for use within the time required to warm up the display. That is, as soon as the system is turned on, the first screen becomes immediately visible on a display and gives the user the appearance of an "instant" load and initialization.

Feigenbaum, 1:22-39

> It is also known that the speed at which a ROM can be accessed is several orders of magnitude faster than the speed at which a hard disk or a floppy diskette can be accessed. The invention takes advantage of this known speed difference by storing portions of DOS in a ROM where such portions can be accessed at a speed much faster than the speed at which DOS could be accessed if such portions of DOS were stored on a hard disk or floppy diskette. However, the

> invention is more than simply substituting a ROM for a disk because of several problems. A first thought that might occur to many persons is that by storing DOS in a ROM, DOS can then be executed directly from the ROM. However, while certain portions of DOS, such as commands in the COMMAND.COM program, can be executed directly from ROM, other portions, such as IBMBIO.COM and IBMDOS.COM are altered and cannot be used in a read only device which precludes any alteration.

Feigenbaum, 1:40-57

> System 10 is further provided with a DOS 32 that comprises two major portions, ROM DOS 34 and DISK DOS 36 respectively stored in ROM 16 and disk 24. ROM DOS 34 includes the programs that are booted into the RAM when the system is initially powered on or when the system is reset, which programs provide the minimum level of operating system support to make system 10 operational to the user. The remaining portions of DOS, i.e. those programs and files that together form a complete DOS, are included in DISK DOS 36. DISK DOS 36 further includes DOS programs similar to those in ROM DOS so that the system can be booted up and operated from disk 24 at the option of the user, as described in more detail hereafter. Also stored in ROM 16 are a basic input/output system (BIOS) 38 and a power on self test (POST) routine 40. When system 10 is first turned on, or when it is restarted, POST 40 first performs the test and upon successful completion, it boots or loads a portion of DOS and transfers control to it in the manner described in detail below. Except for ROM DOS 34, the system as thus far described is constructed and operates in accordance with known principles of the prior art.

Feigenbaum, 5:3-25

> FIG. 3 also illustrates another facet of the invention which is that the data and programs of ROM DOS 34 are stored in ROM 16 in a file system using a packed format. In contrast, any data and files stored on disk 24, including those in RAM DOS 36, are stored in the conventional DOS FAT file system in which information is stored in clusters and sectors. In the FAT file system, if a particular program doesn't have the same number of bytes as are in a sector or cluster, space is wasted. On the average, about one half the number of bytes in a cluster or in a sector are wasted for each file. Since ROM units are more expensive than disk storage, any wasted space is inefficient. Thus, in ROM 16, the ROM DOS 34 files are stored beginning at a segment address, each segment being sixteen bytes. Each succeeding file begins immediately at the next segment location following the end of a preceding file so that on average only eight bytes per file would be wasted, such number being far less than the average waste in a FAT file system. By way of example, suppose file 50 begins at ROM segment X and ends at Y. Then, the succeeding file 52 begins at segment Y+n, where n is less than sixteen.

The other files are similarly stored.

Feigenbaum, 6:60-7:15

> When system 11 is turned on, the general sequence of operations that occur are BIOSPOST 64, ROMBOOT 65, IBMINIT 66, SYSINIT 67, and COMMAND 68. BIOSPOST 64 first performs a power on self test 70 which is the same as or similar to what is done within the prior art. Upon successful completion of such test, step 72 then checks flag 58 to see if the fast boot process of the invention has been selected. If it has not, then the system would boot from disk 24 in accordance with the prior art. If such selection was made, then step 74 copies the first 512 bytes of ROM DOS 34 into RAM 14 at the top of such memory, the bytes thus copied including ROMBOOT 48. Control is then passed via step 75 to ROMBOOT 65 by jumping from POST to instruction 44 and then to the start of ROMBOOT program 48.

Feigenbaum, 8:37-52

> ROMBOOT 65 first loads IBMBIO.COM 50, including RAM LOADER 51, from ROM 16 into RAM 14 by step 76. IBMBIO.COM includes two code segments, IBMINIT and SYSINIT. Step 78 then transfers control to IBMINIT 66. Step 80 initializes the system device drivers and hardware. Step 81 provides or sets up a hook into interrupt handler INT 2BH for the RAM LOADER. Step 82 relocates the initialization routine of IBMBIO.COM 50 to predetermined locations at the top of the DOS address space in RAM 16. Next, step 84 transfers control to SYSINIT 67 which first, by step 86, loads IBMDOS.COM 52 from ROM 16 into RAM 14. Step 88 then executes the initialization routine of IBMDOS.COM. Afterwards, step 90 attaches the IFS Handler as the next available drive which is drive D: because system 10 includes a fixed disk drive. Next, step 91 selects the ROM drive created by the IFS handler, as the defualt drive.

Feigenbaum, 8:53-9:2

> In summary, the invention of a ROM based DOS provides several features and advantages. DOS is rapidly loaded from ROM and executed in RAM to instantly boot up and present the user with a screen from a graphical user interface. Once booted, ROM DOS executes normally and acts the same as standard disk DOS and has the same system compatibility as disk DOS. Further, no DOS installation is required to be done by a user, and the user has no diskette dependencies. The invention is further advantageous by virtue of using the IFS interface and allowing the ROM to appear as a "drive", since this minimizes the number of changes to the basic DOS programs, which reduced developement time and minimized the possibility of error being introduced by new code.

Feigenbaum, 10:18-33

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Feigenbaum, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

> A data processing system, such as a personal computer, contains bootable DOS programs that are stored in a ROM as an alternate file system in which the files are stored in packed format. When the system is powered on, the programs are rapidly booted up or loaded from ROM into RAM and executed to "instantly" (as it appears to the user) place the system in operation.

Feigenbaum, Abstract

> When such personal computers are powered up, a power on self test (POST) routine or program is first executed, such program being stored in ROM. Upon the successful completion of such test, portions of DOS are read into the system memory from a hard disk or a floppy diskette and the system is booted up and initialized. The test, booting and initialization process can take many seconds. The present invention is directed to an improvement by which such process is significantly shortened in time. It is also common to start a personal computer by turning on not only a system unit but also a display, and the invention has an objective of accomplishing the start up process so that the system is available for use within the time required to warm up the display. That is, as soon as the system is turned on, the first screen becomes immediately visible on a display and gives the user the appearance of an "instant" load and initialization.

Feigenbaum, 1:22-39

> System 10 is further provided with a DOS 32 that comprises two major portions, ROM DOS 34 and DISK DOS 36 respectively stored in ROM 16 and disk 24. ROM DOS 34 includes the programs that are booted into the RAM when the system is initially powered on or when the system is reset, which programs provide the minimum level of operating system support to make system 10 operational to the user. The remaining portions of DOS, i.e. those programs and files that together form a complete DOS, are included in DISK DOS 36. DISK

> DOS 36 further includes DOS programs similar to those in ROM DOS so that the system can be booted up and operated from disk 24 at the option of the user, as described in more detail hereafter. Also stored in ROM 16 are a basic input/output system (BIOS) 38 and a power on self test (POST) routine 40. When system 10 is first turned on, or when it is restarted, POST 40 first performs the test and upon successful completion, it boots or loads a portion of DOS and transfers control to it in the manner described in detail below. Except for ROM DOS 34, the system as thus far described is constructed and operates in accordance with known principles of the prior art.

Feigenbaum, 5:3-25

> When system 10 is first setup, it is desireable to customize the system in a manner similar to that in which IBM PS/2 systems are configured. During the course of such customization the user has to make decisions and define what features are to be used. The primary decision pertinent to the invention is whether to boot the system from ROM, in accordance with the invention, or to boot the system from disk 24 in the manner of the prior art. The selection of the option is then coded into a flag 58 which is stored during customization in CMOS RAM 20. Other options for the user are whether the system will be started by using the ROM based CONFIG.SYS 62 and/or AUTOEXEC.BAT 61, or the user defined CONFIG.SYS 102 or AUTOEXEC.BAT 102 files stored on disk 24. Flag 58 will also be set to reflect such other options. Default values are provided for such options so that in the absence of selecting other options, the system will automatically boot from ROM 16 and process the AUTOEXEC.BAT and CONFIG.SYS files therein.

Feigenbaum, 7:15-35

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Feigenbaum, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

> When such personal computers are powered up, a power on self test (POST) routine or program is first executed, such program being stored in ROM. Upon the successful completion of such test, portions of DOS are read into the system memory from a hard disk or a floppy diskette and the system is booted up and initialized. The test, booting and initialization process can take many seconds. The present invention is directed to an improvement by which such process is significantly shortened in time. It is also common to start a personal computer by turning on not only a system unit but also a display, and the invention has an objective of accomplishing the start up process so that the system is available for use within the time required to warm up the display. That is, as soon as the system is turned on, the first screen becomes immediately visible on a display and gives the user the appearance of an "instant" load and initialization.

Feigenbaum, 1:22-39

> One of the objects of the invention is to provide a ROM based DOS that rapidly tests, boots, and initializes a personal computer and appears to the user to "instantly" start up when the computer is turned on.

Feigenbaum, 3:30-33

> When switch 35 is initially turned on, display 30 warms up within a relatively short period and the ROM based booting and system initialization, described in detail below, occurs within the period required for the display to warm up so that at the end of such period, the display screen becomes visible to the user to give the appearance of an instant startup.

Appendix B4
Invalidity of U.S. Patent 8,090,936 based on Feigenbaum

Feigenbaum, 4:63-5:2

> In summary, the invention of a ROM based DOS provides several features and advantages. DOS is rapidly loaded from ROM and executed in RAM to instantly boot up and present the user with a screen from a graphical user interface. Once booted, ROM DOS executes normally and acts the same as standard disk DOS and has the same system compatibility as disk DOS. Further, no DOS installation is required to be done by a user, and the user has no diskette dependencies. The invention is further advantageous by virtue of using the IFS interface and allowing the ROM to appear as a "drive", since this minimizes the number of changes to the basic DOS programs, which reduced developement time and minimized the possibility of error being introduced by new code.

Feigenbaum, 10:18-33

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Feigenbaum, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Feigenbaum, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> With reference to FIG. 2, a memory map 39 is illustrated based on the full address space of sixteen megabytes (MB) using the twenty-four bit addressing capability of microprocessor 12. The lowest 640 kilobytes (KB) are assigned to RAM 14 and form a storage area, known as the DOS address space, for containing DOS programs in the lowest portion thereof and application programs in the upper or top portion thereof. Addresses immediately below the one megabyte (MB) region are assigned to that portion of ROM 20 containing POST 38 and BIOS 40. A video buffer occupies the space immediately above the DOS address space beginning at hex address A0000 H. ROM DOS 34 occupies a 256KB region at the top of the 16 MB address space beginning at hex address FC0000 H. Thus, the lowest one MB region is assigned to the same addresses and functions as such region is used in the prior art, while ROM DOS 34 is assigned to an address space which is not immediately addressable by DOS nor application programs running in the DOS address space. The remaining regions are unused memory addresses. Further, the low 1MB address range is accessible in the real mode of operation of microprocessor 12, while the address range above the real mode range is accessible in protected mode.

Feigenbaum, 5:26-49

> IBMDOS.COM provides application support.

Feigenbaum, 6:11

> RAM LOADER 51, IFS HANDLER 53, and CHECKC.COM 63 are new with the invention, while the remaining programs stored in ROM 16 perform functions previously commercially available. ROM 16 may optionally be of a

> larger size to store additional programs such as a code page switching program, a program providing extended keyboard layout, support for other national languages, etc. It may also contain alternative AUTOEXEC.BAT and CONFIG.SYS files 102 and 104 oriented to the use of such additional programs. The size of ROM 16 may thus vary dependent upon the size of and how many additional support programs are stored. Preferably, those items shown in FIG. 3 are stored in a 128K ROM unit and the additional programs are stored in a second 128 K ROM unit.

Feigenbaum, 6:45-59

> FIG. 4 illustrates the relative position and use of IFS HANDLER 53. Within the prior art, an application program 110 can be thought of as sitting on top of FAT file system 112, which in turn sits on top of disk/diskette driver 114 which overlies the disk 24 containing the actual files and data. Application 110 uses interrupt INT 21H to access the file system through IBMDOS.COM, which acts through IBMBIO.COM to access the disk. In accordance with the invention, IFS HANDLER 53 is interposed at the file system level in the INT 21H process and decides in step 116 whether the desired file system access is to be made to ROM DISK D:. If not, the system proceeds as in the prior art to access the FAT file system. If the desired access is to be made to ROM DISK D:, it is done through the ROM file system 118 and RAM LOADER 51 by programs IBMDOS.COM 52 and IBMBIO.COM 50. The term "ROM DISK" is defined hereby to mean a ROM unit containing images of files storable on a hard disk or floppy diskette, which images are stored in a packed format in a ROM file system, that is different from the FAT file system used to store files on a disk or diskette.

Feigenbaum, 9:39-60

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Feigenbaum, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> With reference to FIG. 2, a memory map 39 is illustrated based on the full address space of sixteen megabytes (MB) using the twenty-four bit addressing capability of microprocessor 12. The lowest 640 kilobytes (KB) are assigned to RAM 14 and form a storage area, known as the DOS address space, for containing DOS programs in the lowest portion thereof and application programs in the upper or top portion thereof. Addresses immediately below the one megabyte (MB) region are assigned to that portion of ROM 20 containing POST 38 and BIOS 40. A video buffer occupies the space immediately above the DOS address space beginning at hex address A0000 H. ROM DOS 34 occupies a 256KB region at the top of the 16 MB address space beginning at hex address FC0000 H. Thus, the lowest one MB region is assigned to the same addresses and functions as such region is used in the prior art, while ROM DOS 34 is assigned to an address space which is not immediately addressable by DOS nor application programs running in the DOS address space. The remaining regions are unused memory addresses. Further, the low 1MB address range is accessible in the real mode of operation of microprocessor 12, while the address range above the real mode range is accessible in protected mode.

Feigenbaum, 5:26-49

> IBMDOS.COM provides application support.

Feigenbaum, 6:11

> RAM LOADER 51, IFS HANDLER 53, and CHECKC.COM 63 are new with the invention, while the remaining programs stored in ROM 16 perform

> functions previously commercially available. ROM 16 may optionally be of a larger size to store additional programs such as a code page switching program, a program providing extended keyboard layout, support for other national languages, etc. It may also contain alternative AUTOEXEC.BAT and CONFIG.SYS files 102 and 104 oriented to the use of such additional programs. The size of ROM 16 may thus vary dependent upon the size of and how many additional support programs are stored. Preferably, those items shown in FIG. 3 are stored in a 128K ROM unit and the additional programs are stored in a second 128 K ROM unit.

Feigenbaum, 6:45-59

> FIG. 4 illustrates the relative position and use of IFS HANDLER 53. Within the prior art, an application program 110 can be thought of as sitting on top of FAT file system 112, which in turn sits on top of disk/diskette driver 114 which overlies the disk 24 containing the actual files and data. Application 110 uses interrupt INT 21H to access the file system through IBMDOS.COM, which acts through IBMBIO.COM to access the disk. In accordance with the invention, IFS HANDLER 53 is interposed at the file system level in the INT 21H process and decides in step 116 whether the desired file system access is to be made to ROM DISK D:. If not, the system proceeds as in the prior art to access the FAT file system. If the desired access is to be made to ROM DISK D:, it is done through the ROM file system 118 and RAM LOADER 51 by programs IBMDOS.COM 52 and IBMBIO.COM 50. The term "ROM DISK" is defined hereby to mean a ROM unit containing images of files storable on a hard disk or floppy diskette, which images are stored in a packed format in a ROM file system, that is different from the FAT file system used to store files on a disk or diskette.

Feigenbaum, 9:39-60

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Feigenbaum, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> Referring now to the drawings, and first to FIG. 1, the invention is embodied in a personal computer system 10, and resides in the manner in which such system is programmed and operated. It is to be appreciated that such computers are complex and include many components and data processing devices, such as device controllers and adapters, which have been omitted from the drawings for simplicity of illustration. The description provided herein is limited to only those items which are useful in understanding the invention. System 10 includes a microprocessor 12, such as an Intel 80286 microprocessor, which is commercially available and functions in a known manner to execute programs stored in a RAM 14 and a ROM 16. Such ROM preferably comprises a plurality of ROM units which together form ROM 16 and provide sufficient storage capacity for all of the stored information described in more detail below.

Feigenbaum, 4:10-27

> The steps are performed by microprocessor 12 operating under program control to control the operation of the various components within system 10.

Feigenbaum, 8:34-37

6. The method of claim 1, further comprising updating the list of boot data.	Feigenbaum, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> Another solution that might occur would be to create the ROM within the DOS address space, or within the first one megabyte of memory mapped space, and then load or copy DOS into a RAM (random access memory) within such address space to occupy and execute from the same space in RAM and operate in the same manner as DOS currently does. The disadvantage of such a solution is that two copies of DOS would then exist in such limited address space, one copy being the unalterable ROM DOS and the other copy being the alterable one that is stored in and executed from the RAM. Having two copies of essentially the same program in such a limited address space would not be efficient use of memory nor acceptable by many users.

Feigenbaum, 1:58-2:3

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Feigenbaum, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Feigenbaum, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Feigenbaum, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Feigenbaum, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Feigenbaum, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

> System 10 also includes a circuit or bus network 18 operatively interconnecting the various elements of the system. A complementary metal oxide semi-conductor (CMOS) RAM 20 is connected to and backed up by a battery 22 to provide non-volatile storage. A disk drive 26 includes a fixed disk 24 for storing information in a FAT file system. Drive 26 and CMOS RAM 20 are also connected to bus 18.

Feigenbaum, 4:54-5:2

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Feigenbaum, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Feigenbaum discloses this limitation:

See Claims 1.3, 1.4, and 1.5 above.

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Feigenbaum, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Feigenbaum, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Feigenbaum, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Feigenbaum, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Feigenbaum, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Feigenbaum discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B5
Invalidity of U.S. Patent 8,090,936 based on Greene

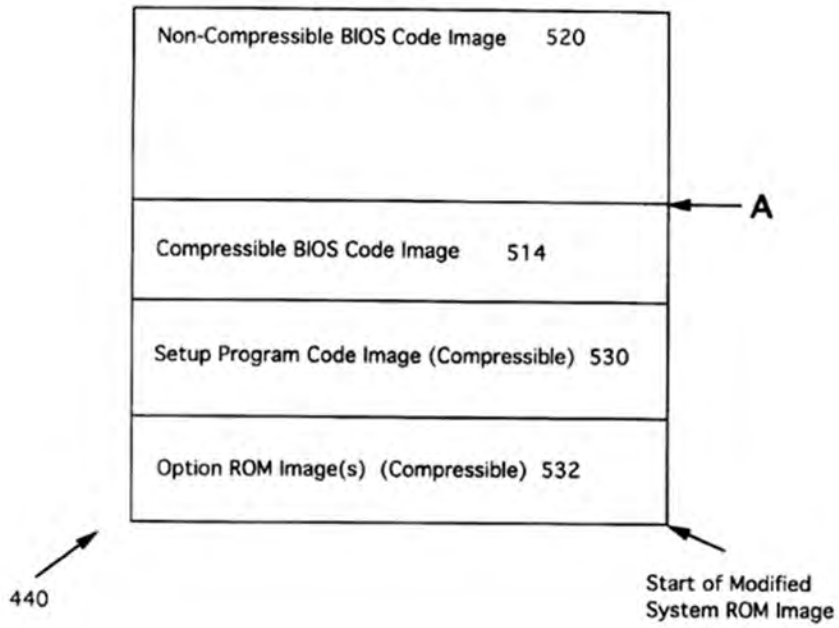
U.S. Patent No. 5,836,013 to Greene ("Greene") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Greene, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:



Greene, Fig. 5.

> "A chipset (platform)-independent method and apparatus for compressing and decompressing a system ROM of a computer (e.g., BIOS, setup program, and one or more option ROMs) are disclosed.

> The setup program, option ROM, and part of the BIOS are compressed using a lossless compression algorithm. A non-compressible portion of the BIOS includes a decompression algorithm and a shadow RAM block table of chipset-specific registers and bit patterns to write-enable and read-enable shadow RAM (RAM that is mapped to the ROM address space). The compressed data is stored in a compressed data block format with the associated location in memory to decompress the compressed data."

Greene, Abstract. *See also* 1:40-1:63.

> "During the BIOS Power-On Self-Test (POST) process of the target computer, the compressed system ROM image is copied to conventional memory (e.g., RAM), and the decompression program is executed. The decompression program write-enables shadow RAM (with reference to the chipset-specific information in the shadow RAM block table), copies the non-compressible BIOS code image from conventional memory to shadow RAM, and read-enable shadow RAM. The decompression program scans the compressed system ROM image for compressed data blocks and decompresses the compressed data therein to the associated locations in memory."

Greene, 1:64-2:8.

> "The setup program code, option ROM code, compressible BIOS code, and non-compressible BIOS code are compiled or assembled and linked to create a modified system ROM image. The starting address of the non-compressible BIOS code image in the modified system ROM image is stored. A lossless compression algorithm is used to compress the modified system ROM image (up to the non-compressible BIOS code image address), thereby generating a compressed system ROM image. Compressed data is stored in a compressed data block comprising the compressed data and various associated information including the location in memory to place the compressed data when decompressed. The compressed system ROM is stored in ROM of a target computer."

Greene, 1:50-63.

> "BIOS code 410, and optionally setup program code 430, and one or more option ROM code modules 432, are each developed, written (e.g., in any computer language such as C or Assembly language) and saved in a computer file. Code and data, as referred to herein, are used interchangeably and comprise computer code and/or data. BIOS code 410 is separated 412 into compressible BIOS code 414 and non-

> compressible BIOS code 420. Non-compressible BIOS code 420 comprises compression-related information table 424, decompression program 422, and shadow RAM block table 1104."

Greene, 3:42-52.

> "Continuing with FIG. 4, compressible BIOS code 414, non-compressible BIOS code 420 (comprising decompression program 422 (1102, 1106, 1108), compression-related information table 424, and shadow RAM block table 1104), and optionally setup program code 430, and option ROM code module(s) 432, are compiled or assembled and linked 434 to generate a modified system ROM image 440."

Greene, 4:64-5:3.

> "Compression-related information table 424, comprises, for example, a preferred compression algorithm to use when compressing system ROM image 440 (see below step 630). In a preferred embodiment, compression algorithm is a lossless decompression algorithm, for example, LZSS or LZARI, which are commercially available compression/ decompression algorithms."

Greene, 3:53-59.

> "Referring now to FIG. 6, modified system ROM image 440 is input to compression utility 600. Compression utility 600 uses compression-related information table 424 (in modified system ROM image 440) to determine 610 a compression algorithm 612 to use in compressing modified system ROM image 440. As discussed above, compression algorithm 612 can be any lossless compression algorithm, but must correspond to the decompression algorithm (1102 below) used in the system ROM."

Greene, 5:19-27, Fig. 6.

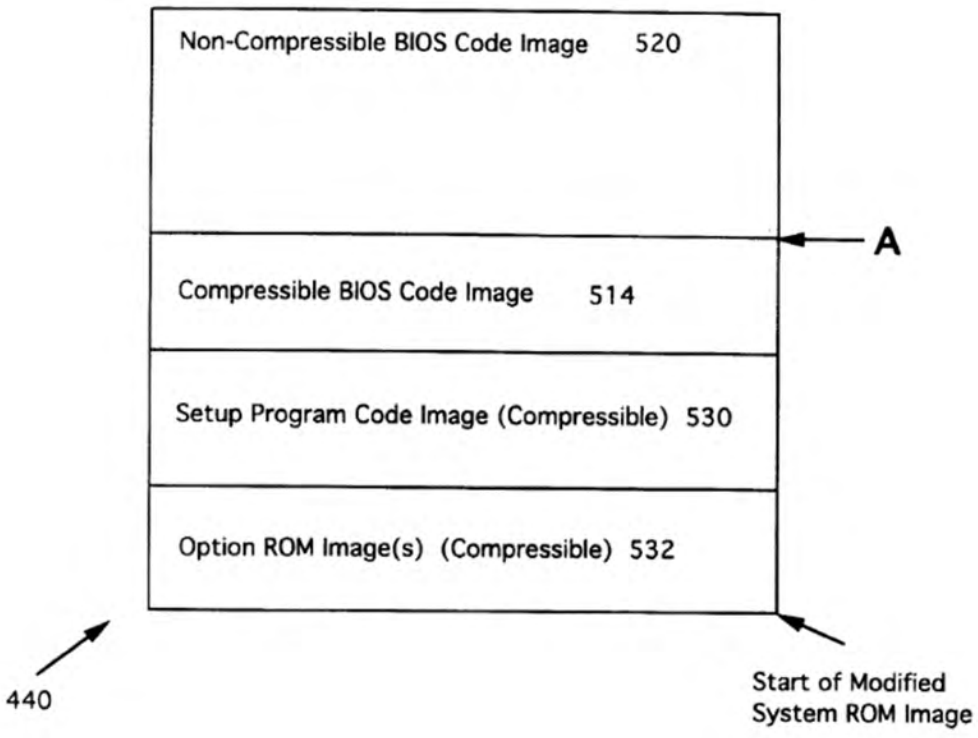
> "Knowing the location of non-compressible BIOS code image 520 in modified system ROM image 440, compression utility 600 uses compression algorithm 612 to compress 630 each compressible image (514, 530, 532) in modified system ROM image 440. Each compressed image is stored in a compressed data block 700."

Greene, 5:31-36.

1.2 initializing a central processing unit of said computer system;	Greene, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:



Greene, Fig. 5.

> "A chipset (platform)-independent method and apparatus for compressing and decompressing a system ROM of a computer (e.g., BIOS, setup program, and one or more option ROMs) are disclosed. The setup program, option ROM, and part of the BIOS are compressed using a lossless compression algorithm. A non-compressible portion of the BIOS includes a decompression algorithm and a shadow RAM block table of chipset-specific registers and bit patterns to write-enable and read-enable shadow RAM (RAM that is mapped to the ROM

address space). The compressed data is stored in a compressed data block format with the associated location in memory to decompress the compressed data."

Greene, Abstract. *See also* 1:40-1:63.

"During the BIOS Power-On Self-Test (POST) process, the compressed system ROM is copied to conventional memory, and the decompression program is executed."

Greene, Abstract. *See also* 1:64-2:13.

"System ROM 210 comprises BIOS (Basic Input/Output System) 310, and optionally setup program 320, and one or more option ROM images for input/output (I/O) devices associated with the computer 332. An "image" is a binary representation of code and/or data of a computer file. BIOS 310 comprises a plurality of routines that initialize the computer hardware and provide primitive I/O services that the operating system and application programs use to manipulate hardware associated with the computer. System setup program 320 is a utility that allows the end user to configure BIOS 310."

Greene, 3:16-28, Fig. 3.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Greene, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

> "The compressed data is stored in a compressed data block format with the associated location in memory to decompress the compressed data. Thus, the data can be decompressed anywhere in memory of a target computer. For example, the BIOS is decompressed to shadow RAM and the setup program is decompressed to conventional memory."

Greene, Abstract.

> "The decompression program scans the compressed system ROM for compressed data blocks and decompresses the compressed data therein to the associated locations in memory. If compressed data is located in shadow RAM, shadow RAM is enabled for writing and reading with reference to the chipset-specific information in the shadow RAM block table. If compressed data was decompressed to conventional memory space, this space is cleared before exiting the POST process."

Greene, Abstract.

> "Compressed data is stored in a compressed data block comprising the compressed data and various associated information including the location in memory to place the compressed data when decompressed. The compressed system ROM is stored in ROM of a target computer."

Greene, 1:58-63.

> "During the BIOS Power-On Self-Test (POST) process of the target computer, the compressed system ROM image is copied to conventional memory (e.g., RAM), and the decompression program is executed. The decompression program write-enables shadow RAM (with reference to

> the chipset-specific information in the shadow RAM block table), copies the non-compressible BIOS code image from conventional memory to shadow RAM, and read-enable shadow RAM. The decompression program scans the compressed system ROM image for compressed data blocks and decompresses the compressed data therein to the associated locations in memory."

Greene, 1:64-2:8.

> "Compressed system ROM image 632 is copied 802 from ROM 130 to conventional memory 110, (e.g., RAM 112)."

Greene, 7:27-29, Fig. 8a.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Greene, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

> "The compressed data is stored in a compressed data block format with the associated location in memory to decompress the compressed data. Thus, the data can be decompressed anywhere in memory of a target computer. For example, the BIOS is decompressed to shadow RAM and the setup program is decompressed to conventional memory."

Greene, Abstract.

> "The decompression program scans the compressed system ROM for compressed data blocks and decompresses the compressed data therein to the associated locations in memory. If compressed data is located in shadow RAM, shadow RAM is enabled for writing and reading with reference to the chipset-specific information in the shadow RAM block table. If compressed data was decompressed to conventional memory space, this space is cleared before exiting the POST process."

Greene, Abstract.

> "During the BIOS Power-On Self-Test (POST) process of the target computer, the compressed system ROM image is copied to conventional memory (e.g., RAM), and the decompression program is executed. The decompression program write-enables shadow RAM (with reference to the the chipset-specific information in the shadow RAM block table), copies the non-compressible BIOS code image from conventional memory to shadow RAM, and read-enables shadow RAM. The decompression program scans the compressed system ROM image for compressed data blocks and decompresses the compressed data therein to the associated locations in memory. If data is located in shadow RAM, shadow RAM is write-enabled and read-enabled (with reference to the chipset-specific information in the shadow RAM block table). If

> compressed data was decompressed to conventional memory, this space is cleared before exiting the POST process."

Greene, 1:64-2:13.

> "Referring to FIG. 10, decompression program 422 comprises a decompression algorithm 1102. Decompression algorithm 1102 corresponds to the compression algorithm used (e.g., LZSS or LZARI)."

Greene, 3:66-4:2.

> "Compressed system ROM 632 image must be decompressed before the compressed data therein can be used on the target computer. In a preferred embodiment, decompression is done early in the POST process."

Greene, 7:21-24.

> "Program execution control is transferred 804 to decompression program 422 (in non-compressible BIOS code image 520 of compressed system ROM image 632)."

Greene, 7:29-32, Fig. 8a. *See also* 7:33-7:64.

> "The decompression scan process repeats 832 until each compressed data block 700 in compressed system ROM image 632 is decompressed. In a preferred embodiment, program execution control is then transferred 834 to BIOS 310, now running in shadow RAM 202. In one embodiment, if data was decompressed to conventional (RAM) memory 110, conventional memory 110 is cleared 836 at the end of the POST process and before the operating system is booted (for compatibility reasons)."

Greene, 7:65-8:21.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Greene, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

> "During the BIOS Power-On Self-Test (POST) process of the target computer, the compressed system ROM image is copied to conventional memory (e.g., RAM), and the decompression program is executed. The decompression program write-enables shadow RAM (with reference to the the chipset-specific information in the shadow RAM block table), copies the non-compressible BIOS code image from conventional memory to shadow RAM, and read-enables shadow RAM. The decompression program scans the compressed system ROM image for compressed data blocks and decompresses the compressed data therein to the associated locations in memory. If data is located in shadow RAM, shadow RAM is write-enabled and read-enabled (with reference to the chipset-specific information in the shadow RAM block table). If compressed data was decompressed to conventional memory, this space is cleared before exiting the POST process."

Greene, 1:64-2:13.

> "Referring to FIG. 10, decompression program 422 comprises a decompression algorithm 1102. Decompression algorithm 1102 corresponds to the compression algorithm used (e.g., LZSS or LZARI)."

Greene, 3:66-4:2.

> "Compressed system ROM 632 image must be decompressed before the compressed data therein can be used on the target computer. In a preferred embodiment, decompression is done early in the POST process."

Greene, 7:21-24.

> "Program execution control is transferred 804 to decompression program 422 (in non-compressible BIOS code image 520 of compressed system ROM image 632)."

Greene, 7:29-32, Fig. 8a. *See also* 7:33-7:64.

> "The decompression scan process repeats 832 until each compressed data block 700 in compressed system ROM image 632 is decompressed. In a preferred embodiment, program execution control is then transferred 834 to BIOS 310, now running in shadow RAM 202. In one embodiment, if data was decompressed to conventional (RAM) memory 110, conventional memory 110 is cleared 836 at the end of the POST process and before the operating system is booted (for compatibility reasons)."

Greene, 7:65-8:21.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Greene, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Greene, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "BIOS 310 comprises a plurality of routines that initialize the computer hardware and provide primitive I/O services that the operating system and application programs use to manipulate hardware associated with the computer."

Greene, 3:21-25

> "Conventional memory 110 comprises a random access memory (RAM) 112 address space that is set aside for use by operating systems and application programs."

Greene, 2:50-53.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Greene, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "BIOS 310 comprises a plurality of routines that initialize the computer hardware and provide primitive I/O services that the operating system and application programs use to manipulate hardware associated with the computer."

Greene, 3:21-25

> "Conventional memory 110 comprises a random access memory (RAM) 112 address space that is set aside for use by operating systems and application programs."

Greene, 2:50-53.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Greene, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Greene, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Greene, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A lossless compression algorithm is used to compress the modified system ROM image (up to the non-compressible BIOS code image address), thereby generating a compressed system ROM image."

Greene, 1:54-58.

> "In a preferred embodiment, compression algorithm is a lossless decompression algorithm, for example, LZSS or LZARI, which are commercially available compression/ decompression algorithms. In general, LZSS decompresses faster than LZARI. LZARI generally compresses data better (smaller) than LZSS. For performance reasons, it is suggested that LZARI decompression code be used with an Intel 80486 or better CPU."

Greene, 3:56-63.

> "Referring to FIG. 10, decompression program 422 comprises a decompression algorithm 1102. Decompression algorithm 1102 corresponds to the compression algorithm used (e.g., LZSS or LZARI)."

Greene, 3:66-4:2.

> "As discussed above, compression algorithm 612 can be any lossless compression algorithm, but must correspond to the decompression algorithm (1102 below) used in the system ROM."

Greene, 5:24-27.

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Greene, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Greene discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A lossless compression algorithm is used to compress the modified system ROM image (up to the non-compressible BIOS code image address), thereby generating a compressed system ROM image."

Greene, 1:54-58.

> "Referring now to FIG. 6, modified system ROM image 440 is input to compression utility 600. Compression utility 600 uses compression-related information table 424 (in modified system ROM image 440) to determine 610 a compression algorithm 612 to use in compressing modified system ROM image 440. As discussed above, compression algorithm 612 can be any lossless compression algorithm, but must correspond to the decompression algorithm (1102 below) used in the system ROM."

Greene, 5:19-27, Fig. 6.

11.1. a processor;	Greene, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Greene, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Greene, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Greene, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Greene, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Greene, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Greene, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Greene, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Greene, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Greene discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B6
Invalidity of U.S. Patent 8,090,936 based on Hillis

U.S. Patent No. 6,421,776 to Hillis ("Hillis") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Hillis, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

> "To increase the effective capacity of BIOS, an initial portion of the power on system reset (POST) code that is required to enable the system memory is stored in ROM in uncompressed form, and substantially the remaining portion of the BIOS code is stored in compressed form."

Hillis, Abstract.

> "After locating a disk with a valid boot record, the BIOS program reads the data stored on the first sector of the disk, and copies that data to specific locations in RAM. This information, found in the same location on every formatted disk, constitutes the DOS boot record. The BIOS then passes control to the boot record which instructs the PC on how to load the two hidden operating system files to RAM (the files named IBMBIO.COM and IBMDOS.COM on IBM computers). After loading other operating system files into RAM to carry out the rest of the boot up sequence, the boot record is no longer needed."

Hillis, 1:46-56.

> "In accordance with an important aspect of the invention, an initial portion of the BIOS code that is required to enable the system memory is in uncompressed form and a remaining portion thereof for carrying out prescribed functions including converting operating signals developed by an operating system executed by the CPU into electrical signals

> compatible with devices that are responsive to signals provided by the CPU to the system bus, is in compressed form."

Hillis, 3:36-43.

> "The next layers of the BIOS ROM consist of compressed set-up data, including descriptive text, and compressed setup code. Next, all but the initial portion of the power on system test (POST) code (termed "phase 2 POST herein") is stored in compressed form, the initial portion of POST (termed "phase 1 POST") being stored in the next lower layer of BIOS ROM."

Hillis, 5:35-41.

> "As shall be described in more detail hereinafter, upon system initialization (bootstrapping), an image of the BIOS ROM is copied to the upper 128 kbyte region of system memory, or shadow RAM, from where system execution takes place for higher operating speed. Shadowing of the BIOS is well known. In accordance with the invention, however, most of the BIOS code is stored in ROM in compressed form."

Hillis, 6:1-8.

> "As has been described, this invention enables compression of most of the contents of the BIOS ROM and boot strapping of the system upon BIOS code decompression, by storing only the initial portion of the POST code in BIOS ROM, sufficient to enable the system memory."

Hillis, 7:66-8:3.

1.2 initializing a central processing unit of said computer system;	Hillis, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

> "Upon system initialization during a cold boot, the uncompressed portion of POST is executed from the ROM to enable the system memory, and then an image of the BIOS code is written to shadow memory."

Hillis, Abstract.

> "In accordance with an aspect of the invention, the portion of the BIOS code that is uncompressed in ROM includes an initial portion of a power on system test (POST) code which is sufficient to enable the system memory, a remaining portion of which is compressed."

Hillis, 3:44-48.

> "Upon cold boot, the initial portion POST is read directly from ROM to enable the system memory, and then an image of the entire BIOS code, the major portion of which is in compressed form, is written to RAM in the system memory, and control is transferred to the image."

Hillis, 3:54-58.

> "The phase 1 POST code, which is stored in uncompressed (unpacked) form, consists of only that portion of POST that is necessary to enable the system memory. That is, under conventional BIOS protocol, the initial portion of POST is first read from the BIOS ROM to enable or "wake up" the system memory, usually composed of CMOS type random access semiconductor memory."

Hillis, 3:42-48.

> As shall be described in more detail hereinafter, upon system initialization (bootstrapping), an image of the BIOS ROM is copied to the upper 128 kbyte region of system memory, or shadow RAM, from Where system execution takes place for higher operating speed."
>
> Hillis: 6:1-4.
>
> *See also* Hillis, 6:16-49.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Hillis, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

> "To increase the effective capacity of BIOS, an initial portion of the power on system reset (POST) code that is required to enable the system memory is stored in ROM in uncompressed form, and substantially the remaining portion of the BIOS code is stored in compressed form."

Hillis, Abstract.

> "Upon system initialization during a cold boot, the uncompressed portion of POST is executed from the ROM to enable the system memory, and then an image of the BIOS code is written to shadow memory."

Hillis, Abstract.

> "In accordance with an important aspect of the invention, an initial portion of the BIOS code that is required to enable the system memory is in uncompressed form and a remaining portion thereof for carrying out prescribed functions including converting operating signals developed by an operating system executed by the CPU into electrical signals compatible with devices that are responsive to signals provided by the CPU to the system bus, is in compressed form."

Hillis, 3:36-43.

> "To reduce the time required for decompression of BIOS code, the code is transferred from ROM to the system memory in compressed form."

Hillis, 3:49-51.

> "Upon cold boot, the initial portion POST is read directly from ROM to

enable the system memory, and then an image of the entire BIOS code, the major portion of which is in compressed form, is written to RAM in the system memory, and control is transferred to the image."

Hillis, 3:54-58.

"The next layers of the BIOS ROM consist of compressed set-up data, including descriptive text, and compressed setup code. Next, all but the initial portion of the power on system test (POST) code (termed "phase 2 POST herein") is stored in compressed form, the initial portion of POST (termed "phase 1 POST") being stored in the next lower layer of BIOS ROM."

Hillis, 5:35-41.

"Then, all the remaining portion of POST and other BIOS code are copied to memory in a region thereof termed "shadow RAM" or "shadow memory."

Hillis, 5:48-55.

"As shall be described in more detail hereinafter, upon system initialization (bootstrapping), an image of the BIOS ROM is copied to the upper 128 kbyte region of system memory, or shadow RAM, from where system execution takes place for higher operating speed. Shadowing of the BIOS is well known. In accordance with the invention, however, most of the BIOS code is stored in ROM in compressed form."

Hillis, 6:1-8.

"Next, for improved performance the entire ROM image is transferred to the system memory (step 50) and the microprocessor cache is turned on (step 52). Control of the system is now transferred to the RAM image of POST, shown in FIG. 4 (step 54)."

Hillis, 6:50-54.

"Next, the uncompressed images are copied from the current region of system RAM back into the shadow RAM (step 58)"

Hillis, 6:61-63.

"As has been described, this invention enables compression of most of the contents of the BIOS ROM and boot strapping of the system upon BIOS code decompression, by storing only the initial portion of the POST

code in BIOS ROM, sufficient to enable the system memory." Hillis, 7:66-8:3.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Hillis, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

> "As BIOS code is needed during the remainder of the boot, the code is selectively decompressed from the shadow memory to another region of the system memory to which control is transferred."

Hillis, Abstract.

> "A further advantage is in performing BIOS code decompression under different boot sceneries, cold and warm, and upon memory conditions of real and protect."

Hillis, 3:21-24.

> "Then, after a jump from one location of the system memory to another, decompression of the code takes place."

Hillis, 3:51-53.

> "As needed, portions of the BIOS code including POST, Setup (if invoked) and then other BIOS routines are selectively decompressed from the shadow memory to another location of the system memory. Normal execution of POST and BIOS then proceeds until the boot is completed."

Hillis, 3:58-63.

> "Mapping from the BIOS ROM to the system memory is followed by decompression only of those BIOS routines that are necessary. Referring to FIG. 5 depicting the BIOS decompressed shadow RAM image, the upper 64K of shadow memory contains decompressed BIOS code, the lowest 32K portion of the upper 128 kbyte block contains decompressed

> phase 2 POST code, decompressed set-up data and code reside at the lowest 128K in system memory, and decompressed video and SES reside as shown."

Hillis, 6:8-16.

> "As BIOS routines are needed, they are now selectively decompressed from system RAM to system RAM (step 56). The POST may be decompressed into any other region of RAM within or outside the system RAM range of addresses including the region ultimately to be occupied by the operating system."

Hillis, 6:55-60.

> "Decompression occurs selectively in a similar manner for other compressed images of the BIOS code, other than setup which in this example has not yet been called (step 60). Control of the system is transferred to the shadow image of POST (step 62)."

Hillis, 6:63-67.

> "As has been described, this invention enables compression of most of the contents of the BIOS ROM and boot strapping of the system upon BIOS code decompression, by storing only the initial portion of the POST code in BIOS ROM, sufficient to enable the system memory."

Hillis, 7:66-8:3.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Hillis, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

> "A further advantage is in performing BIOS code decompression under different boot sceneries, cold and warm, and upon memory conditions of real and protect."

Hillis, 3:21-24.

> "As needed, portions of the BIOS code including POST, Setup (if invoked) and then other BIOS routines are selectively decompressed from the shadow memory to another location of the system memory. Normal execution of POST and BIOS then proceeds until the boot is completed."

Hillis, 3:58-63.

> "As BIOS routines are needed, they are now selectively decompressed from system RAM to system RAM (step 56). The POST may be decompressed into any other region of RAM within or outside the system RAM range of addresses including the region ultimately to be occupied by the operating system."

Hillis, 6:55-60.

> "Decompression occurs selectively in a similar manner for other compressed images of the BIOS code, other than setup which in this example has not yet been called (step 60). Control of the system is transferred to the shadow image of POST (step 62)."

Hillis, 6:63-67.

> "As has been described, this invention enables compression of most of the contents of the BIOS ROM and boot strapping of the system upon BIOS code decompression, by storing only the initial portion of the POST code in BIOS ROM, sufficient to enable the system memory."

Hillis, 7:66-8:3.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Hillis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Hillis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***Look for additional references to application programs***	

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Hillis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***Look for additional references to application programs***	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Hillis, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also ***Look for additional references to controller***

Appendix B6
Invalidity of U.S. Patent 8,090,936 based on Hillis

6. The method of claim 1, further comprising updating the list of boot data.	Hillis, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***add disclosure from 862 patent on updating***	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Hillis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***add disclosure on Lempel-Ziv***	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Hillis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also ***add disclosure on a plurality of encoders***

11.1. a processor;	Hillis, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Hillis, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Hillis, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hillis discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also ***add disclosure of non-volatile memory to the extent not in claim 1 already***

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Hillis, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Hillis, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Hillis, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Hillis, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Hillis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Hillis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hillis discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B7
Invalidity of U.S. Patent 8,090,936 based on Horning

U.S. Patent No. 5,420,998 to Horning ("Horning") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B7
Invalidity of U.S. Patent 8,090,936 based on Horning

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Horning, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

> "The hard disk 42 data source provides persistent or nonvolatile data storage together with the necessary data accessing mechanisms and logic for reading and writing data. The buffer memory 50 is preferably solid state memory providing data storage plus data transfer rates much higher than the hard disk 42, in fact, preferably rates greater than the transfer rate of the host 18. Thus, both the cache memory 54 and the SSD memory 58 are intended to supply storage for those data items that are accessed frequently by the host 18. The cache memory 54 stores frequently accessed copies of hard disk 42 data items, preferably without any control bits relating to hard disk 42 data formatting. The SSD memory 58 stores frequently accessed data items in a "disk-like" format."

Horning, 5:17-31.

> "Depending on the dual disk drive 10 configuration, the microcode instructions for the processor 30 can reside in either a read only memory (ROM) associated with processor 30 or, alternatively, the microcode can reside on the hard disk 42."

Horning, 7:46-50, Fig. 2.

> "In step 68, the processor 30 requests the disk data transfer unit 38 to retrieve the SSD memory 58, or more generally the buffer memory 50, configuration parameter values from a manufacturer specified location on

the hard disk 42."

Horning, 7:59-63. *See also* Horning, 7:59-8:2.

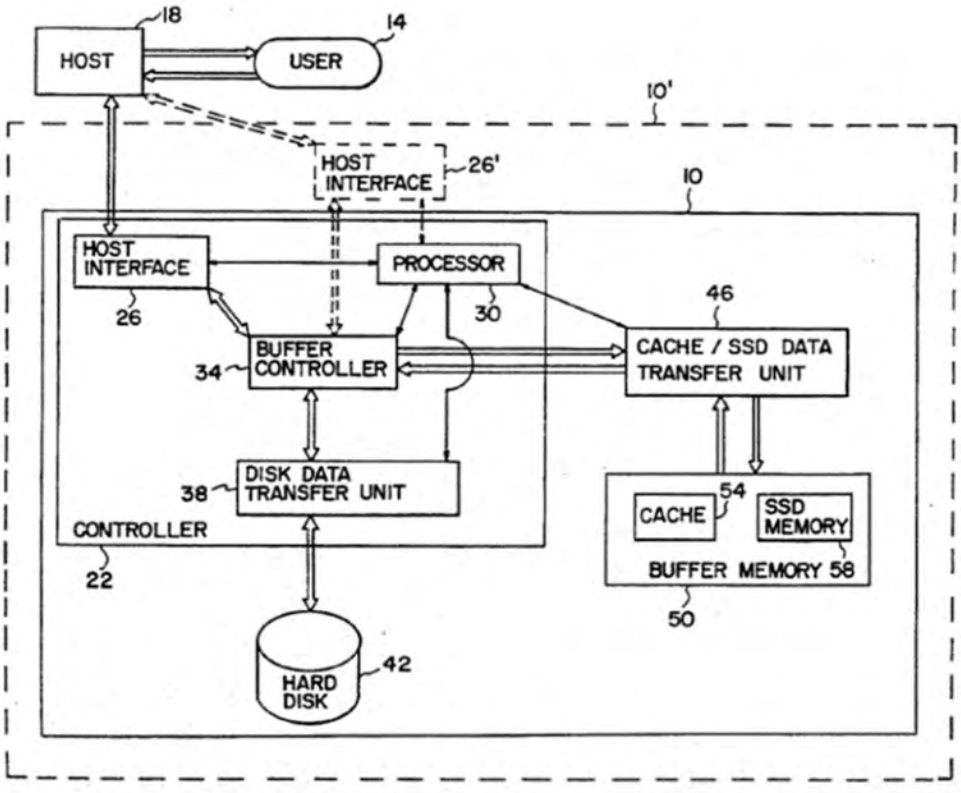
> "The configuring is similar to step 220 in Fig. 4 in that it includes providing the cache/SSD data transfer unit 46 with: (i) the number of blocks to be read, "blk_cnt," (ii) the initial header location address, in the SSD memory 58 where the data is to be read as determined from the physical address established in step 312 above, (iii) the expected value of the sector header at this location, and (iv) a signal indicating that the data to be received from the SSD memory 58 is to be transferred to the host 18 without header and error correction bits."

Horning, 13:2-12.

1.2 initializing a central processing unit of said computer system;	Horning, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:



Horning, Fig. 1.

> "Additional software is also provided to initialize the dual disk drive and to assure SSD data integrity upon power failure. To make the initialization of the dual disk drive as simple as possible for system administration personnel, the initialization procedure has been constructed so that the dual disk drive can be initialized comparable to a conventional hard disk drive. In this case, the parameters for configuring the dual disk drive are supplied with default values during dual disk drive

initialization."

Horning, 3:19-28. *See also*, Horning, 3:28-35.

> "Returning to the controller 22, it includes: a host interface 26, a buffer controller 34, a disk data transfer unit 38 and a processor 30."

Horning, 5:49-52.

> "Referring now to FIG. 2, a preferred procedure is presented for initialization of the dual disk drive 10. In step 60 a user physically switches on power to the dual disk drive 10. Assuming the data connection between the dual disk drive 10 and the host 18 has been physically established, the user preferably activates all further initialization tasks from the host 18. In step 64, both the hard disk 42 and the processor 30 become fully functional. That is, the hard disk 42 is directed to spin-up (i.e. accelerate rotation of its included magnetic storage disk until the proper rotation rate is attained to allow data to be transferred) and the processor 30 is directed to boot-up."

Horning, 7:34-46, Fig. 2.

> "If the microcode resides in ROM, then the processor 30 can boot-up simultaneously with the hard disk 42 spin-up. Otherwise, the processor 30 boot-up will occur immediately after spin-up and the microcode is downloaded into processor 30."

Horning, 7:50-55.

> "Upon completion of the SSD memory 58 formatting, the cache/SSD data transfer unit 46 interrupts the processor 30 signaling that SSD memory 58 initialization is complete."

Horning, 8:16-19.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Horning, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

> "The hard disk 42 data source provides persistent or nonvolatile data storage together with the necessary data accessing mechanisms and logic for reading and writing data. The buffer memory 50 is preferably solid state memory providing data storage plus data transfer rates much higher than the hard disk 42, in fact, preferably rates greater than the transfer rate of the host 18. Thus, both the cache memory 54 and the SSD memory 58 are intended to supply storage for those data items that are accessed frequently by the host 18. The cache memory 54 stores frequently accessed copies of hard disk 42 data items, preferably without any control bits relating to hard disk 42 data formatting. The SSD memory 58 stores frequently accessed data items in a "disk-like" format."

Horning, 5:17-31.

> "If the data transfer is directed to a hard disk 42 location, then preferably the transfer must be through the cache memory 54 regardless of whether the host request is a read or write. That is, if a data read is requested then a- valid copy of the requested data must either reside in the cache memory 54 or be transferred into the cache memory 54 from the hard disk 42 via the disk data transfer unit 38, the buffer controller 34 and the cache/SSD data transfer unit 46. In any case, the requested data is subsequently transferred from the cache memory 54 to the host 18, via the cache/SSD data transfer unit 46, the buffer controller 34 and the host interface 26."

Horning, 6:33-45.

> "If the microcode resides in ROM, then the processor 30 can boot-up simultaneously with the hard disk 42 spin-up. Otherwise, the processor

> 30 boot-up will occur immediately after spin-up and the microcode is downloaded into processor 30."

Horning, 7:50-55.

> "In step 68, the processor 30 requests the disk data transfer unit 38 to retrieve the SSD memory 58, or more generally the buffer memory 50, configuration parameter values from a manufacturer specified location on the hard disk 42."

Horning, 7:59-63. *See also* Horning, 7:59-8:2.

> "Referring to FIG. 5B, in step 324 the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. In step 328, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer data from the SSD memory 58 to the host 19. The requested data is transferred from the SSD memory 58 through the cache/SSD data transfer unit 46 a sector at a time."

Horning, 13:12-22.

> "In this step the processor 30 determines whether or not a current version of the data to be read is in the cache 54. If so, then in step 344, the processor assigns the address of the data cache location to the variable, -"cache_loc." In step 348, the processor 30 configures the cache/SSD data transfer unit 46 to read "blk_cnt" data blocks from the cache 54 and transfer them to the buffer controller 34 without any modification or formatting. In step 352, the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_cnt" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. Referring to FIG. SC, in step 356, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer the data blocks from the cache 54 to the host 18."

Horning, 13:47-62, Fig. 5A-5C.

> "In step 364, the processor 30 determines a cache location where the data can be written from the hard disk 42 and assigns the address of this location to "cache__loc." In step 368, the processor 30 configures the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer "blk_cnt" number of data blocks from the disk data transfer unit 38 to the cache 54. Note, in this step, the cache/SSD data transfer unit 46 is

configured to write the data to the cache 54 without any modification or formatting. In step 372, the processor 30 waits for an interrupt from the disk data transfer unit 38 indicating the seek command has completed and successfully located the data to be read. In step 376, the processor instructs the disk transfer unit 38, the buffer controller 34 and the cache/SSD data transfer unit 42 to transfer the data located on the hard disk 42 to the cache 54. Preferably after a predetermined number of data bytes have been transferred to the cache 54, the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 will commence multiplexing the data transfer to the cache 54 with an additional data transfer of this same data from the cache 54 to the host 18."

Horning, 14:1-23, Fig. 5A-5C.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Horning, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

> "The hard disk 42 data source provides persistent or nonvolatile data storage together with the necessary data accessing mechanisms and logic for reading and writing data. The buffer memory 50 is preferably solid state memory providing data storage plus data transfer rates much higher than the hard disk 42, in fact, preferably rates greater than the transfer rate of the host 18. Thus, both the cache memory 54 and the SSD memory 58 are intended to supply storage for those data items that are accessed frequently by the host 18. The cache memory 54 stores frequently accessed copies of hard disk 42 data items, preferably without any control bits relating to hard disk 42 data formatting. The SSD memory 58 stores frequently accessed data items in a "disk-like" format."

Horning, 5:17-31.

> "If the data transfer is directed to a hard disk 42 location, then preferably the transfer must be through the cache memory 54 regardless of whether the host request is a read or write. That is, if a data read is requested then a- valid copy of the requested data must either reside in the cache memory 54 or be transferred into the cache memory 54 from the hard disk 42 via the disk data transfer unit 38, the buffer controller 34 and the cache/SSD data transfer unit 46. In any case, the requested data is subsequently transferred from the cache memory 54 to the host 18, via the cache/SSD data transfer unit 46, the buffer controller 34 and the host interface 26."

Horning, 6:33-45.

> "If the microcode resides in ROM, then the processor 30 can boot-up simultaneously with the hard disk 42 spin-up. Otherwise, the processor 30 boot-up will occur immediately after spin-up and the microcode is downloaded into processor 30."

Horning, 7:50-55.

> "In step 68, the processor 30 requests the disk data transfer unit 38 to retrieve the SSD memory 58, or more generally the buffer memory 50, configuration parameter values from a manufacturer specified location on the hard disk 42."

Horning, 7:59-63. *See also* Horning, 7:59-8:2.

> "Referring to FIG. 5B, in step 324 the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. In step 328, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer data from the SSD memory 58 to the host 19. The requested data is transferred from the SSD memory 58 through the cache/SSD data transfer unit 46 a sector at a time."

Horning, 13:12-22.

> "In this step the processor 30 determines whether or not a current version of the data to be read is in the cache 54. If so, then in step 344, the processor assigns the address of the data cache location to the variable, -"cache_loc." In step 348, the processor 30 configures the cache/SSD data transfer unit 46 to read "blk_cnt" data blocks from the cache 54 and transfer them to the buffer controller 34 without any modification or formatting. In step 352, the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_cnt" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. Referring to FIG. SC, in step 356, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer the data blocks from the cache 54 to the host 18."

Horning, 13:47-62, Fig. 5A-5C.

> "In step 364, the processor 30 determines a cache location where the data can be written from the hard disk 42 and assigns the address of this location to "cache__loc." In step 368, the processor 30 configures the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer "blk_cnt" number of data blocks from the disk data transfer unit 38 to the cache 54. Note, in this step, the cache/SSD data transfer unit 46 is configured to write the data to the cache 54 without any modification or

> formatting. In step 372, the processor 30 waits for an interrupt from the disk data transfer unit 38 indicating the seek command has completed and successfully located the data to be read. In step 376, the processor instructs the disk transfer unit 38, the buffer controller 34 and the cache/SSD data transfer unit 42 to transfer the data located on the hard disk 42 to the cache 54. Preferably after a predetermined number of data bytes have been transferred to the cache 54, the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 will commence multiplexing the data transfer to the cache 54 with an additional data transfer of this same data from the cache 54 to the host 18."

Horning, 14:1-23, Fig. 5A-5C.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	REFERENCE, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

> "Referring to FIG. 5B, in step 324 the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. In step 328, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer data from the SSD memory 58 to the host 19. The requested data is transferred from the SSD memory 58 through the cache/SSD data transfer unit 46 a sector at a time."

Horning, 13:12-22.

> "In this step the processor 30 determines whether or not a current version of the data to be read is in the cache 54. If so, then in step 344, the processor assigns the address of the data cache location to the variable, -"cache_loc." In step 348, the processor 30 configures the cache/SSD data transfer unit 46 to read "blk_cnt" data blocks from the cache 54 and transfer them to the buffer controller 34 without any modification or formatting. In step 352, the processor 30 configures the host interface 26 and the buffer controller 34 to transfer "blk_cnt" number of data blocks from the cache/SSD data transfer unit 46 to the host 18. Referring to FIG. SC, in step 356, the processor 30 instructs the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer the data blocks from the cache 54 to the host 18."

Horning, 13:47-62, Fig. 5A-5C.

> "In step 364, the processor 30 determines a cache location where the data can be written from the hard disk 42 and assigns the address of this location to "cache__loc." In step 368, the processor 30 configures the buffer controller 34 and the cache/SSD data transfer unit 46 to transfer "blk_cnt" number of data blocks from the disk data transfer unit 38 to the cache 54. Note, in this step, the cache/SSD data transfer unit 46 is configured to write the data to the cache 54 without any modification or formatting. In step 372, the processor 30 waits for an interrupt from the disk data transfer unit 38 indicating the seek command has completed and successfully located the data to be read. In step 376, the processor instructs the disk transfer unit 38, the buffer controller 34 and the cache/SSD data transfer unit 42 to transfer the data located on the hard disk 42 to the cache 54. Preferably after a predetermined number of data bytes have been transferred to the cache 54, the host interface 26, the buffer controller 34 and the cache/SSD data transfer unit 46 will commence multiplexing the data transfer to the cache 54 with an additional data transfer of this same data from the cache 54 to the host 18."

Horning, 14:1-23, Fig. 5A-5C.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Horning, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Horning, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "In particular, since most of the information for anticipating future data requirements of host applications reside in the host, for most efficient cache use, the host must make decisions as to what data should be retained in the cache."

Horning, 1:31-35.

> "The embodiment described hereinabove is further intended to explain the best mode presently known of practicing the invention and to enable other skilled in the art to utilized the invention is such, or other embodiments, and with the various modifications required by their particular application or uses of the invention. It is intended that the appended claims be construed to include alternative embodiments to the extent permitted by the prior art."

Horning, 14:47-55.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Horning, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "In particular, since most of the information for anticipating future data requirements of host applications reside in the host, for most efficient cache use, the host must make decisions as to what data should be retained in the cache."

Horning, 1:31-35.

> "The embodiment described hereinabove is further intended to explain the best mode presently known of practicing the invention and to enable other skilled in the art to utilized the invention is such, or other embodiments, and with the various modifications required by their particular application or uses of the invention. It is intended that the appended claims be construed to include alternative embodiments to the extent permitted by the prior art."

Horning, 14:47-55.

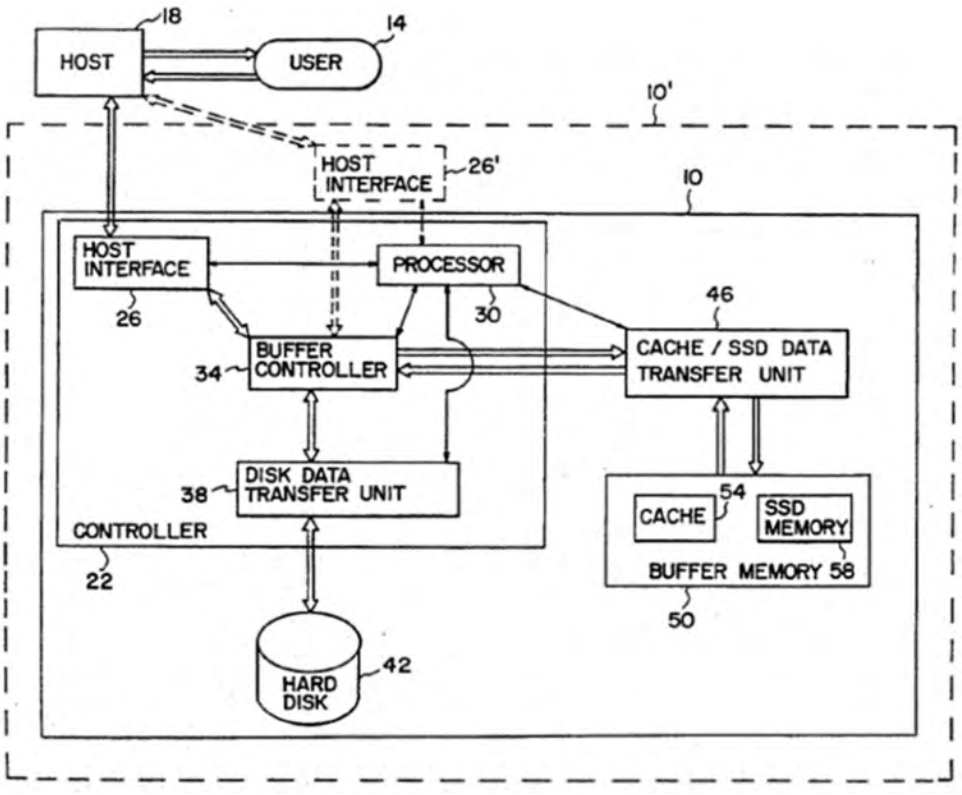
5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Horning, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Horning discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also



Horning, Fig. 1. *See also* Fig. 4B, 4C, 5B, 5C.

"The disk controller, therefore, is left with the responsibility of managing

> the data transfer between the cache, the hard disk and the host. In such cases the controller usually supplies a relatively simple, and less than optimal, data caching strategy such as the strategy where the most recently accessed data is always stored in the cache regardless of the access frequency."

Horning, 1:42-49. *See also* Horning, 2:14-26, 3:50-51, 4:4-6, 5:5-17, 5:42-6:10, 6:37-7:18, 10:37-14:29.

6. The method of claim 1, further comprising updating the list of boot data.	Horning, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Horning, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Horning, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Horning, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Horning, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Horning, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Horning, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Horning, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Horning, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Horning, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

Appendix B7
Invalidity of U.S. Patent 8,090,936 based on Horning

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Horning, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B7
Invalidity of U.S. Patent 8,090,936 based on Horning

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Horning, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Horning discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B8
Invalidity of U.S. Patent 8,090,936 based on Hovis

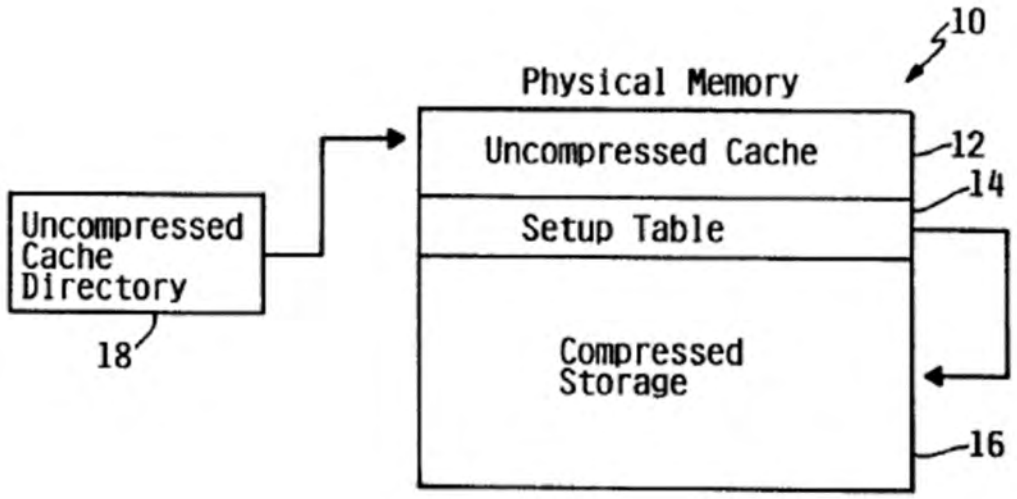
U.S. Patent No. 5,812,817 to Hovis ("Hovis") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Hovis, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:



Hovis, Fig. 1. Hovis, 2:34-43.

> "The architecture includes a cache section, a setup table, and a compressed storage, all of which are partitioned from a computer memory. The cache section is used for storing uncompressed data and is a fast access memory for data which is frequently referenced. The compressed storage is used for storing compressed data. The setup table is used for specifying locations of compressed data stored within the

compressed storage."

Hovis, Abstract. Hovis, 1:39-51, 2:18-24.

> "The function of the setup table 14 is to provide a directory for the memory locations which are in the compressed storage 16. When an access to the memory 10 misses the cache 12, it generates an access to the setup table 14. The data from this access contains the location of the data within compressed storage 16. The address results in an access to the compressed storage 16 which in turn results in compressed data being accessed and processed by the compression engine 28, which performs compression and decompression on the data. The now uncompressed data is placed in the uncompressed cache 12 and transferred to the requesting element (for a fetch), or updated and maintained within the uncompressed cache 12 (for a store)."

Hovis, 3:3-15. *See also* Fig. 2, 3:23-38.

> "Fig. 4 shows a basic dataflow structure of the memory 10 with a hardware assist compression engine 28. A memory control 24 interfaces with the memory 10, compression engine 28, and processor units 22. The purpose of a hardware based compression engine 28 is to provide the necessary bandwidth and latency required by memory entities that reside close to the processor data/instruction units (i.e. L1, L2, L3). Typical software based compression techniques have limited bandwidth and significantly greater latency which would substantially degrade the processor performance."

Hovis, 3:39-50, Fig. 4.

> "When an access to memory misses the uncompressed cache 12, it results in an access to the setup table 14 in order to fetch pointer information for indicating where the compressed data resides in the compressed storage 16. This data is sent to the compression engine which in turn uses the addresses to fetch the necessary compressed data from compressed storage 16. The data is then uncompressed and sent back to the requesting element and uncompressed cache in the memory by the compression engine 28."

Hovis, 3:63-4:4.

> "Also, the directory for the uncompressed cache must be updated when new locations are added or when old (the least recently used) locations are removed to compressed memory 16."

Hovis, 4:18-21.

> "If the cache was not full, the system uses the setup table to retrieve the compressed data (34) and sends the compressed data to a compression engine for decompressing (38). The system continues as if data was in the cache."

Hovis, 4:40-44.

> "Aspects of the control of the hardware assisted memory compression can be implemented either by hardware, software (operating system, namely memory management), or a combination of both. The present invention is not dependent upon a particular hardware or software control scheme. The compression engine is preferably implemented in hardware in order to perform the compression and decompression in a timely fashion and with sufficient bandwidth."

Hovis, 5:4-11.

> "The implementation involves a hardware compression engine and the control can be via hardware, software, or a combination of both."

Hovis, 5:27-30.

1.2 initializing a central processing unit of said computer system;	Hovis, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Hovis, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

> "The architecture includes a cache section, a setup table, and a compressed storage, all of which are partitioned from a computer memory. The cache section is used for storing uncompressed data and is a fast access memory for data which is frequently referenced. The compressed storage is used for storing compressed data. The setup table is used for specifying locations of compressed data stored within the compressed storage. A high speed uncompressed cache directory is coupled to the memory for determining if data is stored in the cache section or compressed storage and for locating data in the cache."

Hovis, Abstract. Hovis, 1:39-51, 2:18-24, 2:34-43.

> "The function of the setup table 14 is to provide a directory for the memory locations which are in the compressed storage 16. When an access to the memory 10 misses the cache 12, it generates an access to the setup table 14. The data from this access contains the location of the data within compressed storage 16. The address results in an access to the compressed storage 16 which in turn results in compressed data being accessed and processed by the compression engine 28, which performs compression and decompression on the data. The now uncompressed data is placed in the uncompressed cache 12 and transferred to the requesting element (for a fetch), or updated and maintained within the uncompressed cache 12 (for a store)."

Hovis, 3:3-15.

> "When an access to memory misses the uncompressed cache 12, it results in an access to the setup table 14 in order to fetch pointer information for indicating where the compressed data resides in the compressed storage

> 16. This data is sent to the compression engine which in turn uses the addresses to fetch the necessary compressed data from compressed storage 16. The data is then uncompressed and sent back to the requesting element and uncompressed cache in the memory by the compression engine 28."

Hovis, 3:63-4:4.

> "If the cache was not full, the system uses the setup table to retrieve the compressed data (34) and sends the compressed data to a compression engine for decompressing (38). The system continues as if data was in the cache."

Hovis, 4:40-44.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Hovis, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

> "The architecture includes a cache section, a setup table, and a compressed storage, all of which are partitioned from a computer memory. The cache section is used for storing uncompressed data and is a fast access memory for data which is frequently referenced. The compressed storage is used for storing compressed data. The setup table is used for specifying locations of compressed data stored within the compressed storage. A high speed uncompressed cache directory is coupled to the memory for determining if data is stored in the cache section or compressed storage and for locating data in the cache."

Hovis, Abstract. Hovis, 1:39-51, 2:18-24, 2:34-43.

> "The function of the setup table 14 is to provide a directory for the memory locations which are in the compressed storage 16. When an access to the memory 10 misses the cache 12, it generates an access to the setup table 14. The data from this access contains the location of the data within compressed storage 16. The address results in an access to the compressed storage 16 which in turn results in compressed data being accessed and processed by the compression engine 28, which performs compression and decompression on the data. The now uncompressed data is placed in the uncompressed cache 12 and transferred to the requesting element (for a fetch), or updated and maintained within the uncompressed cache 12 (for a store)."

Hovis, 3:3-15.

> "When an access to memory misses the uncompressed cache 12, it results in an access to the setup table 14 in order to fetch pointer information for indicating where the compressed data resides in the compressed storage

> 16. This data is sent to the compression engine which in turn uses the addresses to fetch the necessary compressed data from compressed storage 16. The data is then uncompressed and sent back to the requesting element and uncompressed cache in the memory by the compression engine 28."

Hovis, 3:63-4:4.

> "If the cache was not full, the system uses the setup table to retrieve the compressed data (34) and sends the compressed data to a compression engine for decompressing (38). The system continues as if data was in the cache."

Hovis, 4:40-44.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Hovis, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

> "The function of the setup table 14 is to provide a directory for the memory locations which are in the compressed storage 16. When an access to the memory 10 misses the cache 12, it generates an access to the setup table 14. The data from this access contains the location of the data within compressed storage 16. The address results in an access to the compressed storage 16 which in turn results in compressed data being accessed and processed by the compression engine 28, which performs compression and decompression on the data. The now uncompressed data is placed in the uncompressed cache 12 and transferred to the requesting element (for a fetch), or updated and maintained within the uncompressed cache 12 (for a store)."

Hovis, 3:3-15.

> "When an access to memory misses the uncompressed cache 12, it results in an access to the setup table 14 in order to fetch pointer information for indicating where the compressed data resides in the compressed storage 16. This data is sent to the compression engine which in turn uses the addresses to fetch the necessary compressed data from compressed storage 16. The data is then uncompressed and sent back to the requesting element and uncompressed cache in the memory by the compression engine 28."

Hovis, 3:63-4:4.

> "If the cache was not full, the system uses the setup table to retrieve the compressed data (34) and sends the compressed data to a compression

engine for decompressing (38). The system continues as if data was in the cache." Hovis, 4:40-44.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Hovis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Hovis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Hovis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Hovis, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Hovis, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The function of the setup table 14 is to provide a directory for the memory locations which are in the compressed storage 16. When an access to the memory 10 misses the cache 12, it generates an access to the setup table 14. The data from this access contains the location of the data within compressed storage 16. The address results in an access to the compressed storage 16 which in turn results in compressed data being accessed and processed by the compression engine 28, which performs compression and decompression on the data. The now uncompressed data is placed in the uncompressed cache 12 and transferred to the requesting element (for a fetch), or updated and maintained within the uncompressed cache 12 (for a store)."

Hovis, 3:3-15. *See also* Fig. 2, 3:23-38.

> "Also, the directory for the uncompressed cache must be updated when new locations are added or when old (the least recently used) locations are removed to compressed memory 16."

Hovis, 4:18-21.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Hovis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> "A compression technique is typically used With this memory architecture. For example, loss-less compression techniques used with the present invention could provide a two times or greater improvement in real memory capacity. This gain is dependent on both data patterns and the chosen compression algorithm. The present invention is not dependent on any particular compression algorithm; it can use any lossless compression algorithm in general."

Hovis, 2:23-30.

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Hovis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Hovis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A compression technique is typically used with this memory architecture. For example, loss-less compression techniques used with the present invention could provide a two times or greater improvement in real memory capacity. This gain is dependent on both data patterns and the chosen compression algorithm. The present invention is not dependent on any particular compression algorithm; it can use any lossless compression algorithm in general."

Hovis, 2:23-30.

> "The address results in an access to the compressed storage 16 Which in turn results in compressed data being accessed and processed by the compression engine 28, Which performs compression and decompression on the data."

Hovis, 3:8-12.

> "FIG. 4 shows a basic data flow structure of the memory 10 with a hardware assist compression engine 28. A memory control 24 interfaces With the memory 10, compression engine 28, and processor units 22. The purpose of a hardware based compression engine 28 is to provide the necessary bandwidth and latency required by memory entities that reside close to the processor data/instruction units (i.e. L1,L2,L3)."

Hovis, 3:41-46

> "The data is then uncompressed and sent back to the requesting element and uncompressed cache in the memory by the compression engine 28."

Hovis, 4:2-4.

> "If the cache is full, then the system must do a cast out and transfer the least recently used data element in the cache to the compressed storage (48), Which requires sending the data to the compression engine for compressing (46). The system then updates the data element in the cache (50) and continues as if the cache Was not full. If the cache Was not full, the system uses the setup table to retrieve the compressed data (34) and sends the compressed data to a compression engine for decompressing (38)."

Hovis, 4:34-44.

> "Aspects of the control of the hardware assisted memory compression can be implemented either by hardware, software (operating system, namely memory management), or a combination of both. The present invention is not dependent upon a particular hardware or software control scheme. The compression engine is preferably implemented in hardware in order to perform the compression and decompression in a timely fashion and With sufficient bandwidth."

Hovis, 5:4-11.

> "The implementation involves a hardware compression engine and the control can be via hardware, software, or a combination of both."

Hovis, 5:27-30.

11.1. a processor;	Hovis, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Hovis, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Hovis, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Hovis, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Hovis, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Hovis, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Hovis, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Hovis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Hovis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Hovis discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B9
Invalidity of U.S. Patent 8,090,936 based on Ingvar

G.B. Patent No. 2276257 to Ingvar ("Ingvar") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B9
Invalidity of U.S. Patent 8,090,936 based on Ingvar

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Ingvar, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

> "A method for enabling the transfer of stored firmware during start-up of a computer, comprising the steps of reading configuration information stored in a air configuration table in the first memory device (40), transferring selected software modules from the first memory device to a second memory device (130, 60, 50), and storing the selected software modules at selected address areas in the second memory device; in all accordance with the configuration information."

Ingvar, Abstract.

> "The memory device 40 may include firmware, such as startup software and BIOS program modules. According to the invention, certain software modules may be stored in a compressed form in the memory device 40. A compressed data packet which includes one or more software modules will take up less memory space than that taken up by a corresponding amount of data or information stored in an uncompressed state, as is well known to the person skilled in this art."

Ingvar at 10.

> "The resultant configuration data or information can be stored in a configuration table which will thus disclose those program modules which are required by the computer system in respect of the current

configuration of said system."

Ingvar at 11-12. *See also* Ingvar at 12, ¶¶ 1-2.

> "The area c may include the aforesaid configuration table containing information as to which software modules shall be used, and information as to which addresses and software modules respectively shall be work-stored."

Ingvar at 13.

> "In Step S150, the startup program can use the information in the configuration table to decide which of the program modules shall be used in the computer system and to obtain information as to where, i.e. at which addresses, respective software modules shall be placed in the working memory 130 for future use, until the computer unit is stopped and restarted."

Ingvar at 14-15.

> "According to this embodiment, when the configuration table is stored in a compressed state in the memory device 40, the reconfiguration command can provide access to the decompressed configuration table in the working memory 130. According to this latter embodiment of the invention, the user is able to initiate compression and storage of the modified table in the permanent memory device 40."

Ingvar at 17.

> "Steps S220 and S230 The selected software is compressed in Step S220 and is stored in Step S230 in a selected memory area in the first memory device 40, which will retain its data content in the absence of an applied voltage."

Ingvar at 18.

> "Step S250 According to one embodiment of the invention, there is inserted in Step S250 configuration information which may also be stored in the first memory device. The configuration information may have the form of a data table which discloses, among other things, the memory addresses at which the various software modules shall be stored when decompressed. The table may also include information as to which software modules shall be work-stored in the working memory and/or the addresses at which the modules shall be work-stored when carrying out

the first method according to the invention, as described above."

Ingvar at 18.

> "According to another embodiment of the invention, the software modules are stored in the permanent memory device 40 in an uncompressed state, together with the aforesaid configuration information and startup program."

Ingvar at 19.

1.2 initializing a central processing unit of said computer system;	Ingvar, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

> "According to one aspect on the present invention there is provided a method for use when starting up a computer system arrangement, the arrangement comprising at least one computer unit and at least one first memory device which includes at least one first data block with information stored as firmware, the method including the step of initiating the start of the computer unit and the step of transferring software modules from the first memory device to a second memory device, characterized by the steps of reading changeable configuration information stored in a configuration table in the first memory device; transferring selected modules among said software modules from the first memory device to the second memory device, and storing the selected software modules at selected address areas in the second memory device in accordance with the configuration information."

Ingvar at 10.

> "When starting-up the computer unit 20, the data processing device 30 reads information contained in a memory device 40."

Ingvar at 10.

> "Step S120 and S130 In Step S120, the data processing unit 30 included in the computer unit executes a first startup program. This startup program can be executed and when executed, the first startup program may include a routine which requires a check to be made as to which peripheral units are included in the hardware included in the computer system, as illustrated by Step S130 in Fig. 2."

Ingvar at 11.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Ingvar, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

> "A method for enabling the transfer of stored firmware during start-up of a computer, comprising the steps of reading configuration information stored in a air configuration table in the first memory device (40), transferring selected software modules from the first memory device to a second memory device (130, 60, 50), and storing the selected software modules at selected address areas in the second memory device; in all accordance with the configuration information."

Ingvar, Abstract.

> "The memory device 40 may include firmware, such as startup software and BIOS program modules. According to the invention, certain software modules may be stored in a compressed form in the memory device 40. A compressed data packet which includes one or more software modules will take up less memory space than that taken up by a corresponding amount of data or information stored in an uncompressed state, as is well known to the person skilled in this art."

Ingvar at 10.

> "Steps S220 and S230 The selected software is compressed in Step S220 and is stored in Step S230 in a selected memory area in the first memory device 40, which will retain its data content in the absence of an applied voltage."

Ingvar at 18.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Ingvar, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

> "Step 5140 In Step S140, the first startup program summons a decompression or decompaction program which unpacks or decompresses the firmware modules that are given in the configuration table and that are stored in a compressed state. According to another embodiment of the inventive method, the first decompression program decompresses all firmware modules stored in the memory device 40, wherein the choice of those modules that shall be used is made in accordance with the configuration table after said decompression."

Ingvar at 12.

> "Step S240 In Step S240, there is inserted a decompression program, which can also be stored in the first memory device 40."

Ingvar at 18.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Ingvar, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

> "Step 5140 In Step S140, the first startup program summons a decompression or decompaction program which unpacks or decompresses the firmware modules that are given in the configuration table and that are stored in a compressed state. According to another embodiment of the inventive method, the first decompression program decompresses all firmware modules stored in the memory device 40, wherein the choice of those modules that shall be used is made in accordance with the configuration table after said decompression."

Ingvar at 12.

> "Step S240 In Step S240, there is inserted a decompression program, which can also be stored in the first memory device 40."

Ingvar at 18.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Ingvar, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Ingvar, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Another object of the present invention is to-enable the selection of which software modules shall be placed in which memory positions with the aid of software."

Ingvar at 7.

> "According to one aspect on the present invention there is provided a method for use when starting up a computer system arrangement, the arrangement comprising at least one computer unit and at least one first memory device which includes at least one first data block with information stored as firmware, the method including the step of initiating the start of the computer unit and the step of transferring software modules from the first memory device to a second memory device, characterized by the steps of reading changeable configuration information stored in a configuration table in the first memory device; transferring selected modules among said software modules from the first memory device to the second memory device, and storing the selected software modules at selected address areas in the second memory device in accordance with the configuration information."

Ingvar at 10.

> "The invention also relates to a method of storing firmware in a computer unit so that software modules stored as firmware can be easily made movable to selected addresses in the working memory 130."

Ingvar at 18.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Ingvar, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Another object of the present invention is to-enable the selection of which software modules shall be placed in which memory positions with the aid of software."

Ingvar at 7.

> "According to one aspect on the present invention there is provided a method for use when starting up a computer system arrangement, the arrangement comprising at least one computer unit and at least one first memory device which includes at least one first data block with information stored as firmware, the method including the step of initiating the start of the computer unit and the step of transferring software modules from the first memory device to a second memory device, characterized by the steps of reading changeable configuration information stored in a configuration table in the first memory device; transferring selected modules among said software modules from the first memory device to the second memory device, and storing the selected software modules at selected address areas in the second memory device in accordance with the configuration information."

Ingvar at 10.

"The invention also relates to a method of storing firmware in a computer unit so that software modules stored as firmware can be easily made movable to selected addresses in the working memory 130." Ingvar at 18.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Ingvar, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Ingvar, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A further object of the present invention is to provide a method to enable program modules or data tables which are stored as firmware when starting-up the computer system to be repositioned such that the remaining unoccupied memory space can be readily utilized."

Ingvar at 7.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Ingvar, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The memory area c may also include the aforesaid decompression program, which can be used to decompress compressed data. The memory area d may be a memory area which contains compressed software modules."

Ingvar at 13.

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Ingvar, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Yet another object is to reduce the amount of hardware required by a computer system, for instance such hardware as address decoders, jumpers, permanent memory sockets, etc."

Ingvar at 8.

> "The memory area c may also include the aforesaid decompression program, which can be used to decompress compressed data. The memory area d may be a memory area which contains compressed software modules."

Ingvar at 13.

> "According to this embodiment, when the configuration table is stored in a compressed state in the memory device 40, the reconfiguration command can provide access to the decompressed configuration table in the working memory 130. According to this latter embodiment of the invention, the user is able to initiate compression and storage of the modified table in the permanent memory device 40."

Ingvar at 17.

11.1. a processor;	Ingvar, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Ingvar, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Ingvar, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Ingvar discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> "Permanent Memory A memory which will retain stored data even in the absence of power supply."

Ingvar at 7.

> "The memory means 40 may be a permanent memory. According to one embodiment of the invention, the permanent memory 40 is a FLASH-memory device. According to another embodiment of the invention, the permanent memory 40 is a conventional EEPROM."

Ingvar at 10.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Ingvar, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Ingvar, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Ingvar, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Ingvar, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Ingvar, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B9
Invalidity of U.S. Patent 8,090,936 based on Ingvar

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Ingvar, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Ingvar discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B10
Invalidity of U.S. Patent 8,090,936 based on Kikinis

PCT Application No. WO 94/19768 to Kikinis ("Kikinis") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B10
Invalidity of U.S. Patent 8,090,936 based on Kikinis

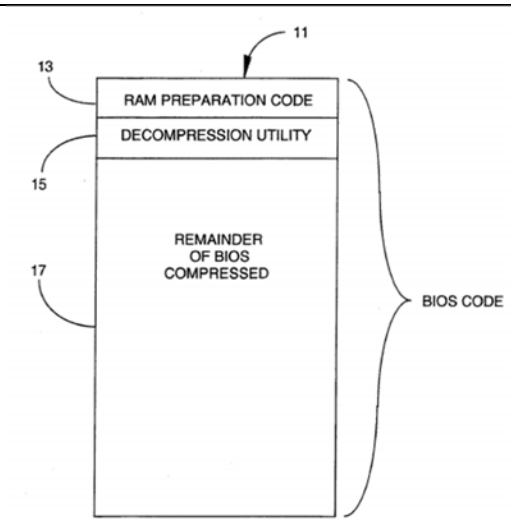
1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Kikinis, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

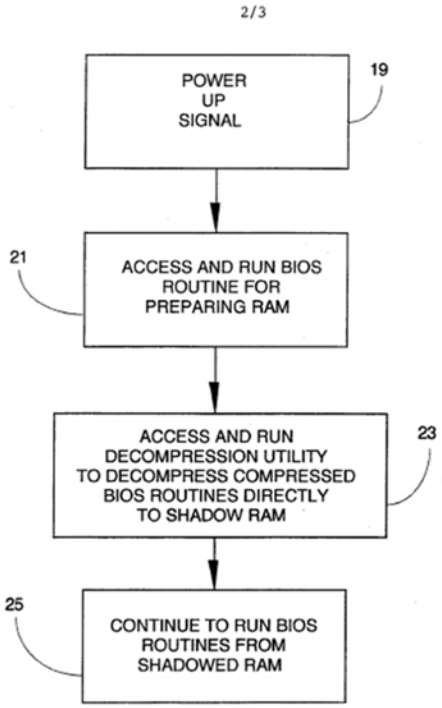
Kikinis discloses this claim limitation:

> A means of providing a BIOS routine from an EPROM to RAM in a general purpose computer, where the BIOS routine is greater in number of lines of code than the line capacity of the EPROM is provided. The BIOS routine is stored in EPROM in three portions (11). A first portion (13) is uncompressed, and loadable and operable by the CPU of the computer to initialize RAM. A second portion (17) is compressed by one of a number of schemes. A third portion (15) is a decompression utility loadable and operable by the computer CPU to decompress the compressed portion from EPROM and copy the resulting code to RAM, resulting in a BIOS in RAM (4).

Kikinis, Abstract



Kikinis, Fig. 1



Kikinis, Fig 2

As is well known in the art, a computer system, to be of any use, must comprise all the computer hardware, such as the CPU, memory devices, communication buses, and so forth. There must also be instruction sets

(programs/software) for the CPU to follow to accomplish tasks. The programmed information (application software) is typically stored on an internal or peripheral memory device to be accessed by the CPU as needed.

Kikinis, 1

A firmware device according to a preferred embodiment of the invention provides a BIOS routine for a general-purpose computer having a CPU microprocessor, comprising a programmable non-volatile memory device, and a BIOS routine stored on the programmable non-volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access memory in the general-purpose computer, and a decompression utility code operable by the CPU to load the compressed portion, decompress it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device

Kikinis, 2-3

In most BIOS systems for general purpose computers, the BIOS is stored in an EPROM device as described in the background section above. Upon power up the BIOS initializes the system, doing basic tasks like accessing and checking the operation of on-board random-access memory RAM, and typically, somewhere during the initialization, at least a part of the BIOS code is copied (the BIOS copies itself) into a portion of the on-board RAM.

Kikinis, 4

In typical general-purpose computers, as soon as the system receives power, the BIOS tests and initializes system RAM, then copies (shadows) itself from the EPROM to the RAM. The BIOS continues to run in RAM. The purpose of shadowing the BIOS in RAM is to give the CPU microprocessor much faster access to the BIOS code than it would have by accessing the EPROM every time a BIOS code sequence is needed in continuing operations.

Kikinis, 4

The present invention comprises a means of compressing at least a significant portion of the BIOS code, storing all of the BIOS code, including the compressed portion, in EPROM, and releasing the

compressed code on powerup, so all of the code is available for the computer to use. Also on powerup, the entire code is shadowed to RAM

Kikinis, 4-5

Fig. 1 is a diagrammatical representation of a compressed BIOS 11 according to the present invention. There are three different portions of the code. Portion 13 is code to perform all operations to initialize and test the system RAM, and make it ready for use, and is a familiar portion of conventional BIOS routines. This portion in some applications needs to perform such functions as initializing and testing a memory controller and cache controllers and cache memory. Portion 15 is a decompression utility. Portion 17 represents the balance of the BIOS code in compressed form. It will be apparent to those with skill in the art that there are a number of compression schemes and related decompression routines that might be used.

Kikins 5

Fig. 2 is a flow chart showing the operation of a computer from startup following a BIOS routine according to the present invention. From powerup signal 19, which is typically derived from the act of closing the power on switch, operation goes to initialization operation 21, during which system RAM is initialized. In operation 21, the system runs portion 13 of Fig. 1.

Next, decompression utility 15 (Fig. 1) is accessed and run in operation 23. The decompression utility processes the balance of the BIOS code (compressed), translates it into operable code, and shadows it to system RAM. Although such decompression utilities are available, the code pointing to the compressed portion of the BIOS, and that which causes the decompressed code to be shadowed to RAM is not a part of a conventional decompression routine. These commands are added to the BIOS of the invention.

After the BIOS is shadowed operation continues (25) from the BIOS in system RAM. All remaining BIOS processes, including testing and initializing the remainder of the computer subsystems are accomplished in this operating portion.

Kikinis, 5-6

One means by which the BIOS code may be compressed is based on the fact that BIOS routines, as is common in most other coded instruction sets, make use of frequently repeated code sequences. EPROMs used

> for BIOS are typically byte-wide devices, that is, the device can store "words" of 8 bits. A sixteen bit word requires, then, two lines of BIOS code.

Kikinis, 6

> Fig. 3 is a diagrammatical representation of the token decompression scheme described above. After the BIOS according to the embodiment of the invention has initialized and tested the RAM, the decompression utility is booted, and begins to read the compressed portion of the BIOS at Start 27. At 29 the decompression utility loads the first/next byte from the compressed portion of
> the EPROM BIOS. If this byte is hex FF (31), it is recognized as a token, and control goes to 33,. where the system reads the byte following the token flag. This byte is always a pointer to a code sequence.

Kikinis, 7

See also Kikinis Fig. 3.

1.2 initializing a central processing unit of said computer system;	Kikinis, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this claim limitation:

> A means of providing a BIOS routine from an EPROM to RAM in a general purpose computer, where the BIOS routine is greater in number of lines of code than the line capacity of the EPROM is provided. The BIOS routine is stored in EPROM in three portions (11). A first portion (13) is uncompressed, and loadable and operable by the CPU of the computer to initialize RAM. A second portion (17) is compressed by one of a number of schemes. A third portion (15) is a decompression utility loadable and operable by the computer CPU to decompress the compressed portion from EPROM and copy the resulting code to RAM, resulting in a BIOS in RAM (4).

Kikinis, Abstract

> As is well known in the art, a computer system, to be of any use, must comprise all the computer hardware, such as the CPU, memory devices, communication buses, and so forth. There must also be instruction sets (programs/software) for the CPU to follow to accomplish tasks. The programmed information (application software) is typically stored on an internal or peripheral memory device to be accessed by the CPU as needed.

Kikinis, 1

> A firmware device according to a preferred embodiment of the invention provides a BIOS routine for a general-purpose computer having a CPU microprocessor, comprising a programmable non-volatile memory device, and a BIOS routine stored on the programmable non-volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access memory in the general-purpose computer, and a decompression utility code operable by the CPU to load the compressed portion, decompress

it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device

Kikinis, 2-3

In most BIOS systems for general purpose computers, the BIOS is stored in an EPROM device as described in the background section above. Upon power up the BIOS initializes the system, doing basic tasks like accessing and checking the operation of on-board random-access memory RAM, and typically, somewhere during the initialization, at least a part of the BIOS code is copied (the BIOS copies itself) into a portion of the on-board RAM.

Kikinis, 4

In typical general-purpose computers, as soon as the system receives power, the BIOS tests and initializes system RAM, then copies (shadows) itself from the EPROM to the RAM. The BIOS continues to run in RAM. The purpose of shadowing the BIOS in RAM is to give the CPU microprocessor much faster access to the BIOS code than it would have by accessing the EPROM every time a BIOS code sequence is needed in continuing operations.

Kikinis, 4

Fig. 1 is a diagrammatical representation of a compressed BIOS 11 according to the present invention. There are three different portions of the code. Portion 13 is code to perform all operations to initialize and test the system RAM, and make it ready for use, and is a familiar portion of conventional BIOS routines. This portion in some applications needs to perform such functions as initializing and testing a memory controller and cache controllers and cache memory. Portion 15 is a decompression utility. Portion 17 represents the balance of the BIOS code in compressed form. It will be apparent to those with skill in the art that there are a number of compression schemes and related decompression routines that might be used.

Kikins 5

Fig. 2 is a flow chart showing the operation of a computer from startup following a BIOS routine according to the present invention. From powerup signal 19, which is typically derived from the act of closing the power on switch, operation goes to initialization operation 21, during

> which system RAM is initialized. In operation 21, the system runs portion 13 of Fig. 1.
>
> Next, decompression utility 15 (Fig. 1) is accessed and run in operation 23. The decompression utility processes the balance of the BIOS code (compressed), translates it into operable code, and shadows it to system RAM. Although such decompression utilities are available, the code pointing to the compressed portion of the BIOS, and that which causes the decompressed code to be shadowed to RAM is not a part of a conventional decompression routine. These commands are added to the BIOS of the invention.
>
> After the BIOS is shadowed operation continues (25) from the BIOS in system RAM. All remaining BIOS processes, including testing and initializing the remainder of the computer subsystems are accomplished in this operating portion.

Kikinis, 5-6

See also Kikinis Fig. 1-3.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Kikinis, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this claim limitation:

> A means of providing a BIOS routine from an EPROM to RAM in a general purpose computer, where the BIOS routine is greater in number of lines of code than the line capacity of the EPROM is provided. The BIOS routine is stored in EPROM in three portions (11). A first portion (13) is uncompressed, and loadable and operable by the CPU of the computer to initialize RAM. A second portion (17) is compressed by one of a number of schemes. A third portion (15) is a decompression utility loadable and operable by the computer CPU to decompress the compressed portion from EPROM and copy the resulting code to RAM, resulting in a BIOS in RAM (4).

Kikinis, Abstract

> As is well known in the art, a computer system, to be of any use, must comprise all the computer hardware, such as the CPU, memory devices, communication buses, and so forth. There must also be instruction sets (programs/software) for the CPU to follow to accomplish tasks. The programmed information (application software) is typically stored on an internal or peripheral memory device to be accessed by the CPU as needed.

Kikinis, 1

> A firmware device according to a preferred embodiment of the invention provides a BIOS routine for a general-purpose computer having a CPU microprocessor, comprising a programmable non-volatile memory device, and a BIOS routine stored on the programmable non-volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access

> memory in the general-purpose computer, and a decompression utility code operable by the CPU to load the compressed portion, decompress it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device

Kikinis, 2-3

> In most BIOS systems for general purpose computers, the BIOS is stored in an EPROM device as described in the background section above. Upon power up the BIOS initializes the system, doing basic tasks like accessing and checking the operation of on-board random-access memory RAM, and typically, somewhere during the initialization, at least a part of the BIOS code is copied (the BIOS copies itself) into a portion of the on-board RAM.

Kikinis, 4

> In typical general-purpose computers, as soon as the system receives power, the BIOS tests and initializes system RAM, then copies (shadows) itself from the EPROM to the RAM. The BIOS continues to run in RAM. The purpose of shadowing the BIOS in RAM is to give the CPU microprocessor much faster access to the BIOS code than it would have by accessing the EPROM every time a BIOS code sequence is needed in continuing operations.

Kikinis, 4

> The present invention comprises a means of compressing at least a significant portion of the BIOS code, storing all of the BIOS code, including the compressed portion, in EPROM, and releasing the compressed code on powerup, so all of the code is available for the computer to use. Also on powerup, the entire code is shadowed to RAM

Kikinis, 4-5

> Fig. 1 is a diagrammatical representation of a compressed BIOS 11 according to the present invention. There are three different portions of the code. Portion 13 is code to perform all operations to initialize and test the system RAM, and make it ready for use, and is a familiar portion of conventional BIOS routines. This portion in some applications needs to perform such functions as initializing and testing a memory controller and cache controllers and cache memory. Portion 15 is a decompression utility. Portion 17 represents the balance of the BIOS code in compressed form. It will be apparent to those with skill in

the art that there are a number of compression schemes and related decompression routines that might be used.

Kikins 5

Fig. 2 is a flow chart showing the operation of a computer from startup following a BIOS routine according to the present invention. From powerup signal 19, which is typically derived from the act of closing the power on switch, operation goes to initialization operation 21, during which system RAM is initialized. In operation 21, the system runs portion 13 of Fig. 1.

Next, decompression utility 15 (Fig. 1) is accessed and run in operation 23. The decompression utility processes the balance of the BIOS code (compressed), translates it into operable code, and shadows it to system RAM. Although such decompression utilities are available, the code pointing to the compressed portion of the BIOS, and that which causes the decompressed code to be shadowed to RAM is not a part of a conventional decompression routine. These commands are added to the BIOS of the invention.

After the BIOS is shadowed operation continues (25) from the BIOS in system RAM. All remaining BIOS processes, including testing and initializing the remainder of the computer subsystems are accomplished in this operating portion.

Kikinis, 5-6

One means by which the BIOS code may be compressed is based on the fact that BIOS routines, as is common in most other coded instruction sets, make use of frequently repeated code sequences. EPROMs used for BIOS are typically byte-wide devices, that is, the device can store "words" of 8 bits. A sixteen bit word requires, then, two lines of BIOS code.

Kikinis, 6

See also Kikinis Fig. 1-3.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Kikinis, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this claim limitation:

> A means of providing a BIOS routine from an EPROM to RAM in a general purpose computer, where the BIOS routine is greater in number of lines of code than the line capacity of the EPROM is provided. The BIOS routine is stored in EPROM in three portions (11). A first portion (13) is uncompressed, and loadable and operable by the CPU of the computer to initialize RAM. A second portion (17) is compressed by one of a number of schemes. A third portion (15) is a decompression utility loadable and operable by the computer CPU to decompress the compressed portion from EPROM and copy the resulting code to RAM, resulting in a BIOS in RAM (4).

Kikinis, Abstract

> As is well known in the art, a computer system, to be of any use, must comprise all the computer hardware, such as the CPU, memory devices, communication buses, and so forth. There must also be instruction sets (programs/software) for the CPU to follow to accomplish tasks. The programmed information (application software) is typically stored on an internal or peripheral memory device to be accessed by the CPU as needed.

Kikinis, 1

> A firmware device according to a preferred embodiment of the invention provides a BIOS routine for a general-purpose computer having a CPU microprocessor, comprising a programmable non-volatile memory device, and a BIOS routine stored on the programmable non-volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access memory in the general-purpose computer, and a decompression utility

code operable by the CPU to load the compressed portion, decompress it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device

Kikinis, 2-3

In most BIOS systems for general purpose computers, the BIOS is stored in an EPROM device as described in the background section above. Upon power up the BIOS initializes the system, doing basic tasks like accessing and checking the operation of on-board random-access memory RAM, and typically, somewhere during the initialization, at least a part of the BIOS code is copied (the BIOS copies itself) into a portion of the on-board RAM.

Kikinis, 4

In typical general-purpose computers, as soon as the system receives power, the BIOS tests and initializes system RAM, then copies (shadows) itself from the EPROM to the RAM. The BIOS continues to run in RAM. The purpose of shadowing the BIOS in RAM is to give the CPU microprocessor much faster access to the BIOS code than it would have by accessing the EPROM every time a BIOS code sequence is needed in continuing operations.

Kikinis, 4

The present invention comprises a means of compressing at least a significant portion of the BIOS code, storing all of the BIOS code, including the compressed portion, in EPROM, and releasing the compressed code on powerup, so all of the code is available for the computer to use. Also on powerup, the entire code is shadowed to RAM

Kikinis, 4-5

Fig. 2 is a flow chart showing the operation of a computer from startup following a BIOS routine according to the present invention. From powerup signal 19, which is typically derived from the act of closing the power on switch, operation goes to initialization operation 21, during which system RAM is initialized. In operation 21, the system runs portion 13 of Fig. 1.

Next, decompression utility 15 (Fig. 1) is accessed and run in operation 23. The decompression utility processes the balance of the BIOS code (compressed), translates it into operable code, and shadows it to system

> RAM. Although such decompression utilities are available, the code pointing to the compressed portion of the BIOS, and that which causes the decompressed code to be shadowed to RAM is not a part of a conventional decompression routine. These commands are added to the BIOS of the invention.
>
> After the BIOS is shadowed operation continues (25) from the BIOS in system RAM. All remaining BIOS processes, including testing and initializing the remainder of the computer subsystems are accomplished in this operating portion.

Kikinis, 5-6

> Fig. 3 is a diagrammatical representation of the token decompression scheme described above. After the BIOS according to the embodiment of the invention has initialized and tested the RAM, the decompression utility is booted, and begins to read the compressed portion of the BIOS at Start 27. At 29 the decompression utility loads the first/next byte from the compressed portion of
> the EPROM BIOS. If this byte is hex FF (31), it is recognized as a token, and control goes to 33,. where the system reads the byte following the token flag. This byte is always a pointer to a code sequence.

Kikinis, 7

See also Kikinis Fig. 1-3.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Kikinis, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this claim limitation:

> A means of providing a BIOS routine from an EPROM to RAM in a general purpose computer, where the BIOS routine is greater in number of lines of code than the line capacity of the EPROM is provided. The BIOS routine is stored in EPROM in three portions (11). A first portion (13) is uncompressed, and loadable and operable by the CPU of the computer to initialize RAM. A second portion (17) is compressed by one of a number of schemes. A third portion (15) is a decompression utility loadable and operable by the computer CPU to decompress the compressed portion from EPROM and copy the resulting code to RAM, resulting in a BIOS in RAM (4).

Kikinis, Abstract

> As is well known in the art, a computer system, to be of any use, must comprise all the computer hardware, such as the CPU, memory devices, communication buses, and so forth. There must also be instruction sets (programs/software) for the CPU to follow to accomplish tasks. The programmed information (application software) is typically stored on an internal or peripheral memory device to be accessed by the CPU as needed.

Kikinis, 1

> A firmware device according to a preferred embodiment of the invention provides a BIOS routine for a general-purpose computer having a CPU microprocessor, comprising a programmable non-volatile memory device, and a BIOS routine stored on the programmable non-

> volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access memory in the general-purpose computer, and a decompression utility code operable by the CPU to load the compressed portion, decompress it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device

Kikinis, 2-3

> In most BIOS systems for general purpose computers, the BIOS is stored in an EPROM device as described in the background section above. Upon power up the BIOS initializes the system, doing basic tasks like accessing and checking the operation of on-board random-access memory RAM, and typically, somewhere during the initialization, at least a part of the BIOS code is copied (the BIOS copies itself) into a portion of the on-board RAM.

Kikinis, 4

> In typical general-purpose computers, as soon as the system receives power, the BIOS tests and initializes system RAM, then copies (shadows) itself from the EPROM to the RAM. The BIOS continues to run in RAM. The purpose of shadowing the BIOS in RAM is to give the CPU microprocessor much faster access to the BIOS code than it would have by accessing the EPROM every time a BIOS code sequence is needed in continuing operations.

Kikinis, 4

> The present invention comprises a means of compressing at least a significant portion of the BIOS code, storing all of the BIOS code, including the compressed portion, in EPROM, and releasing the compressed code on powerup, so all of the code is available for the computer to use. Also on powerup, the entire code is shadowed to RAM

Kikinis, 4-5

> Fig. 2 is a flow chart showing the operation of a computer from startup following a BIOS routine according to the present invention. From powerup signal 19, which is typically derived from the act of closing the power on switch, operation goes to initialization operation 21, during which system RAM is initialized. In operation 21, the system runs portion 13 of Fig. 1.

> Next, decompression utility 15 (Fig. 1) is accessed and run in operation 23. The decompression utility processes the balance of the BIOS code (compressed), translates it into operable code, and shadows it to system RAM. Although such decompression utilities are available, the code pointing to the compressed portion of the BIOS, and that which causes the decompressed code to be shadowed to RAM is not a part of a conventional decompression routine. These commands are added to the BIOS of the invention.
>
> After the BIOS is shadowed operation continues (25) from the BIOS in system RAM. All remaining BIOS processes, including testing and initializing the remainder of the computer subsystems are accomplished in this operating portion.

Kikinis, 5-6

> Fig. 3 is a diagrammatical representation of the token decompression scheme described above. After the BIOS according to the embodiment of the invention has initialized and tested the RAM, the decompression utility is booted, and begins to read the compressed portion of the BIOS at Start 27. At 29 the decompression utility loads the first/next byte from the compressed portion of
> the EPROM BIOS. If this byte is hex FF (31), it is recognized as a token, and control goes to 33,. where the system reads the byte following the token flag. This byte is always a pointer to a code sequence.

Kikinis, 7

See also Kikinis Fig. 1-3.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Kikinis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Kikinis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Kikinis, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Kikinis, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> Fig. 1 is a diagrammatical representation of a compressed BIOS 11 according to the present invention. There are three different portions of the code. Portion 13 is code to perform all operations to initialize and test the system RAM, and make it ready for use, and is a familiar portion of conventional BIOS routines. This portion in some applications needs to perform such functions as initializing and testing a memory controller and cache controllers and cache memory. Portion 15 is a decompression utility. Portion 17 represents the balance of the BIOS code in compressed form. It will be apparent to those with skill in the art that there are a number of compression schemes and related decompression routines that might be used.

Kikinis, 5

6. The method of claim 1, further comprising updating the list of boot data.	Kikinis, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Kikinis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> In a particular compression scheme compression is accomplished by a substituting a two line token for a longer sequence, where the longer sequence is a sequence often repeated in the BIOS. The value of the first line is a flag to the decompression routine that the next line is to be used to correlate to the longer sequence and copy that longer sequence in decompression.

Kikinis, 3

> It is also true that there are a truly large number of compression schemes that might be employed to compress a portion of the BIOS. The invention should not be limited by the specific code relationship determined to compress the BIOS code.

Kikinis, 8

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Kikinis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> In a particular compression scheme compression is accomplished by a substituting a two line token for a longer sequence, where the longer sequence is a sequence often repeated in the BIOS. The value of the first line is a flag to the decompression routine that the next line is to be used to correlate to the longer sequence and copy that longer sequence in decompression.

Kikinis, 3

> It is also true that there are a truly large number of compression schemes that might be employed to compress a portion of the BIOS. The invention should not be limited by the specific code relationship determined to compress the BIOS code.

Kikinis, 8

11.1. a processor;	Kikinis, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Kikinis, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Kikinis, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Kikinis discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> Most BIOS implementations in general purpose computers are implemented in single chip erasable programmable non-volatile (EPROM) memory devices resident on a "motherboard" in the computer. There are other suitable non-volatile memory devices, however, which have been or may be used, including but not limited to EEPROM devices, "Flash Card" memories known in the art, masked ROM devices, CMOS RAM with a battery backup, and magnetic bubble memory.

Kikinis, 2

> A firmware device according to a preferred embodiment of the invention provides a BIOS routine for α general-purpose computer having a CPU microprocessor, comprising a programmable non-volatiJe memory device, and a BIOS routine stored on the programmable non-volatile memory device. The BIOS routine has a compressed portion, an uncompressed portion operable by the CPU to initialize random access

> memory in the general-purpose computer, and a decompression utility code operable by the CPU to load the compressed portion, decompress it, and copy the decompressed code to the random access memory. In a preferred embodiment the programmable memory device is an EPROM device.

Kikinis, 2-3

> For example, there are many non-volatile memories in which a compressed BIOS may be stored, retrieved, and decompressed, and several of these have been listed above. The fact of compressing the routines in the BIOS, including a loadable decompression routine, extends the capacity of any such finite non-volatile memory devices and hence the size of a BIOS routine that may be stored thereon.

Kikinis, 7-8

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Kikinis, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Kikinis, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Kikinis, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Kikinis, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Kikinis, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Kikinis, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Kikinis discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B11
Invalidity of U.S. Patent 8,090,936 based on Krocker

U.S. Patent No. 6,073,232 to Krocker ("Krocker") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B11
Invalidity of U.S. Patent 8,090,936 based on Krocker

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Krocker, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation:	

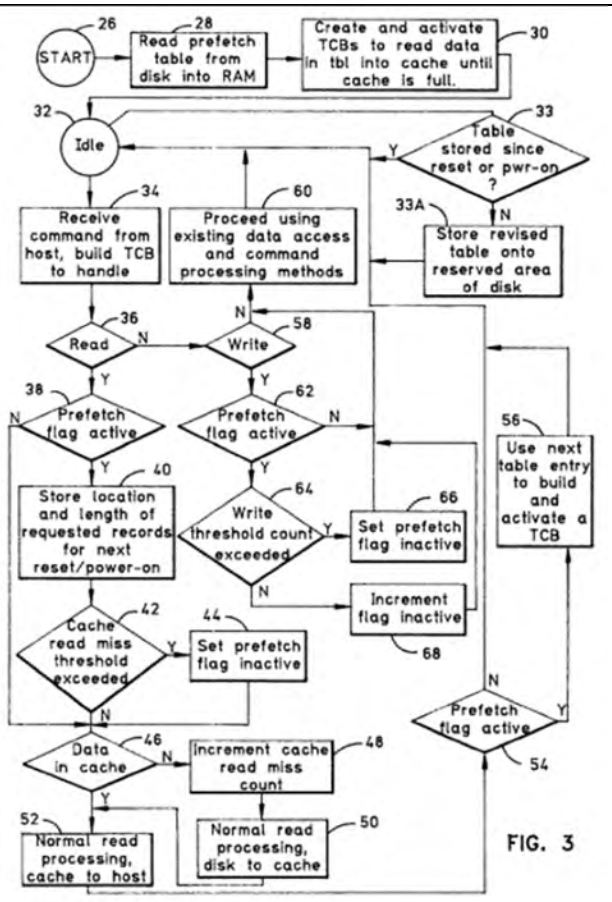


FIG. 3

Kroeker, Fig. 3.

> "A method for increasing boot speed of a host computer with associated hard disk drive generates a prefetch table that contains pointers to disk locations and lengths of the records of an application program requested by the host computer during an initial power-on/reset."

Kroeker, Abstract.

> "The present invention relates generally to peripheral storage apparatus for computers, and more particularly to shortening the load time of computer programs from a hard disk drive to a host computer."

Kroeker, 1:9-12.

> "In accordance with the present invention, steps executed by the digital processing apparatus include, after an initial power-up or reset of the hard disk drive and a host computer associated with the drive, receiving an initial read command from the host computer for transferring to the host

> computer a plurality of data records of a program stored on the disk. A prefetch table is then generated, with the table representing a disk location and length of each data record requested by the initial read command."

Kroeker, 2:20-37.

> "When the command is a read command, code means generate a prefetch table representative of at least the disk location of the records requested by the read command for transfer of the records from the disk to the cache for a subsequent power-on or reset of the host computer. Moreover, code means are provided for determining whether the records have been stored in the cache in response to a previous power-on or reset of the host computer, and the records are communicated to the host computer in response."

Kroeker, 3:21-29.

> "As discussed further below, the prefetch table contains a listing of the disk locations and lengths of data records that were requested by the host computer 14 in the immediately previous power-on/reset. Additionally, a copy of a prefetch flag, if enabled by the user, is created and set active at block 28."

Kroeker, 3:3-9. *See also* Kroeker, 3:9-16.

1.2 initializing a central processing unit of said computer system;	Krocker, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

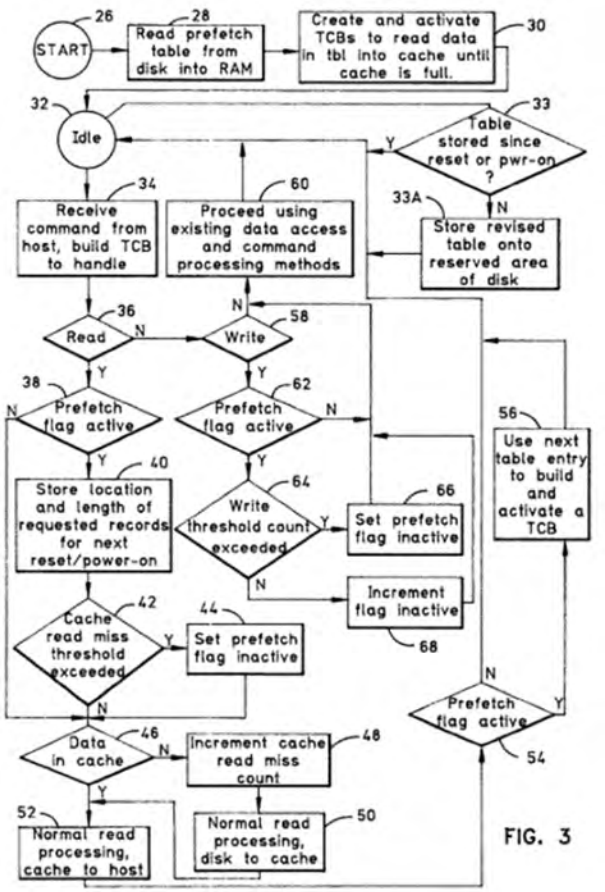


FIG. 3

Kroeker, Fig. 3.

> "During the next power-on/reset, before the host computer is ready for data but after the disk drive has completed its reset routine, using the prefetch table the disk drive accesses the previously requested data and copies it onto the cache of the disk drive, from where it is transferred to

the host computer when the host computer requests it."

Kroeker, Abstract.

> "Then, after a subsequent power-on or reset of the hard disk drive, and during a second power-on or reset of the host computer, the prefetch table is accessed to read into the data cache the data records."

Kroeker, 2:37-40.

> "As disclosed in detail below, these code means are for enhanced loading of the program from the hard disk drive to the host computer during power-on or reset of the host computer. In accordance with the present invention, code means receive a command from the host computer during a power-up or reset of the host computer."

Kroeker, 3:15-20.

> "Commencing at start state 26, the process moves to block 28, wherein a prefetch table is read from a reserved area of the disks 16 into the RAM cache 18."

Kroeker, 5:1-3.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Krocker, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

> "During the next power-on/reset, before the host computer is ready for data but after the disk drive has completed its reset routine, using the prefetch table the disk drive accesses the previously requested data and copies it onto the cache of the disk drive, from where it is transferred to the host computer when the host computer requests it."

Kroeker, Abstract.

> "This invention is realized in a critical machine component that causes a digital processing apparatus to adaptively store a computer program on the cache of the hard disk drive and communicate the program to the host computer."

Kroeker, 2:23-26.

> "Then, after a subsequent power-on or reset of the hard disk drive, and during a second power-on or reset of the host computer, the prefetch table is accessed to read into the data cache the data records."

Kroeker, 2:37-40.

> "In still another aspect, a computer hard disk drive includes at least one data storage disk and a data storage cache. Furthermore, the hard disk drive includes means for recording onto the cache, immediately after a hardware reset of the hard disk drive, data on the disk that has been requested by a host computer during a first hardware reset of the host computer."

Kroeker, 3:30-36.

> "From block 44, or from decision diamonds 38 or 42 when the decisions there are negative, the process moves to decision diamond 46 to determine whether the requested data exists in cache."

Kroeker, 5:65-6:1.

> "From block 50, or from decision diamond 46 if it was determined that the requested data exists in cache, the process moves to block 52 to transfer the record from cache 18 to the host computer 14."

Kroeker, 5:65-6:1.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Krocker, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

> "In response to a subsequent read command from the host computer, it is determined whether records requested by the subsequent read command are stored in the data cache. If they are, the records are communicated from the cache to the host computer; otherwise, the records are communicated from the disk to the host computer."

Kroeker, 2:40-46.

> "Preferably, the accessing and determining steps are repeated for each power-on or reset of the host computer."

Kroeker, 2:47-48.

> "Additionally, the disk drive includes means for communicating the data from the cache to the host computer during a second hardware reset of the host computer."

Kroeker, 3:36-38.

> "If it is active, the logic, at block 56, uses the next entry in the prefetch table to build a task control block (TCB) to fetch data into the same segment of the cache 18 that the just-transferred record had occupied prior to being communicated to the host computer 14. In accordance with the present invention, the TCB in block 50 is activated as though a command otherwise was received across the device/file interface. In other words, when the host computer 14 is a PC, the TCB in block 50 is activated as though a command otherwise was received across the SCSI (or IDE)--disk drive interface. In this way, the relatively small amount of cache storage space can be optimally used during the adaptive caching process until all records designated in the prefetch table have been loaded

into cache and then transferred to the host computer 14." Kroeker, 6:16-31.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Krocker, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Krocker, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Krocker, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A method for increasing boot speed of a host computer with associated hard disk drive generates a prefetch table that contains pointers to disk locations and lengths of the records of an application program requested by the host computer during an initial power-on/reset."

Kroeker, Abstract.

> "As recognized by the present invention, however, it is possible to provide, without operating system intervention, a method for adaptively preparing a disk drive to effect rapid application program loading to a host computer."

Kroeker, 1:55-58.

> "And, the hard disk drive 12 can be any hard disk drive suitable for computer applications, provided that the hard disk drive 12 includes at least one, and typically a plurality of, data storage disks 16 and an on-board, solid state, random access memory (RAM) data cache 18."

Kroeker, 4:5:10.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Krocker, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A method for increasing boot speed of a host computer with associated hard disk drive generates a prefetch table that contains pointers to disk locations and lengths of the records of an application program requested by the host computer during an initial power-on/reset."

Kroeker, Abstract.

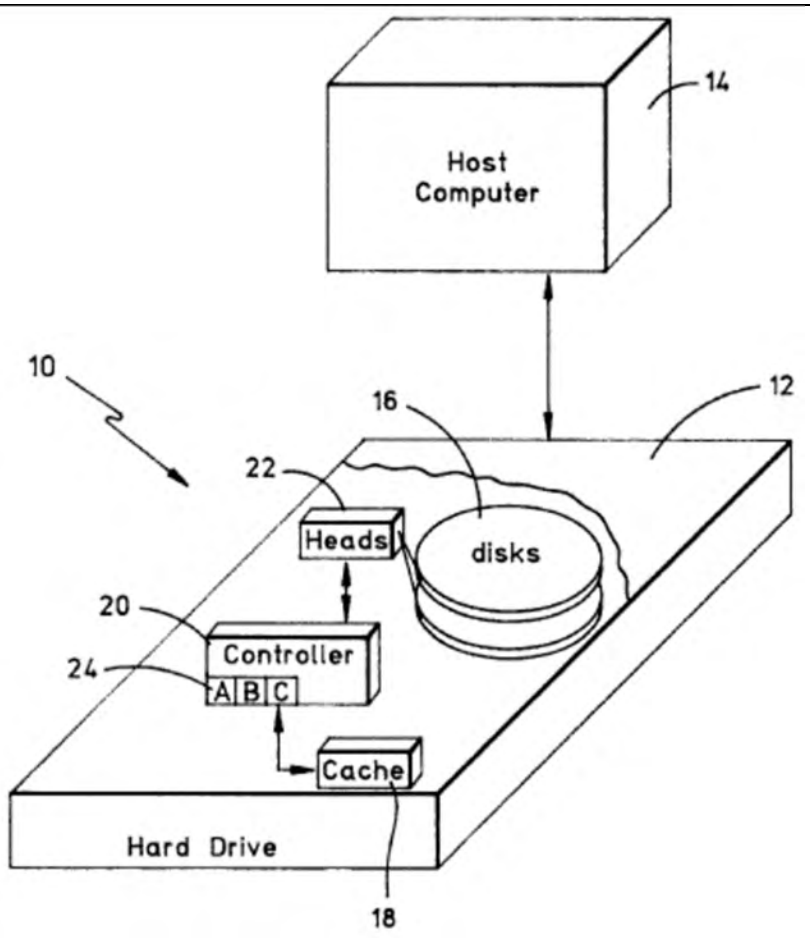
> "As recognized by the present invention, however, it is possible to provide, without operating system intervention, a method for adaptively preparing a disk drive to effect rapid application program loading to a host computer."

Kroeker, 1:55-58.

> "And, the hard disk drive 12 can be any hard disk drive suitable for computer applications, provided that the hard disk drive 12 includes at least one, and typically a plurality of, data storage disks 16 and an on-board, solid state, random access memory (RAM) data cache 18."

Kroeker, 4:5:10.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Krocker, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***Look for additional references to controller***	



Kroeker, Fig. 1.

> "Indeed, the host computer 14 can be an embedded controller that is part of a music synthesizer, or part of an industrial instrument."

Kroeker, 4:3-6.

> "As shown in FIG. 1, the hard disk drive 12 also includes an onboard controller 20. In accordance with principles well-known in the art, the onboard controller 20 is a digital processor Which, among other things, controls read heads 22 in the disk drive 12 for effecting data transfer to and from the disks 16."

Kroeker, 4:11-16.

> "Additionally, as intended by the present invention the onboard controller 20 includes an adaptive cache module 24. Per the present invention, the adaptive cache module 24 is executed by the onboard

> controller 20 as a series of computer-executable instructions. These instructions are embodied as microcode in a memory, e.g., read-only memory (ROM) of the onboard controller 20. Such a ROM is indicated by reference numeral 21 in FIG. 2."

Kroeker, 4:17-24.

> "Manifestly, the invention may be practiced in its essential embodiment by a machine component, embodied by the ROM 21,that renders the computer program code elements in a form that instructs a digital processing apparatus (e.g., the onboard controller 20) to perform a sequence of function steps corresponding to those shown in the Figures. The machine component is shown in FIGS. 1 and 2 as a combination of program code elements A—C in computer readable form that are embodied in a computer usable data medium (the ROM 21) of the onboard controller 20."

Kroeker, 4:43-53, Fig. 3.

6. The method of claim 1, further comprising updating the list of boot data.	Krocker, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Krocker discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The prefetch table is updated to reflect disk location changes for the various records, or to reflect new records that were requested by the host computer but not found in cache during the previous power-on/reset."

Kroeker, Abstract.

> "In accordance with the present invention, steps executed by the digital processing apparatus include, after an initial power-up or reset of the hard disk drive and a host computer associated with the drive, receiving an initial read command from the host computer for transferring to the host computer a plurality of data records of a program stored on the disk. A prefetch table is then generated, with the table representing a disk location and length of each data record requested by the initial read command."

Kroeker, 2:20-37.

> "Additionally, the steps further include updating the data prefetch table, communicating the records from the disk to the host computer when the read counter exceeds a predetermined threshold, and setting a prefetch flag to inactive when the read counter exceeds the predetermined threshold."

Kroeker, 2:59-64.

> "When the command is a read command, code means generate a prefetch table representative of at least the disk location of the records requested by the read command for transfer of the records from the disk to the cache for a subsequent power-on or reset of the host computer. Moreover, code means are provided for determining whether the records have been stored

> in the cache in response to a previous power-on or reset of the host computer, and the records are communicated to the host computer in response."

Kroeker, 3:21-29.

> "As discussed further below, the prefetch table contains a listing of the disk locations and lengths of data records that were requested by the host computer 14 in the immediately previous power-on/reset. Additionally, a copy of a prefetch flag, if enabled by the user, is created and set active at block 28."

Kroeker, 3:3-9. *See also* Kroeker, 3:9-16.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Krocker, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Krocker, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Krocker, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Krocker, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Krocker, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Krocker, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Krocker, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Krocker, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Krocker, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

Appendix B11
Invalidity of U.S. Patent 8,090,936 based on Krocker

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Krocker, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B11
Invalidity of U.S. Patent 8,090,936 based on Krocker

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Krocker, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Krocker discloses this limitation: *See* Claims 1, 9, and 11 above.	

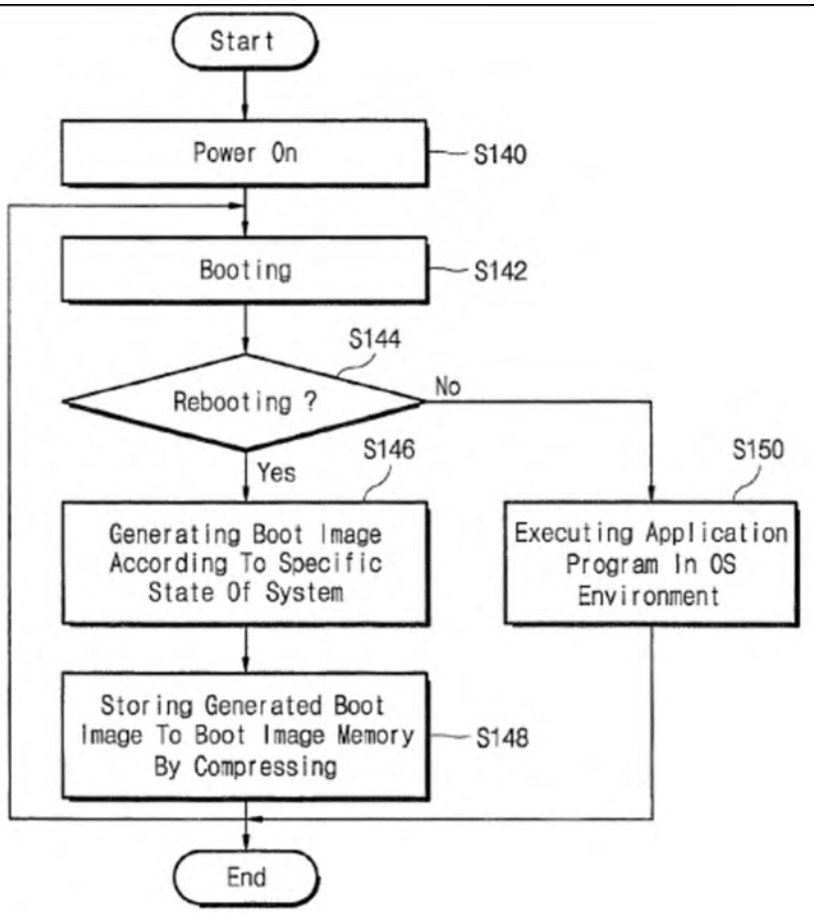
Appendix B12
Invalidity of U.S. Patent 8,090,936 based on Lee

U.S. Patent Application Publication No. 2001/0039612 Lee ("Lee") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B12
Invalidity of U.S. Patent 8,090,936 based on Lee

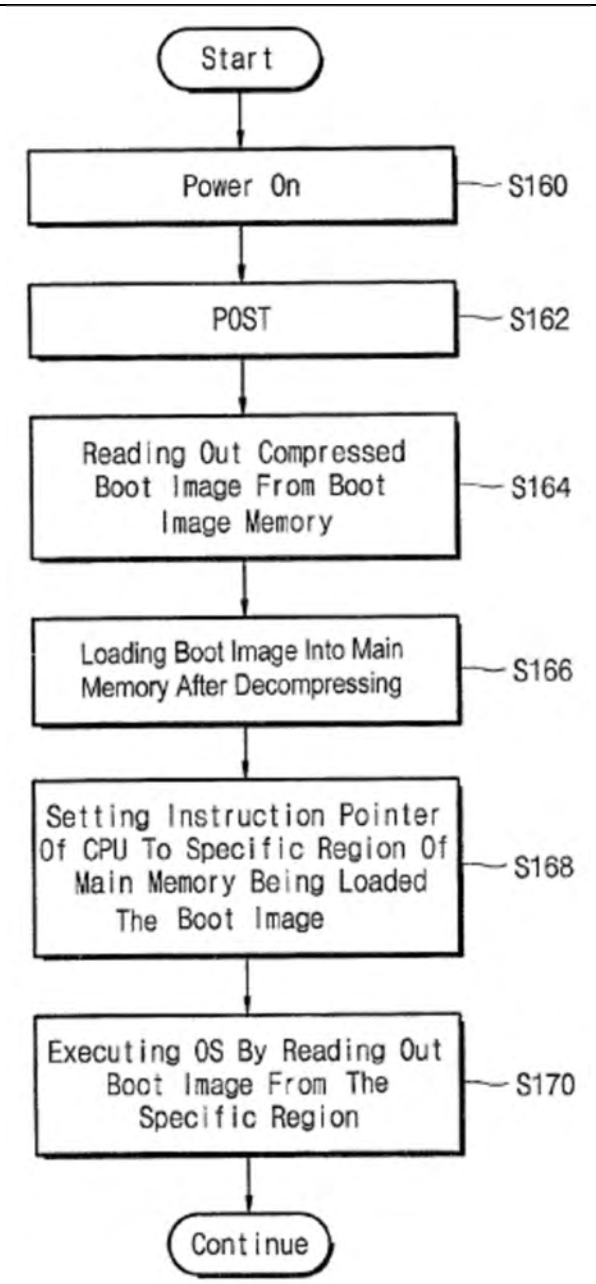
1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Lee, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation:	



Lee, Fig.2.

Lee **Claim 1.1**

"maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"



Lee, Fig. 3.

> "The ROM-based code attempts to establish communication with a so-called "boot device". A boot device holds information that is necessary to boot the system. In attempting to establish communication with a boot device, the ROM-based code operates according to a so-called "boot order.""

Lee, ¶ 0012.

> "More particularly, the ROM-based code attempts to retrieve a so-called "master boot record" from a particular sector of the diskette. If the communication attempt is successful, the ROM-based code uses that device as the boot device. If not, the ROM-based code proceeds to attempt communication With a device of the next boot order, e.g., local media."

Lee, ¶ 0012.

> "In order to attain the above objects, according to an aspect of the present invention, there is provided a computer system having a central processing unit; a main and/or auxiliary power supply for supplying main and/or auxiliary power of the computer system; a boot image storing device for storing a boot image of the computer system; a main memory for storing the boot image from the boot image storing device by receiving the auxiliary power when the main power is shut off; and a composition memory for setting an instruction pointer of the central processing unit to a specific region of the main memory storing the boot image; wherein the central processing unit loads the boot image from the specific region of the main memory in response to the instruction pointer, thereby an operating system program can perform control functions."

Lee, ¶ 0025. *See also* Lee, ¶ 0026.

> "The boot image memory 108 is capable of containing a non-volatile memory such as a flash memory to store a compressed boot image data. The boot image data can be obtained by compressing an initial storing state of the main memory 104 as a data format."

Lee, ¶ 0038.

> "The BIOS ROM 106 and the boot image memory 108 are capable of setting and storing an initial state of main memory by a manufacturer or a user. Thus, the CPU 102 can reduce a device drive loading time by reading out the compressed boot image from the boot image memory 108 and loading the boot image after decompressing, when boot image is loaded to the main memory 104."

Lee, ¶ 0040.

> "In other words, at the step S142, the CPU 102 executes the operating system if a successful boot is detected through a POST routine. Therefore, a certain application program can be executed in the operating system

environment."

Lee, ¶ 0042.

> "At step S148, the generated boot image is stored to the boot image memory 108 after compressing, and then the computer system 100 is rebooted."

Lee, ¶ 0043.

> "FIG. 5 is a flow chart for illustrating a method for booting the computer system 200 shown in FIG. 4. The control flow is a program stored in the BIOS ROM 210, and is executed by the CPU 202 according to processing steps of the BIOS."

Lee, ¶ 0050. *See also* Lee, ¶ 0053.

> "Continually, at step S224, a compressed boot image 216 is loaded from the CD-ROM 214, and the boot image is loaded to the main memory 206 after decompressing in step S226. At step S228, an instruction pointer 204 of the CPU 202 is set to a specific region 208 of the main memory 206 being loaded the boot image. At step S230, the operating system is executed by reading out the boot image from the specific region 208 of the main memory 206. As a result, the operating system can perform control functions."

Lee, ¶ 0051.

1.2 initializing a central processing unit of said computer system;	Lee, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

> "When power is applied to a computer system, a portion of the computer system typically called the "initialization hardware" electronically detects the "power-on" condition and, in response to such a detection, forces certain circuitry of the system to a known state. For example, the CPU typically includes an instruction pointer (IP), which holds a memory address from which the CPU fetches an instruction to be executed by the CPU. The initialization hardware typically electronically forces the IP to an initial address so that the CPU may begin fetching and executing instructions from this initial address. The ROM is prerecorded With computer instructions, referred to below as the "ROM-based code." As a result, shortly after power on, the CPU begins executing the ROM-based code."

Lee, ¶ 0011.

> "The BIOS ROM 106 controls POST routine, interrupt processing, and system environment setting, according to initializing steps of the computer system 100. Especially, the BIOS ROM 106 sets the instruction pointer (IP)."

Lee, ¶ 0039.

> "In other words, at the step S142, the CPU 102 executes the operating system if a successful boot is detected through a POST routine. Therefore, a certain application program can be executed in the operating system environment."

Lee, ¶ 0042.

> "Referring to FIG. 3, the computer system 100 is powered on in step

S160, and POST routine is performed in step S162."

Lee, ¶ 0045. *See also* Lee, ¶ 0051.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Lee, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

> "The ROM-based code attempts to establish communication with a so-called "boot device". A boot device holds information that is necessary to boot the system. In attempting to establish communication with a boot device, the ROM-based code operates according to a so-called "boot order.""

Lee, ¶ 0012.

> "More particularly, the ROM-based code attempts to retrieve a so-called "master boot record" from a particular sector of the diskette. If the communication attempt is successful, the ROM-based code uses that device as the boot device. If not, the ROM-based code proceeds to attempt communication With a device of the next boot order, e.g., local media."

Lee, ¶ 0012.

> "Assuming that the correct first diskette is inserted into the system, the ROM-based code retrieves the master boot record from sector 0 of the first diskette."

Lee, ¶ 0014.

> "The ROM-based code then copies the OS loader into RAM from the first diskette, starting at the address indicated in the master boot record and continuing for a length indicated by the offset provided by the master boot record. After copying the OS loader into RAM, the ROM-based code jumps to the OS loader."

Lee, ¶ 0014.

> "The OS loader is more sophisticated than the ROM-based code and performs certain preliminary functions, such as sizing memory. After performing preliminary functions, the OS loader copies into RAM a portion of the operating system known as the "kernel.""

Lee, ¶ 0015.

> "In order to attain the above objects, according to an aspect of the present invention, there is provided a computer system having a central processing unit; a main and/or auxiliary power supply for supplying main and/or auxiliary power of the computer system; a boot image storing device for storing a boot image of the computer system; a main memory for storing the boot image from the boot image storing device by receiving the auxiliary power when the main power is shut off; and a composition memory for setting an instruction pointer of the central processing unit to a specific region of the main memory storing the boot image; wherein the central processing unit loads the boot image from the specific region of the main memory in response to the instruction pointer, thereby an operating system program can perform control functions."

Lee, ¶ 0025. *See also* Lee, ¶ 0026.

> "The boot image memory 108 is capable of containing a non-volatile memory such as a flash memory to store a compressed boot image data. The boot image data can be obtained by compressing an initial storing state of the main memory 104 as a data format."

Lee, ¶ 0038.

> "The BIOS ROM 106 and the boot image memory 108 are capable of setting and storing an initial state of main memory by a manufacturer or a user. Thus, the CPU 102 can reduce a device drive loading time by reading out the compressed boot image from the boot image memory 108 and loading the boot image after decompressing, when boot image is loaded to the main memory 104."

Lee, ¶ 0040.

> "Continually, at step S164, the compressed boot image is read out. At step S166, the compressed boot image is loaded to the main memory 104 after decompressing. At step S168, an instruction pointer (IP) of the CPU 102 is set to a specific region of the main memory 104 being loaded the boot image. And then at step S170, the operating system is executed by

reading out the boot image from the specific region."

Lee, ¶ 0045. *See also* Lee, ¶ 0048.

> "Continually, at step S224, a compressed boot image 216 is loaded from the CD-ROM 214, and the boot image is loaded to the main memory 206 after decompressing in step S226. At step S228, an instruction pointer 204 of the CPU 202 is set to a specific region 208 of the main memory 206 being loaded the boot image. At step S230, the operating system is executed by reading out the boot image from the specific region 208 of the main memory 206. As a result, the operating system can perform control functions."

Lee, ¶ 0051.

> "The CPU 302 reads out the boot image 324 from the hard disk drive 320 and loads it to the main memory 304, under control of the BIOS. In other words, the CPU 302 decompresses the compressed boot image from the specific region of the hard disk drive 320, and loads it to a specific region of the main memory 304. The CPU 302 reads out the location information of the boot image from the BIOS ROM 306, and then reads out the boot image from the specific region of the main memory 304 according to the information. Thus, the operating system can perform control functions 15 by setting the IP of the CPU 302 to the specific region of the main memory 304."

Lee, ¶ 0054.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Lee, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

> "The BIOS ROM 106 and the boot image memory 108 are capable of setting and storing an initial state of main memory by a manufacturer or a user. Thus, the CPU 102 can reduce a device drive loading time by reading out the compressed boot image from the boot image memory 108 and loading the boot image after decompressing, when boot image is loaded to the main memory 104."

Lee, ¶ 0040.

> "Continually, at step S164, the compressed boot image is read out. At step S166, the compressed boot image is loaded to the main memory 104 after decompressing. At step S168, an instruction pointer (IP) of the CPU 102 is set to a specific region of the main memory 104 being loaded the boot image. And then at step S170, the operating system is executed by reading out the boot image from the specific region."

Lee, ¶ 0045. *See also* Lee, ¶ 0048.

> "Continually, at step S224, a compressed boot image 216 is loaded from the CD-ROM 214, and the boot image is loaded to the main memory 206 after decompressing in step S226. At step S228, an instruction pointer 204 of the CPU 202 is set to a specific region 208 of the main memory 206 being loaded the boot image. At step S230, the operating system is executed by reading out the boot image from the specific region 208 of the main memory 206. As a result, the operating system can perform control functions."

Lee, ¶ 0051.

> "The CPU 302 reads out the boot image 324 from the hard disk drive 320

> and loads it to the main memory 304, under control of the BIOS. In other words, the CPU 302 decompresses the compressed boot image from the specific region of the hard disk drive 320, and loads it to a specific region of the main memory 304. The CPU 302 reads out the location information of the boot image from the BIOS ROM 306, and then reads out the boot image from the specific region of the main memory 304 according to the information. Thus, the operating system can perform control functions 15 by setting the IP of the CPU 302 to the specific region of the main memory 304."

Lee, ¶ 0054.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Lee, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

> "The BIOS ROM 106 and the boot image memory 108 are capable of setting and storing an initial state of main memory by a manufacturer or a user. Thus, the CPU 102 can reduce a device drive loading time by reading out the compressed boot image from the boot image memory 108 and loading the boot image after decompressing, when boot image is loaded to the main memory 104."

Lee, ¶ 0040.

> "Continually, at step S164, the compressed boot image is read out. At step S166, the compressed boot image is loaded to the main memory 104 after decompressing. At step S168, an instruction pointer (IP) of the CPU 102 is set to a specific region of the main memory 104 being loaded the boot image. And then at step S170, the operating system is executed by reading out the boot image from the specific region."

Lee, ¶ 0045. *See also* Lee, ¶ 0048.

> "Continually, at step S224, a compressed boot image 216 is loaded from the CD-ROM 214, and the boot image is loaded to the main memory 206 after decompressing in step S226. At step S228, an instruction pointer 204 of the CPU 202 is set to a specific region 208 of the main memory 206 being loaded the boot image. At step S230, the operating system is executed by reading out the boot image from the specific region 208 of the main memory 206. As a result, the operating system can perform control functions."

Lee, ¶ 0051.

> "The CPU 302 reads out the boot image 324 from the hard disk drive 320 and loads it to the main memory 304, under control of the BIOS. In other words, the CPU 302 decompresses the compressed boot image from the specific region of the hard disk drive 320, and loads it to a specific region of the main memory 304. The CPU 302 reads out the location information of the boot image from the BIOS ROM 306, and then reads out the boot image from the specific region of the main memory 304 according to the information. Thus, the operating system can perform control functions 15 by setting the IP of the CPU 302 to the specific region of the main memory 304."

Lee, ¶ 0054.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Lee, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Lee, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The initial storing state is capable of executing a certain application program in an operating system program environment."

Lee, ¶ 0038.

> "In other words, at the step S142, the CPU 102 executes the operating system if a successful boot is detected through a POST routine. Therefore, a certain application program can be executed in the operating system environment."

Lee, ¶ 0042.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Lee, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The initial storing state is capable of executing a certain application program in an operating system program environment."

Lee, ¶ 0038.

> "In other words, at the step S142, the CPU 102 executes the operating system if a successful boot is detected through a POST routine. Therefore, a certain application program can be executed in the operating system environment."

Lee, ¶ 0042.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Lee, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The computer system 100 is an IBM compatible computer system, Which includes a plurality of controllers(for example, an input/output (I/O) controller 110, a hard disk drive (HDD) controller 112, and a floppy disk drive(EDD) controller 114), input devices including a keyboard 118 and a mouse 120, and auxiliary storing devices including a hard disk drive (HDD) 122, a CD-ROM drive 124, a floppy disk drive (FDD) 126, and so on. Further, the computer system 100 includes a video controller 116, and a display 128."

Lee, ¶ 0037.

> "In addition, the computer system 300 further includes a hard disk controller 308, and a hard disk drive (HDD) 320 storing an operating system program 322 and boot image 324."

Lee, ¶ 0052.

6. The method of claim 1, further comprising updating the list of boot data.	Lee, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Lee, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Lee, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "At step S148, the generated boot image is stored to the boot image memory 108 after compressing, and then the computer system 100 is rebooted."

Lee, ¶ 0043.

11.1. a processor;	Lee, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Lee, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Lee, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Lee discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> "The boot image memory 108 is capable of containing a non-volatile memory such as a flash memory to store a compressed boot image data. The boot image data can be obtained by compressing an initial storing state of the main memory 104 as a data format."

Lee, ¶ 0038.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Lee, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Lee, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Lee, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Lee, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Lee, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Lee, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Lee discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B13
Invalidity of U.S. Patent 8,090,936 based on Makinen

U.S. Patent No. 6,237,080 to Makinen ("Makinen") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B13
Invalidity of U.S. Patent 8,090,936 based on Makinen

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Makinen, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the

microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> According to a first aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) with a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), the method comprising reading a set of compressed RISC operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions.

Makinen, 1:63-2:4

> According to a second aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) and a read only memory (ROM), the method comprising reading a set of compressed operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions, the method further comprising generating one or more replacement or additional compressed instructions in the CPU and writing the compressed instruction(s) to the ROM.
>
> The above second aspect of the present invention makes it possible to amend the stored compressed instructions in a dynamic manner. This may, for example, allow a user to configure the computer according to his specific needs.
>
> Preferably, the method comprises the step of reading a set of operating instructions from the ROM into the CPU, which instructions define a program for compressing said replacement or additional instruction(s). More preferably, the instructions defining the compression program form part of said set of compressed operating instructions.

Makinen, 2:18-39

> Preferably, the method of the above first or second aspect of the invention comprises writing the decompressed instruction set to a random access memory (RAM). Thereafter, the decompressed instructions are read from the RAM by the CPU. It is noted that RAM typically offers high access speeds compared to slow (e.g. flash) ROM memory, giving a significant increase in system performance. In this

> case, increased speed also offers reduced power consumption compared to systems which use slow ROM memory and in which power is consumed even when the system is waiting to access the ROM.

Makinen, 2:40-50

> According to a third aspect of the present invention there is provided apparatus having a central processing unit (CPU) a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), there being stored in the ROM a set of compressed RISC operating instructions, the CPU being arranged in use to read the compressed instructions from the ROM, to decompress these instructions, and subsequently to operate the apparatus in accordance with the decompressed instructions.

Makinen, 2:51-59

> According to a fourth aspect of the present invention there is provided apparatus comprising a central processing unit (CPU), a read only memory (ROM), and a set of compressed operating instructions stored in the ROM, the CPU being arranged in use to read the compressed instructions from the ROM, decompress the compressed instructions, and thereafter operate the apparatus in accordance with the decompressed instructions, the apparatus being further arranged in use to compress replacement or additional operating instructions and to write these compressed instructions to the ROM.

Makinen, 2:60-3:3:2

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program

Makinen, 3:45-50

> In order to enable the microprocessor 1 to decompress the stored compressed instructions, the flash ROM 4 additionally stores a set of instructions, in uncompressed form, which define the Pkzip decompression program. FIG. 2 shows a memory map of the ROM 4 prior to booting the computer, where a part of the memory space is occupied by the compressed instructions 6 and a part is occupied by the uncompressed Pkzip instructions 7. In FIG. 2, the ROM 4 is shown

> coupled to the microprocessor 1, as is the RAM 3 which at this stage remains empty.

Makinen, 3:56-65

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6′ is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6′.

Makinen, 3:66-4:9

> It will be appreciated that the operating system of the computer may be altered by directly accessing the flash ROM 5 to erase and/or rewrite compressed instructions 6 stored therein. However, in some circumstances it may be desirable for the end-user to be able to alter the compressed operating instructions 6, or indeed for the computer itself to be able to 'dynamically' alter the instructions. To this end, the Pkzip compression method may also be stored in the flash ROM 4.

Makinen, 4:10-17

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8′ together with the decompressed operating instructions 6′ (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6,8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended version of the operating instructions 6′,8′ and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen, 4:18-32

1.2 initializing a central processing unit of said computer system;	Makinen, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed

> instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6′ is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6′.
>
> Makinen, 3:66-4:9
>
> *See also* Makinen, Fig. 3

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Makinen, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> According to a first aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) with a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), the method comprising reading a set of compressed RISC operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions.

Makinen, 1:63-2:4

> According to a second aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) and a read only memory (ROM), the method comprising reading a set of compressed operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions, the method further comprising generating one or more replacement or additional compressed instructions in the CPU and writing the compressed instruction(s) to the ROM.
>
> The above second aspect of the present invention makes it possible to amend the stored compressed instructions in a dynamic manner. This may, for example, allow a user to configure the computer according to his specific needs.
>
> Preferably, the method comprises the step of reading a set of operating instructions from the ROM into the CPU, which instructions define a program for compressing said replacement or additional instruction(s). More preferably, the instructions defining the compression program form part of said set of compressed operating instructions.

Makinen, 2:18-39

> Preferably, the method of the above first or second aspect of the invention comprises writing the decompressed instruction set to a random access memory (RAM). Thereafter, the decompressed instructions are read from the RAM by the CPU. It is noted that RAM typically offers high access speeds compared to slow (e.g. flash) ROM memory, giving a significant increase in system performance. In this case, increased speed also offers reduced power consumption compared to systems which use slow ROM memory and in which power is consumed even when the system is waiting to access the ROM.

Makinen, 2:40-50

> According to a third aspect of the present invention there is provided apparatus having a central processing unit (CPU) a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), there being stored in the ROM a set of compressed RISC operating instructions, the CPU being arranged in use to read the compressed instructions from the ROM, to decompress these instructions, and subsequently to operate the apparatus in accordance with the decompressed instructions.

Makinen, 2:51-59

> According to a fourth aspect of the present invention there is provided apparatus comprising a central processing unit (CPU), a read only memory (ROM), and a set of compressed operating instructions stored in the ROM, the CPU being arranged in use to read the compressed instructions from the ROM, decompress the compressed instructions, and thereafter operate the apparatus in accordance with the decompressed instructions, the apparatus being further arranged in use to compress replacement or additional operating instructions and to write these compressed instructions to the ROM.

Makinen, 2:60-3:3:2

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program

Makinen, 3:45-50

> In order to enable the microprocessor 1 to decompress the stored compressed instructions, the flash ROM 4 additionally stores a set of instructions, in uncompressed form, which define the Pkzip decompression program. FIG. 2 shows a memory map of the ROM 4 prior to booting the computer, where a part of the memory space is occupied by the compressed instructions 6 and a part is occupied by the uncompressed Pkzip instructions 7. In FIG. 2, the ROM 4 is shown coupled to the microprocessor 1, as is the RAM 3 which at this stage remains empty.

Makinen, 3:56-65

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6′ is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6′.

Makinen, 3:66-4:9

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8′ together with the decompressed operating instructions 6′ (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6,8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended version of the operating instructions 6′,8′ and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen, 4:18-32

See also Makinen, Figs. 2-4

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Makinen, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> According to a first aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) with a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), the method comprising reading a set of compressed RISC operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions.

Makinen, 1:63-2:4

> According to a second aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) and a read only memory (ROM), the method comprising reading a set of compressed operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions, the method further comprising generating one or more replacement or additional compressed instructions in the CPU and writing the compressed instruction(s) to the ROM.
>
> The above second aspect of the present invention makes it possible to amend the stored compressed instructions in a dynamic manner. This may, for example, allow a user to configure the computer according to his specific needs.
>
> Preferably, the method comprises the step of reading a set of operating instructions from the ROM into the CPU, which instructions define a program for compressing said replacement or additional instruction(s). More preferably, the instructions defining the compression program form part of said set of compressed operating instructions.

Makinen, 2:18-39

> Preferably, the method of the above first or second aspect of the invention comprises writing the decompressed instruction set to a random access memory (RAM). Thereafter, the decompressed instructions are read from the RAM by the CPU. It is noted that RAM typically offers high access speeds compared to slow (e.g. flash) ROM memory, giving a significant increase in system performance. In this case, increased speed also offers reduced power consumption compared to systems which use slow ROM memory and in which power is consumed even when the system is waiting to access the ROM.

Makinen, 2:40-50

> According to a third aspect of the present invention there is provided apparatus having a central processing unit (CPU) a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), there being stored in the ROM a set of compressed RISC operating instructions, the CPU being arranged in use to read the compressed instructions from the ROM, to decompress these instructions, and subsequently to operate the apparatus in accordance with the decompressed instructions.

Makinen, 2:51-59

> According to a fourth aspect of the present invention there is provided apparatus comprising a central processing unit (CPU), a read only memory (ROM), and a set of compressed operating instructions stored in the ROM, the CPU being arranged in use to read the compressed instructions from the ROM, decompress the compressed instructions, and thereafter operate the apparatus in accordance with the decompressed instructions, the apparatus being further arranged in use to compress replacement or additional operating instructions and to write these compressed instructions to the ROM.

Makinen, 2:60-3:3:2

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program

Makinen, 3:45-50

> In order to enable the microprocessor 1 to decompress the stored compressed instructions, the flash ROM 4 additionally stores a set of instructions, in uncompressed form, which define the Pkzip decompression program. FIG. 2 shows a memory map of the ROM 4 prior to booting the computer, where a part of the memory space is occupied by the compressed instructions 6 and a part is occupied by the uncompressed Pkzip instructions 7. In FIG. 2, the ROM 4 is shown coupled to the microprocessor 1, as is the RAM 3 which at this stage remains empty.

Makinen, 3:56-65

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6′ is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6′.

Makinen, 3:66-4:9

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8′ together with the decompressed operating instructions 6′ (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6,8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended version of the operating instructions 6′,8′ and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen, 4:18-32

See also Makinen, Figs. 2-4

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Makinen, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> According to a first aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) with a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), the method comprising reading a set of compressed RISC operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions.

Makinen, 1:63-2:4

> According to a second aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) and a read only memory (ROM), the method comprising reading a set of compressed operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions, the method further comprising generating one or more replacement or additional compressed instructions in the CPU and writing the compressed instruction(s) to the ROM.
>
> The above second aspect of the present invention makes it possible to amend the stored compressed instructions in a dynamic manner. This may, for example, allow a user to configure the computer according to his specific needs.
>
> Preferably, the method comprises the step of reading a set of operating instructions from the ROM into the CPU, which instructions define a program for compressing said replacement or additional instruction(s). More preferably, the instructions defining the compression program form part of said set of compressed operating instructions.

Makinen, 2:18-39

> Preferably, the method of the above first or second aspect of the invention comprises writing the decompressed instruction set to a random access memory (RAM). Thereafter, the decompressed instructions are read from the RAM by the CPU. It is noted that RAM typically offers high access speeds compared to slow (e.g. flash) ROM memory, giving a significant increase in system performance. In this case, increased speed also offers reduced power consumption compared to systems which use slow ROM memory and in which power is consumed even when the system is waiting to access the ROM.

Makinen, 2:40-50

> According to a third aspect of the present invention there is provided apparatus having a central processing unit (CPU) a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), there being stored in the ROM a set of compressed RISC operating instructions, the CPU being arranged in use to read the compressed instructions from the ROM, to decompress these instructions, and subsequently to operate the apparatus in accordance with the decompressed instructions.

Makinen, 2:51-59

> According to a fourth aspect of the present invention there is provided apparatus comprising a central processing unit (CPU), a read only memory (ROM), and a set of compressed operating instructions stored in the ROM, the CPU being arranged in use to read the compressed instructions from the ROM, decompress the compressed instructions, and thereafter operate the apparatus in accordance with the decompressed instructions, the apparatus being further arranged in use to compress replacement or additional operating instructions and to write these compressed instructions to the ROM.

Makinen, 2:60-3:3:2

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program

Makinen, 3:45-50

> In order to enable the microprocessor 1 to decompress the stored compressed instructions, the flash ROM 4 additionally stores a set of instructions, in uncompressed form, which define the Pkzip decompression program. FIG. 2 shows a memory map of the ROM 4 prior to booting the computer, where a part of the memory space is occupied by the compressed instructions 6 and a part is occupied by the uncompressed Pkzip instructions 7. In FIG. 2, the ROM 4 is shown coupled to the microprocessor 1, as is the RAM 3 which at this stage remains empty.

Makinen, 3:56-65

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6′ is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6′.

Makinen, 3:66-4:9

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8′ together with the decompressed operating instructions 6′ (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6,8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended version of the operating instructions 6′,8′ and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen, 4:18-32

See also Makinen, Figs. 2-4

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Makinen, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Makinen, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program. The compressed instructions occupy considerably less flash ROM space than would the corresponding uncompressed instructions, e.g. ½ to ⅕ of the memory space. Such a high compression ratio results from the particular structure of RISC instructions.

Makinen, 3:46-55

Appendix B13
Invalidity of U.S. Patent 8,090,936 based on Makinen

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Makinen, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program. The compressed instructions occupy considerably less flash ROM space than would the corresponding uncompressed instructions, e.g. ½ to ⅕ of the memory space. Such a high compression ratio results from the particular structure of RISC instructions.

Makinen, 3:46-55

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Makinen, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> Upon booting of the computer, the microprocessor 1 is directed by hardwired logic to read the first instruction of the decompressed instruction set 7, from the ROM 4. Thereafter the microprocessor is directed, by the decompressed set, to read and decompress the compressed instruction set 6. The expanded instruction set 6' is then stored in the RAM 3 as shown in FIG. 3. The last operation that the set of decompression instructions perform is to cause the microprocessor 1 to jump to the first instruction in the decompressed code. Thereafter, the computer is operated in accordance with the decompressed operating instructions 6'.

Makinen 3:66-4:28

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8' together with the decompressed operating instructions 6' (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6, 8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended version of the operating instructions 6', 8' and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen 4:18-37

6. The method of claim 1, further comprising updating the list of boot data.	Makinen, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

Makinen discloses this claim limitation:

> A computer having a reduced instruction computer (RISC) architecture has a RISC central processing unit (CPU)(1) coupled to a RAM memory (3) and to a flash ROM memory (4). A set of compressed operating instructions (6,8), including a subset defining a compression method (8), are stored in the flash ROM (4) together with a set of uncompressed instructions (7) defining a compression algorithm. Upon booting of the computer, the uncompressed instructions (7) are read from the ROM (4) by the CPU (1) which then also reads the compressed instructions (6,8), decompresses them according to the decompression process (7), and writes the decompressed instructions (6′,8′) to the RAM (3). The compressed instructions (6,8) can be dynamically altered by the CPU (1), by generating an altered set of uncompressed instructions, compressing these in accordance with the now decompressed compression method (8′), and writing these to the flash ROM (4).

Makinen, Abstract

> At the heart of most modern electronic devices is a microprocessor or central processing unit which operates in accordance with a set of software operating instructions which together form an executable program. The instructions are stored in a digital memory which may be internal to the microprocessor or, as is more usually the case, externally connected to the microprocessor. The set of operating instructions generally define the basic input/output system (BIOS) of the microprocessor together with device drivers, libraries, and user applications.

Makinen, 1:10-19

> According to a first aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) with a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), the method comprising reading a set of compressed RISC operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions.

Makinen, 1:63-2:4

> According to a second aspect of the present invention there is provided a method of operating apparatus having a central processing unit (CPU) and a read only memory (ROM), the method comprising reading a set of compressed operating instructions from the ROM into the CPU, decompressing the compressed instructions in the CPU, and thereafter operating the apparatus in accordance with the decompressed instructions, the method further comprising generating one or more replacement or additional compressed instructions in the CPU and writing the compressed instruction(s) to the ROM.
>
> The above second aspect of the present invention makes it possible to amend the stored compressed instructions in a dynamic manner. This may, for example, allow a user to configure the computer according to his specific needs.
>
> Preferably, the method comprises the step of reading a set of operating instructions from the ROM into the CPU, which instructions define a program for compressing said replacement or additional instruction(s). More preferably, the instructions defining the compression program form part of said set of compressed operating instructions.

Makinen, 2:18-39

> Preferably, the method of the above first or second aspect of the invention comprises writing the decompressed instruction set to a random access memory (RAM). Thereafter, the decompressed instructions are read from the RAM by the CPU. It is noted that RAM typically offers high access speeds compared to slow (e.g. flash) ROM memory, giving a significant increase in system performance. In this case, increased speed also offers reduced power consumption compared to systems which use slow ROM memory and in which power is consumed even when the system is waiting to access the ROM.

Makinen, 2:40-50

> According to a third aspect of the present invention there is provided apparatus having a central processing unit (CPU) a reduced instruction set computer (RISC) architecture, and a read only memory (ROM), there being stored in the ROM a set of compressed RISC operating instructions, the CPU being arranged in use to read the compressed instructions from the ROM, to decompress these instructions, and subsequently to operate the apparatus in accordance with the decompressed instructions.

Makinen, 2:51-59

> According to a fourth aspect of the present invention there is provided apparatus comprising a central processing unit (CPU), a read only memory (ROM), and a set of compressed operating instructions stored in the ROM, the CPU being arranged in use to read the compressed instructions from the ROM, decompress the compressed instructions, and thereafter operate the apparatus in accordance with the decompressed instructions, the apparatus being further arranged in use to compress replacement or additional operating instructions and to write these compressed instructions to the ROM.

Makinen, 2:60-3:3:2

> It will be appreciated that the operating system of the computer may be altered by directly accessing the flash ROM 5 to erase and/or rewrite compressed instructions 6 stored therein. However, in some circumstances it may be desirable for the end-user to be able to alter the compressed operating instructions 6, or indeed for the computer itself to be able to 'dynamically' alter the instructions. To this end, the Pkzip compression method may also be stored in the flash ROM 4.

Makinen, 4:10-17

> In order to reduce memory requirements, the corresponding Pkzip instructions may be stored in compressed form 8 as illustrated in FIG. 4. During booting, the compressed Pkzip instructions 8 are read from the flash ROM 4 by the microprocessor 1, decompressed, and stored in the RAM 3 as decompressed instructions 8′ together with the decompressed operating instructions 6′ (FIG. 4). Alternatively, the compressed Pkzip instructions 8 may only be read from the flash ROM 4, and subsequently decompressed, when a specific request is made to alter the compressed instructions 6,8. In either case, the microprocessor 1 employs the decompressed Pkzip method to compress an amended

version of the operating instructions 6′,8′ and writes the compressed instructions to the corresponding areas of the flash ROM.

Makinen, 4:18-32

See also Makinen, Figs. 2-4

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Makinen, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The flash ROM 4 is used to store a set of RISC operating instructions which define the basic input/output system (BIOS) of the microprocessor as well as the device drivers, libraries, and user applications. The instructions are in compressed form, having previously been compressed using the Pkzip compression program. The compressed instructions occupy considerably less flash ROM space than would the corresponding uncompressed instructions, e.g. ½ to ⅕ of the memory space. Such a high compression ratio results from the particular structure of RISC instructions.

3:46-55

> It will be appreciated by the person of skill in the art that other modifications may be made to the embodiments described above without departing from the scope of the present invention. For example, compression algorithms other than Pkzip may be used, including Gzip-9, Zoo-a, and Arj-a. Compression methods may also be optimised for ARM (Trade Mark) processors.

4:38-45

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Makinen, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> Compared to conventional ROM, flash ROM remains expensive. There therefore exists a desire to reduce the amount of flash ROM used in individual products. Furthermore, accessing flash ROM is relatively slow, significantly reducing the performance of electronic devices. Conventional computer architectures rely upon a complex instruction set computer (CISC) architecture. This utilises a large number (e.g. 1000) of instructions which define very specific tasks. The instructions are of variable length and are decoded by the computer's CPU before execution. There is described in Japanese non-examined patent publication no. 55-131848 a data processing unit which comprises a central processing unit, a main memory unit, and an external memory unit. A compressed program is stored in the external memory unit and, before commencing processing operations, the compressed program is read from the external memory unit in blocks and decompressed. The decompressed program is then written to the main memory unit. However, the compression ratio which can be achieved with CISC code is relatively low and it is not believed that the disclosure of JP-131848 has been widely used.

Makinen, 1:34-55

11.1. a processor;	Makinen, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Makinen, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Makinen, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Makinen discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

> Software instructions defining the basic operation of a microprocessor are usually stored in non-volatile read only memory (ROM). Until recently, the preferred choice for storing operating instructions was UV light erasable programmable read only memory. More recently however, the preferred choice for storing operating instructions, especially in embedded devices (e.g. mobile phones, personal digital assistants, etc) has become flash ROM. Flash ROM is both non-volatile and electrically erasable and is used, to a large extent, because it can be programmed following assembly of the PCB containing the flash ROM oar of the completed device. A further advantage is that it is possible to upgrade operating instruction stored in the flash ROM at some future date.

Makinen, 1:20-33

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Makinen, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Makinen, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Makinen, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Makinen, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Makinen, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B13
Invalidity of U.S. Patent 8,090,936 based on Makinen

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Makinen, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Makinen discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B14
Invalidity of U.S. Patent 8,090,936 based on Noll

U.S. Patent No. 5,793,943 to Noll ("Noll") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Noll, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

> "A computer system includes a dual basic input-output system (BIOS) read-only memory (ROM) system to initialize the computer. When the computer is first powered on or reset, the primary BIOS ROM is initially enabled. The computer analyzes the entire program contents of the primary BIOS memory to detect data errors. If a data error is detected, a chip enable circuit disables the primary BIOS ROM and enables a secondary BIOS ROM containing essentially the same initialization instructions as the primary BIOS ROM. If no errors are detected in the secondary BIOS ROM, the initialization of the computer proceeds using the secondary BIOS ROM. As part of the initialization procedure, the contents of the secondary BIOS ROM are copied to a random access memory. The primary BIOS ROM can then be reprogrammed with the contents of the secondary BIOS ROM using the copy in random access memory, or from the secondary BIOS ROM itself."

Noll, Abstract.

> "The present invention is embodied in a system for the automatic recovery of a BIOS ROM failure. In one embodiment, the system includes a first BIOS memory that contains a series of computer instructions to initialize the computer. The first BIOS memory has a chip enable input that is initially enabled. An error detection circuit analyzes data contained within the first BIOS memory and detects errors therein. The error detection circuit generates an error signal upon detection of

> errors in the first BIOS memory. The system also includes a second BIOS memory containing the series of computer instructions to initialize the computer and also having a chip enable input. An enabling circuit is included to disable the first BIOS memory chip enable input and to enable the second BIOS memory chip enable input in response to the error signal. This effectively causes the computer system to switch to the second BIOS memory so that the series of computer instructions to initialize the computer are executed from the second BIOS memory rather the first BIOS memory."

Noll, 1:44-63.

> "Another reason that the primary BIOS ROM 22 is copied into the RAM 14 is that some data in the primary BIOS ROM may be in a compressed format and must be decompressed before the CPU 12 can use it."

Noll, 4:66-5:2.

1.2 initializing a central processing unit of said computer system;	Noll, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

> "A computer system includes a dual basic input-output system (BIOS) read-only memory (ROM) system to initialize the computer. When the computer is first powered on or reset, the primary BIOS ROM is initially enabled. The computer analyzes the entire program contents of the primary BIOS memory to detect data errors. If a data error is detected, a chip enable circuit disables the primary BIOS ROM and enables a secondary BIOS ROM containing essentially the same initialization instructions as the primary BIOS ROM. If no errors are detected in the secondary BIOS ROM, the initialization of the computer proceeds using the secondary BIOS ROM. As part of the initialization procedure, the contents of the secondary BIOS ROM are copied to a random access memory. The primary BIOS ROM can then be reprogrammed with the contents of the secondary BIOS ROM using the copy in random access memory, or from the secondary BIOS ROM itself."

Noll, Abstract.

> "For example, when the computer is first powered up or reset, a software program, typically designated as a "basic input-output system" (BIOS) initializes the computer and permits the startup of an operating system, such as Microsoft MS-DOS®. The BIOS program typically resides in a read-only memory (ROM). If the BIOS ROM is defective for any reason, the computer will not function properly. Therefore, it can be appreciated that there is a significant need for a system to recover from a BIOS ROM failure in a manner that does not require user intervention. The present invention provides this and other advantages as will be apparent from the following detailed description and accompanying figures."

Noll, 1:29-42.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Noll, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

> "A computer system includes a dual basic input-output system (BIOS) read-only memory (ROM) system to initialize the computer. When the computer is first powered on or reset, the primary BIOS ROM is initially enabled. The computer analyzes the entire program contents of the primary BIOS memory to detect data errors. If a data error is detected, a chip enable circuit disables the primary BIOS ROM and enables a secondary BIOS ROM containing essentially the same initialization instructions as the primary BIOS ROM. If no errors are detected in the secondary BIOS ROM, the initialization of the computer proceeds using the secondary BIOS ROM. As part of the initialization procedure, the contents of the secondary BIOS ROM are copied to a random access memory. The primary BIOS ROM can then be reprogrammed with the contents of the secondary BIOS ROM using the copy in random access memory, or from the secondary BIOS ROM itself."

Noll, Abstract.

> "The present invention is embodied in a system for the automatic recovery of a BIOS ROM failure. In one embodiment, the system includes a first BIOS memory that contains a series of computer instructions to initialize the computer. The first BIOS memory has a chip enable input that is initially enabled. An error detection circuit analyzes data contained within the first BIOS memory and detects errors therein. The error detection circuit generates an error signal upon detection of errors in the first BIOS memory. The system also includes a second BIOS memory containing the series of computer instructions to initialize the computer and also having a chip enable input. An enabling circuit is included to disable the first BIOS memory chip enable input and to enable the second BIOS memory chip enable input in response to the error

signal. This effectively causes the computer system to switch to the second BIOS memory so that the series of computer instructions to initialize the computer are executed from the second BIOS memory rather the first BIOS memory."

Noll, 1:44-63.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Noll, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

> "A computer system includes a dual basic input-output system (BIOS) read-only memory (ROM) system to initialize the computer. When the computer is first powered on or reset, the primary BIOS ROM is initially enabled. The computer analyzes the entire program contents of the primary BIOS memory to detect data errors. If a data error is detected, a chip enable circuit disables the primary BIOS ROM and enables a secondary BIOS ROM containing essentially the same initialization instructions as the primary BIOS ROM. If no errors are detected in the secondary BIOS ROM, the initialization of the computer proceeds using the secondary BIOS ROM. As part of the initialization procedure, the contents of the secondary BIOS ROM are copied to a random access memory. The primary BIOS ROM can then be reprogrammed with the contents of the secondary BIOS ROM using the copy in random access memory, or from the secondary BIOS ROM itself."

Noll, Abstract.

> "For example, when the computer is first powered up or reset, a software program, typically designated as a "basic input-output system" (BIOS) initializes the computer and permits the startup of an operating system, such as Microsoft MS-DOS®. The BIOS program typically resides in a read-only memory (ROM). If the BIOS ROM is defective for any reason, the computer will not function properly. Therefore, it can be appreciated that there is a significant need for a system to recover from a BIOS ROM failure in a manner that does not require user intervention. The present invention provides this and other advantages as will be apparent from the

following detailed description and accompanying figures."

Noll, 1:29-42.

"Another reason that the primary BIOS ROM 22 is copied into the RAM 14 is that some data in the primary BIOS ROM may be in a compressed format and must be decompressed before the CPU 12 can use it."

Noll, 4:66-5:2.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Noll, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

> "A computer system includes a dual basic input-output system (BIOS) read-only memory (ROM) system to initialize the computer. When the computer is first powered on or reset, the primary BIOS ROM is initially enabled. The computer analyzes the entire program contents of the primary BIOS memory to detect data errors. If a data error is detected, a chip enable circuit disables the primary BIOS ROM and enables a secondary BIOS ROM containing essentially the same initialization instructions as the primary BIOS ROM. If no errors are detected in the secondary BIOS ROM, the initialization of the computer proceeds using the secondary BIOS ROM. As part of the initialization procedure, the contents of the secondary BIOS ROM are copied to a random access memory. The primary BIOS ROM can then be reprogrammed with the contents of the secondary BIOS ROM using the copy in random access memory, or from the secondary BIOS ROM itself."

Noll, Abstract.

> "For example, when the computer is first powered up or reset, a software program, typically designated as a "basic input-output system" (BIOS) initializes the computer and permits the startup of an operating system, such as Microsoft MS-DOS®. The BIOS program typically resides in a read-only memory (ROM). If the BIOS ROM is defective for any reason, the computer will not function properly. Therefore, it can be appreciated that there is a significant need for a system to recover from a BIOS ROM failure in a manner that does not require user intervention. The present invention provides this and other advantages as will be apparent from the

following detailed description and accompanying figures."

Noll, 1:29-42.

> "Another reason that the primary BIOS ROM 22 is copied into the RAM 14 is that some data in the primary BIOS ROM may be in a compressed format and must be decompressed before the CPU 12 can use it."

Noll, 4:66-5:2.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Noll, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Noll, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Noll, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Noll, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The computer 10 also includes a chip enable circuit 36 that selectively enables and disables the primary BIOS ROM 22 and the secondary BIOS ROM 30. When the computer 10 is first powered up, the chip enable circuit 36 activates a ROMSEL1 control line 40 and a ROMSEL2 control line 42 by setting both of these control lines to a high logic level. The ROMSEL1 control line 40 and the ROMSEL2 control line 42 serve as inputs to an AND gate 44 whose output is coupled to the chip enable input 24 on the primary BIOS ROM 22. Thus, the chip enable circuit 36 initially activates the chip enable input 24 on the primary BIOS ROM 22.

Noll, 3:25-37

> The ROMSEL2 control line 42 is also coupled to the input of an inverter 48 whose output is coupled to the input of an AND gate 50. The ROMSEL1 control line 40 serves as another input to the AND gate 50. The output of the AND gate 50 is coupled to the chip enable input 32 of the secondary BIOS ROM 30. The inversion of the ROMSEL2 control line 42 assures that the secondary BIOS ROM 30 is initially disabled when the computer 10 is first powered up, or reset. In the event that the computer 10 detects an error in the primary BIOS ROM 22, the chip enable circuit 36 causes the ROMSEL2 control line 42 to change to a low logic level. This disables the primary BIOS ROM 22, while also enabling the secondary BIOS ROM 30.

Noll, 3:38-50

> The chip enable circuit 36 may be virtually any type of data storage element, such as a register, that can be controlled by the CPU 12. In the presently preferred embodiment, the chip enable circuit 36 is part of a peripheral

component interconnect (PCI) local bus to an industry standard architecture (ISA) bus bridge chip (not shown). The computer 10 switches between the primary BIOS ROM 22 and the secondary BIOS ROM 30 by toggling the ROMSEL to control line 42 in the PCI to ISA bridge chip (not shown). However, those of ordinary skill in the art will readily recognize that any programmable register will operate satisfactorily with the computer 10.

Noll, 3:50-61

6. The method of claim 1, further comprising updating the list of boot data.	Noll, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Noll, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Noll, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Noll, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Noll, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Noll, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Noll discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Noll, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Noll, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Noll, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Noll, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Noll, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B14
Invalidity of U.S. Patent 8,090,936 based on Noll

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Noll, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Noll discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B15
Invalidity of U.S. Patent 8,090,936 based on Pearce

U.S. Patent No. 5,828,877 to Pearce ("Pearce") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B15
Invalidity of U.S. Patent 8,090,936 based on Pearce

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Pearce, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

> "During compression of the main memory, the record containing the identity of the allocable units compressed is also stored for use by the main memory restoration method detailed in FIG. 5."

Pearce, 8:27-30, Fig. 5.

> "As a part of the block 515, the units previously allocated to the reducing task are marked as unallocated and therefore free for subsequent allocation by the operating system."

Pearce, 8:67-9:3, Fig. 5.

See also Pearce 4:16-22.

1.2 initializing a central processing unit of said computer system;	Pearce, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

> "The most effective solution to power saving following a long period of user inactivity is to turn off the PC completely. However, it is unacceptable simply to interrupt has not yet been stored in non-volatile memory, particularly the hard disk drive. Rather, it is advantageous to store the contents of the volatile main memory of the PC as an image in the nonvolatile secondary storage device (a so-called "suspend-to-disk" or "STD"). When the user restores power to the PC, the image is restored to the main memory (a "resume-from-disk" or "RFD"), placing the PC in exactly the same functional state as it was when power was interrupted."

Pearce, 1:65-2:10.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Pearce, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

> "The most effective solution to power saving following a long period of user inactivity is to turn off the PC completely. However, it is unacceptable simply to interrupt has not yet been stored in non-volatile memory, particularly the hard disk drive. Rather, it is advantageous to store the contents of the volatile main memory of the PC as an image in the nonvolatile secondary storage device (a so-called "suspend-to-disk" or "STD"). When the user restores power to the PC, the image is restored to the main memory (a "resume-from-disk" or "RFD"), placing the PC in exactly the same functional state as it was when power was interrupted."

Pearce, 1:65-2:10.

> "Accordingly, a first embodiment of the present invention provides, in a computer system having a CPU, a main memory (either real or virtual) divisible into allocable units, a secondary storage unit and an operating system for allocating the allocable units to tasks for use thereby, a suspend circuit for creating an optimized compressed image of data in the main memory."

Pearce, 2:36-43.

> "Once the contents of main memory have been compressed and stored as an image in the secondary storage device, power to the computer system is interrupted."

Pearce, 2:57-59.

> "The present invention is designed to conserve computer power by storing the pertinent contents of the main memory 210 as a compressed

> image in the nonvolatile secondary storage device 260, allowing power to be interrupted to the portable PC 100 in its entirety. When the user so directs, power is restored to the PC, and the image is decompressed back into the main memory 210, allowing the CPU 200 to continue processing at the point where it stopped prior to interruption of power."
>
> Pearce, 5:36-44.
>
> "Execution begins in a start block 500 wherein the portable PC 100 boots, loading initial portions of code commonly stored in nonvolatile memory within the portable PC 100."
>
> Pearce, 8:39-41, Fig. 5.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Pearce, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

> "The most effective solution to power saving following a long period of user inactivity is to turn off the PC completely. However, it is unacceptable simply to interrupt has not yet been stored in non-volatile memory, particularly the hard disk drive. Rather, it is advantageous to store the contents of the volatile main memory of the PC as an image in the nonvolatile secondary storage device (a so-called "suspend-to-disk" or "STD"). When the user restores power to the PC, the image is restored to the main memory (a "resume-from-disk" or "RFD"), placing the PC in exactly the same functional state as it Was When power was interrupted."

Pearce, 1:65-2:10.

> "Accordingly, a first embodiment of the present invention provides, in a computer system having a CPU, a main memory (either real or virtual) divisible into allocable units, a secondary storage unit and an operating system for allocating the allocable units to tasks for use thereby, a suspend circuit for creating an optimized compressed image of data in the main memory."

Pearce, 2:36-43.

> "In a preferred embodiment, the suspend circuit further comprises a circuit for restoring the main memory by decompressing the compressed image into the main memory. The allocable units allocated to the reducing task are designated as unallocated units in the decompressed image."

Pearce, 3:47-52.

> "The present invention is designed to conserve computer power by

> storing the pertinent contents of the main memory 210 as a compressed image in the nonvolatile secondary storage device 260, allowing power to be interrupted to the portable PC 100 in its entirety. When the user so directs, power is restored to the PC, and the image is decompressed back into the main memory 210, allowing the CPU 200 to continue processing at the point where it stopped prior to interruption of power."

Pearce, 5:36-44.

> "If a compressed image file is detected, execution proceeds to a block 515, Wherein a corresponding (e.g., run length decoding) decompression routine decompresses and restores the main memory 210 to the state in which it was prior to shutdown. In the first embodiment, the units previously allocated to the reducing task remain filled with the bit pattern. In the second embodiment, the stored record is retrieved and used as a guide by the decompression routine to restore the compressed allocable units to their proper position in main memory."

Pearce, 8:52-61, Fig. 5.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Pearce, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

> "The most effective solution to power saving following a long period of user inactivity is to turn off the PC completely. However, it is unacceptable simply to interrupt has not yet been stored in non-volatile memory, particularly the hard disk drive. Rather, it is advantageous to store the contents of the volatile main memory of the PC as an image in the nonvolatile secondary storage device (a so-called "suspend-to-disk" or "STD"). When the user restores power to the PC, the image is restored to the main memory (a "resume-from-disk" or "RFD"), placing the PC in exactly the same functional state as it Was When power was interrupted."

Pearce, 1:65-2:10.

> "In a preferred embodiment, the suspend circuit further comprises a circuit for restoring the main memory by decompressing the compressed image into the main memory. The allocable units allocated to the reducing task are designated as unallocated units in the decompressed image."

Pearce, 3:47-52.

> "The present invention is designed to conserve computer power by storing the pertinent contents of the main memory 210 as a compressed image in the nonvolatile secondary storage device 260, allowing power to be interrupted to the portable PC 100 in its entirety. When the user so directs, power is restored to the PC, and the image is decompressed back into the main memory 210, allowing the CPU 200 to continue processing

at the point where it stopped prior to interruption of power."

Pearce, 5:36-44.

> "If a compressed image file is detected, execution proceeds to a block 515, wherein a corresponding (e.g., run length decoding) decompression routine decompresses and restores the main memory 210 to the state in which it was prior to shutdown. In the first embodiment, the units previously allocated to the reducing task remain filled with the bit pattern. In the second embodiment, the stored record is retrieved and used as a guide by the decompression routine to restore the compressed allocable units to their proper position in main memory."

Pearce, 8:52-61, Fig. 5.

See also Pearce 4:16-22.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Pearce, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Pearce, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "One of the purposes of the operating system is to allocate resources to tasks (or "application programs," such as Word processors, spreadsheets, communications programs, database managers, games and the like and their associated data) as they are executed on the portable PC 100."

Pearce, 5:55-61.

> "The user again loads an application task and begins to edit a document. For purposes of this example, it is again assumed that the user leaves the portable PC 100 Without saving the document. A period of inactivity thus begins."

Pearce, 7:65-8:3.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Pearce, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "One of the purposes of the operating system is to allocate resources to tasks (or "application programs," such as Word processors, spreadsheets, communications programs, database managers, games and the like and their associated data) as they are executed on the portable PC 100."

Pearce, 5:55-61.

> "The user again loads an application task and begins to edit a document. For purposes of this example, it is again assumed that the user leaves the portable PC 100 Without saving the document. A period of inactivity thus begins."

Pearce, 7:65-8:3.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Pearce, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "A bus controller 250 manages communication of data between the relatively fast local bus 240 and a relatively slow expansion bus 290 via lines 251, 252, respectively."

Pearce, 5:17-19.

6. The method of claim 1, further comprising updating the list of boot data.	Pearce, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "In a preferred embodiment, the suspend circuit further comprises a circuit for restoring the main memory by decompressing the compressed image into the main memory. The allocable units allocated to the reducing task are designated as unallocated units in the decompressed image."

Pearce, 3:47-52.

> "As a part of the block 515, the units previously allocated to the reducing task are marked as unallocated and therefore free for subsequent allocation by the operating system."

Pearce, 8:67-9:3, Fig. 5.

See also Pearce 4:16-22.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Pearce, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "There are a wide variety of conventional data compression algorithms that are suitable to compress the contents of main memory to create a compressed image to be stored in the secondary storage device. Those skilled in the art are familiar With standard compression algorithms such as run length encoding, adaptive pattern substitution, variable length character encoding (such as Huffman coding), restricted variability codes, dictionary substitution, differencing and ordered data schemes."

Pearce, 7:11-19.

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Pearce, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "(3) a circuit for executing the data compression process to store a compressed image of the main memory in the secondary storage unit, the bit pattern allowing a size of the compressed image to be reduced and a time required to compress and store the compressed image to be minimized."

Pearce, 2:49-53. *See also* Pearce, 3:36-42, 9:25-31.

11.1. a processor;	Pearce, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Pearce, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Pearce, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Pearce discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also ***add disclosure of non-volatile memory to the extent not in claim 1 already***

"The present invention is directed, in general, to computer systems and, more particularly, to a circuit and method for conserving power used by the computer system by storing a compressed image of the contents of the computer system's main memory on a nonvolatile secondary storage device, allowing power to the main memory to be interrupted."

Pearce, 1:6-11.

"However, it is unacceptable simply to interrupt power, because the user may have been engaged in work that has not yet been stored in non-volatile memory, particularly the hard disk drive. Rather, it is advantageous to store the contents of the volatile main memory of the PC as an image in the nonvolatile secondary storage device (a so-called "suspend-to-disk" or "STD")."

Pearce, 1:67-2:6.

"In a preferred embodiment, the secondary storage device is a

conventional, nonvolatile hard disk drive."

Pearce, 3:4-5.

> "The expansion bus 290 couples at least one nonvolatile secondary storage device 260 (such as a floppy, Bernoulli, magneto-optical, CD-ROM or hard disk drive), expansion slots 270 (capable or receiving daughter-cards providing functions such as facsimile or modem, network interface or sound) and ports 280 (for coupling a printer or serial devices to the portable PC 100) via lines 261, 271, 281, respectively."

Pearce, 5:19-26.

> "The present invention is designed to conserve computer power by storing the pertinent contents of the main memory 210 as a compressed image in the nonvolatile secondary storage device 260, allowing power to be interrupted to the portable PC 100 in its entirety."

Pearce, 5:35-39.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Pearce, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Pearce, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Pearce, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

Appendix B15
Invalidity of U.S. Patent 8,090,936 based on Pearce

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Pearce, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

Appendix B15
Invalidity of U.S. Patent 8,090,936 based on Pearce

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Pearce, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B15
Invalidity of U.S. Patent 8,090,936 based on Pearce

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Pearce, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Pearce discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B16
Invalidity of U.S. Patent 8,090,936 based on Rahman

U.S. Patent No. 5,901,310 Rahman ("Rahman") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

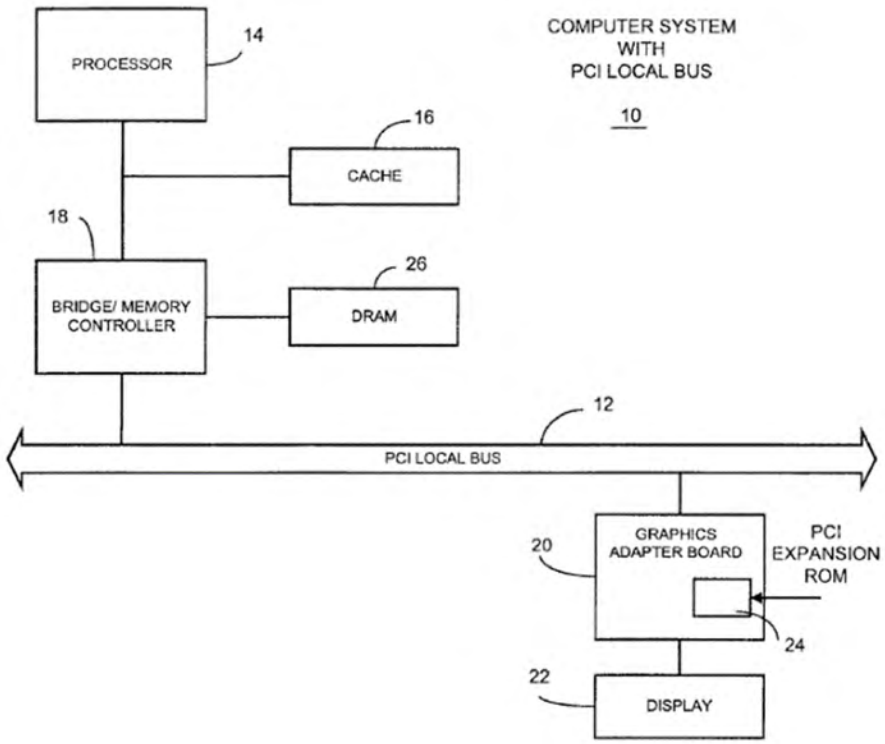
The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B16
Invalidity of U.S. Patent 8,090,936 based on Rahman

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Rahman, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:



Rahman, Fig. 1.

> "Initializing and configuring computer hardware with firmware stored in compressed form in nonvolatile semiconductor memory. Upon startup of the computer hardware, decompression software decompress the firmware, which is then stored in another memory. The computer hardware may be an adapter board (e.g., a graphics board connected to PCI bus), the nonvolatile semiconductor memory may be physically located on the adapter board, and the firmware may be firmware for initializing and configuring the adapter board."

Rahman, Abstract.

> "The firmware may be the BIOS for initializing and configuring a personal computer."

Rahman, 2:1-2.

> "Devices that connect to the PCI bus provide initialization code and runtime code for the device. This code resides in ROM 24 on the device. When the computer system starts up or initializes, it detects the device, reads the code from ROM into the host system memory, such as a RAM or dynamic random access memory 26 (DRAM), and interprets the code."

Rahman, 2:51-57.

> "When the computer system starts up or initializes, it detects the adapter board connected to the PCI bus. It then locates the decompression code 29 in the adapter board's ROM. The computer system processor runs an FCode interpreter, which interprets the instructions in the decompression program as it reads the code from the adapter board's ROM. The compressed code 28 is decompressed, using the computer system's memory (such as RAM or DRAM 26) to store the image. After the image is decompressed, the system recognizes the decompressed image as a device driver that configures and initializes the adapter board, and supplies the runtime set of instructions for the adapter board."

Rahman, 2:66-3:11.

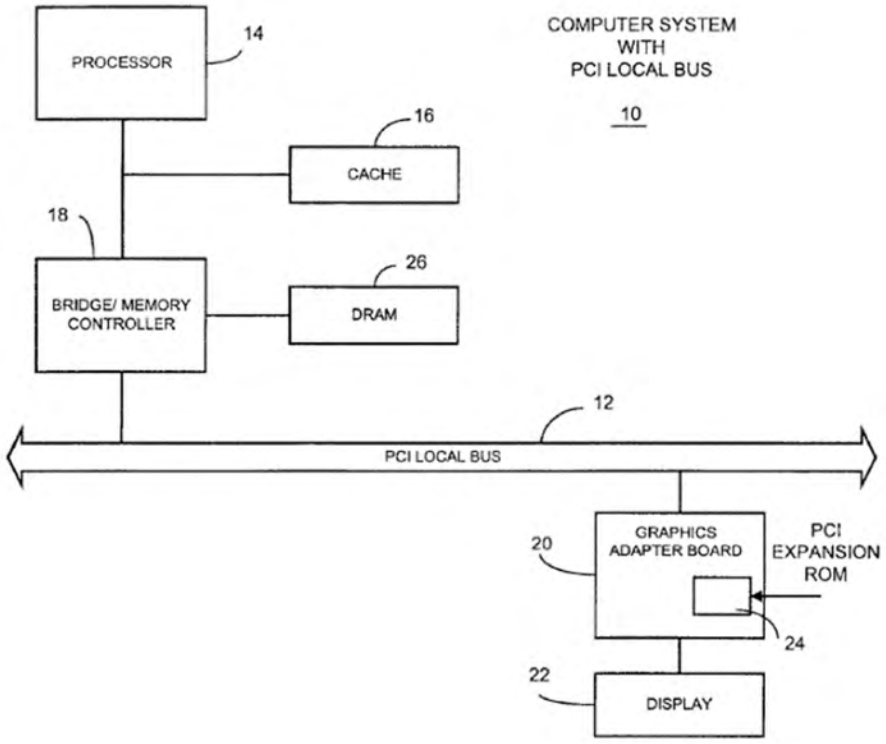
> "Other embodiments are within the scope of the following claims. For example, a personal computer's basic input/output system (BIOS) is firmware stored in ROM and could be stored in a compressed format. Furthermore, other types of nonvolatile semiconductor memory, such as programmable ROMs (PROMS) and erasable programmable ROMs

(EPROMs), could be used to store the compressed firmware." Rahman, 4:63-5:3.

1.2 initializing a central processing unit of said computer system;	Rahman, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:



Rahman, Fig. 1.

> "Initializing and configuring computer hardware with firmware stored in compressed form in nonvolatile semiconductor memory. Upon startup of the computer hardware, decompression software decompress the firmware, which is then stored in another memory. The computer hardware may be an adapter board (e.g., a graphics board connected to PCI bus), the nonvolatile semiconductor memory may be physically located on the adapter board, and the firmware may be firmware for

initializing and configuring the adapter board."

Rahman, Abstract.

> "The firmware may be the BIOS for initializing and configuring a personal computer."

Rahman, 2:1-2.

> "Devices that connect to the PCI bus provide initialization code and runtime code for the device. This code resides in ROM 24 on the device. When the computer system starts up or initializes, it detects the device, reads the code from ROM into the host system memory, such as a RAM or dynamic random access memory 26 (DRAM), and interprets the code."

Rahman, 2:51-57.

> "When the computer system starts up or initializes, it detects the adapter board connected to the PCI bus. It then locates the decompression code 29 in the adapter board's ROM. The computer system processor runs an FCode interpreter, which interprets the instructions in the decompression program as it reads the code from the adapter board's ROM. The compressed code 28 is decompressed, using the computer system's memory (such as RAM or DRAM 26) to store the image. After the image is decompressed, the system recognizes the decompressed image as a device driver that configures and initializes the adapter board, and supplies the runtime set of instructions for the adapter board."

Rahman, 2:66-3:11.

> "Other embodiments are within the scope of the following claims. For example, a personal computer's basic input/output system (BIOS) is firmware stored in ROM and could be stored in a compressed format. Furthermore, other types of nonvolatile semiconductor memory, such as programmable ROMs (PROMS) and erasable programmable ROMs (EPROMs), could be used to store the compressed firmware."

Rahman, 4:63-5:3.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Rahman, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

> "Initializing and configuring computer hardware with firmware stored in compressed form in nonvolatile semiconductor memory."

Rahman, Abstract.

> "When the computer system starts up or initializes, it detects the adapter board connected to the PCI bus. It then locates the decompression code 29 in the adapter board's ROM. The computer system processor runs an FCode interpreter, which interprets the instructions in the decompression program as it reads the code from the adapter board's ROM. The compressed code 28 is decompressed, using the computer system's memory (such as RAM or DRAM 26) to store the image. After the image is decompressed, the system recognizes the decompressed image as a device driver that configures and initializes the adapter board, and supplies the runtime set of instructions for the adapter board."

Rahman, 2:66-3:11.

> "Other embodiments are within the scope of the following claims. For example, a personal computer's basic input/output system (BIOS) is firmware stored in ROM and could be stored in a compressed format. Furthermore, other types of nonvolatile semiconductor memory, such as programmable ROMs (PROMS) and erasable programmable ROMs (EPROMs), could be used to store the compressed firmware."

Rahman, 4:63-5:3.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Rahman, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

> "Initializing and configuring computer hardware with firmware stored in compressed form in nonvolatile semiconductor memory."

Rahman, Abstract.

> "Upon startup of the computer hardware, decompression software decompress the firmware, which is then stored in another memory."

Rahman, Abstract.

> "The invention virtually increases the size of the nonvolatile semiconductor memory (e.g., a ROM) available for storing firmware by storing the firmware in compressed form and quickly decompressing it on startup. This permits the firmware memory to hold more instructions than would otherwise be possible. The invention is able to decompress firmware quickly, reliably, and fully automatically, so that the fact of the firmware being compressed is substantially invisible to the end-user."

Rahman, 1:48-56.

> "When the computer system starts up or initializes, it detects the adapter board connected to the PCI bus. It then locates the decompression code 29 in the adapter board's ROM. The computer system processor runs an FCode interpreter, which interprets the instructions in the decompression program as it reads the code from the adapter board's ROM. The compressed code 28 is decompressed, using the computer system's memory (such as RAM or DRAM 26) to store the image. After the image is decompressed, the system recognizes the decompressed image as a device driver that configures and initializes the adapter board, and

supplies the runtime set of instructions for the adapter board."

Rahman, 2:66-3:11.

> "Other embodiments are within the scope of the following claims. For example, a personal computer's basic input/output system (BIOS) is firmware stored in ROM and could be stored in a compressed format. Furthermore, other types of nonvolatile semiconductor memory, such as programmable ROMs (PROMS) and erasable programmable ROMs (EPROMs), could be used to store the compressed firmware."

Rahman, 4:63-5:3.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Rahman, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

> "Initializing and configuring computer hardware with firmware stored in compressed form in nonvolatile semiconductor memory."

Rahman, Abstract.

> "Upon startup of the computer hardware, decompression software decompress the firmware, which is then stored in another memory."

Rahman, Abstract.

> "The invention virtually increases the size of the nonvolatile semiconductor memory (e.g., a ROM) available for storing firmware by storing the firmware in compressed form and quickly decompressing it on startup. This permits the firmware memory to hold more instructions than would otherwise be possible. The invention is able to decompress firmware quickly, reliably, and fully automatically, so that the fact of the firmware being compressed is substantially invisible to the end-user."

Rahman, 1:48-56.

> "The firmware may be the BIOS for initializing and configuring a personal computer."

Rahman, 2:1-2.

> "When the computer system starts up or initializes, it detects the adapter

> board connected to the PCI bus. It then locates the decompression code 29 in the adapter board's ROM. The computer system processor runs an FCode interpreter, which interprets the instructions in the decompression program as it reads the code from the adapter board's ROM. The compressed code 28 is decompressed, using the computer system's memory (such as RAM or DRAM 26) to store the image. After the image is decompressed, the system recognizes the decompressed image as a device driver that configures and initializes the adapter board, and supplies the runtime set of instructions for the adapter board."

Rahman, 2:66-3:11.

> "Other embodiments are within the scope of the following claims. For example, a personal computer's basic input/output system (BIOS) is firmware stored in ROM and could be stored in a compressed format. Furthermore, other types of nonvolatile semiconductor memory, such as programmable ROMs (PROMS) and erasable programmable ROMs (EPROMs), could be used to store the compressed firmware."

Rahman, 4:63-5:3.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Rahman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Rahman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

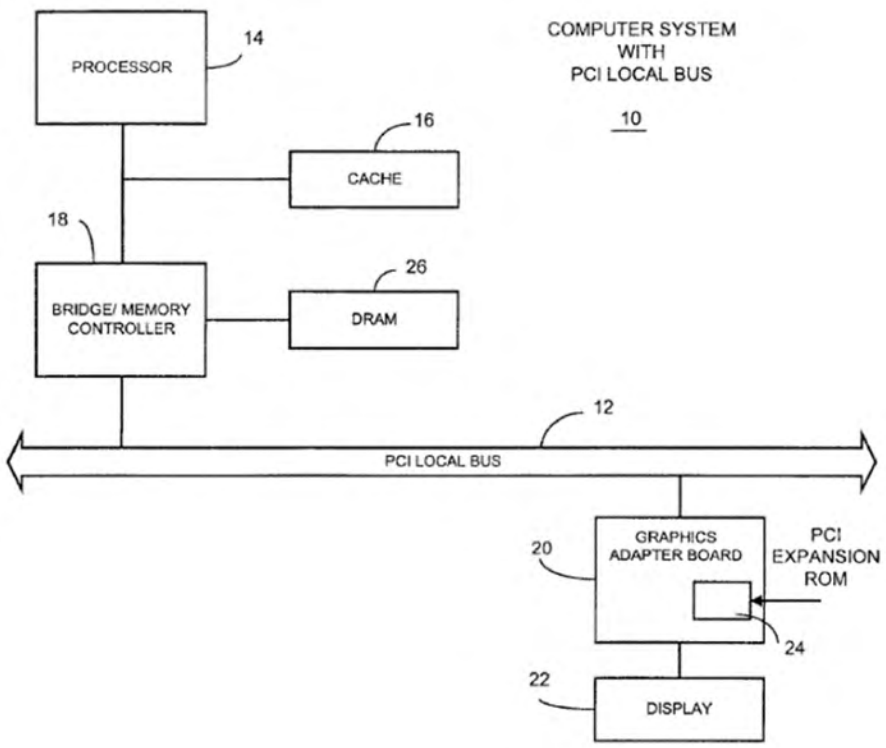
4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Rahman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Rahman, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.



Rahman, Fig. 1.

> "The processor accesses devices connected to the PCI local bus 12 through a bridge/memory controller 18."

Rahman, 2:47-48.

6. The method of claim 1, further comprising updating the list of boot data.	Rahman, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Rahman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> "The compression technique used may include both run-length encoding and pattern compression, and may operate at the bit level."

Rahman, Abstract.

> "Compression techniques using both run-length encoding and pattern compression may be used (e.g., the Ross Compression technique). The compression technique may scan an input file. First, it may determine if it can perform run-length-encoding compression on the current character. If not, the compression technique may determine if it can compress a pattern of characters. If the run-length-encoding and pattern matching Were not successful, the compression technique may copy the character directly to the compressed file."

Rahman, 2:7-16.

> "The compression and decompression techniques utilize four 3-byte formats. FIG. 3 illustrates the formats, which are a short run-length-encoded format 30, a long run-length encoded format 32, a short patterned format 34, and a long patterned format 36."

Rahman, 3:12-16. *See also* 3:17-17, Appendix.

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Rahman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rahman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> "The compression technique used may include both run-length encoding and pattern compression, and may operate at the bit level."

Rahman, Abstract.

> "Compression techniques using both run-length encoding and pattern compression may be used (e.g., the Ross Compression technique). The compression technique may scan an input file. First, it may determine if it can perform run-length-encoding compression on the current character. If not, the compression technique may determine if it can compress a pattern of characters. If the run-length-encoding and pattern matching Were not successful, the compression technique may copy the character directly to the compressed file."

Rahman, 2:7-16.

> "The compression and decompression techniques utilize four 3-byte formats. FIG. 3 illustrates the formats, which are a short run-length-encoded format 30, a long run-length encoded format 32, a short patterned format 34, and a long patterned format 36."

Rahman, 3:12-16. *See also* 3:17-17, Appendix.

11.1. a processor;	Rahman, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Rahman, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Rahman, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Rahman, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Rahman, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Rahman, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Rahman, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

Appendix B16
Invalidity of U.S. Patent 8,090,936 based on Rahman

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Rahman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B16
Invalidity of U.S. Patent 8,090,936 based on Rahman

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Rahman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rahman discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B17
Invalidity of U.S. Patent 8,090,936 based on Settsu

U.S. Patent No. 6,374,353 to Settsu ("Settsu") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

In addition, Apple incorporates by reference, as if set forth fully herein, all arguments related to Settsu in pending inter partes review petitions IPR2016-01737, IPR2016-01738, and IPR2016-01739

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Settsu, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

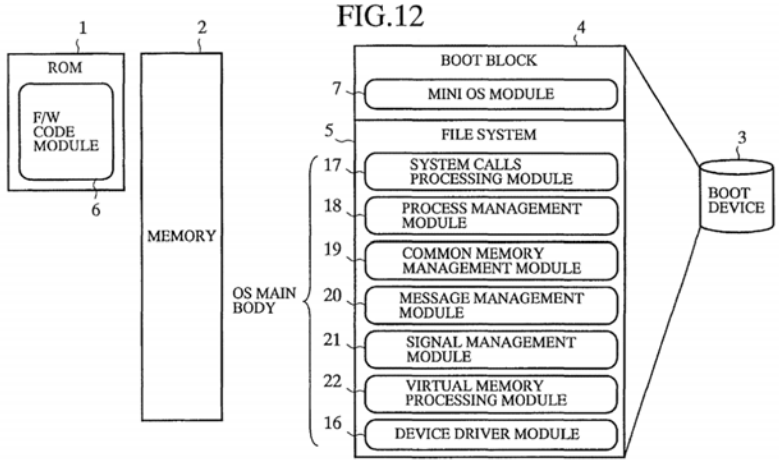
Settsu discloses this claim limitation:

> Referring next to FIG. 1, there is illustrated a block diagram showing the structure of an information processing apparatus according to a first embodiment of th present invention. In the figure, reference numeral 1 denotes a ROM of the information processing apparatus, 2 denotes a memory of the information processing apparatus, 3 denotes 5 a boot device of the information processing apparatus, 4 denotes a boot block in the boot device 3, 5 denotes a file system I the boot device 3, and 6 denotes a firmware or FI/W code module stored in the ROM 1. The FI/W code module 6 is directly executed on the ROM 1 so as to load data from the boot block 4 in the boot device 3 into the memory 2 and then assume that the loaded data is a code and execute the code after setting up and running diagnostic checks on a hardware or H/W register. Further, reference numeral 7 denotes a mini operating system (OS) module complied and liked in the same way as ordinary program files and located in the boot block 4 within the boot device, the mini OS module having OS functions required for bootstrap processing, and 8 denotes an OS main body module located in the file system 5 within the boot device and provided with OS functions except the OS functions included in the mini OS module 7. When the information processing apparatus is powered on, it goes through initialization and transfers control to the F/W code module 6 stored in the ROM 1.

Referring next to FIG. 2, there is illustrated a diagram showing the structure of the mini OS module 7 stored in the boot block of the boot device of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the mini OS module 7 consists of a mini kernel module 9 which is a basic part of the OS, a boot device driver module 10 for performing I/O operations on the boot device 3, and an OS loading and initialization processing module 11 for loading the OS main body module 8 from the boot device 3 into the memory 2 and for executing the initialization of the OS main body module 8.

Settsu, 7:65-8:35

FIG.12



Settsu, Fig. 12

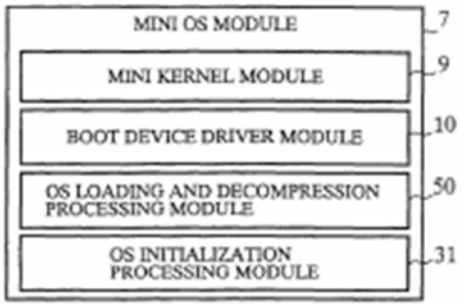
Referring next to FIG. 12, there is illustrated a block diagram showing the structure of an information processing apparatus according to a fourth embodiment of the present invention. In the figure, the same reference numerals as shown in FIG. 5 designate the same or like elements, and therefore the description of those elements will be omitted hereinafter. Like the OS of the second embodiment, the OS of the second embodiment is divided into a mini OS module 7 and a main body of the OS, and the main body is further divided into a plurality of functional modules, such as a system call processing module 17, a process management module 18, a common memory management module 19, a message management module 20, a signal management module 21, a virtual memory processing module 22, and a device driver module 16. The plurality of functional modules are separately stored as compressed files in a file system 5 of a boot device 3.

Settsu **Claim 1.1**

"maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

Settsu, 13:55-65

FIG.13



Settsu, Fig. 13.

> Referring next to FIG. 13, there is illustrated a block diagram showing the structure of the mini OS module 7 of the information processing apparatus according to the fourth embodiment of the present invention. Like the mini OS module 7 of the second embodiment as shown in FIG. 6, the mini OS module 7 of the fourth embodiment is provided with a mini kernel module 9, a boot device driver module 10, and an OS initialization processing module 31. The mini OS module 7 of the fourth embodiment further comprises an OS loading and decompression processing module 50 having a function of decompressing a loaded functional module in addition to the function of the OS load processing module 30 of the second embodiment, instead of the OS load processing module 30.

Settsu, 13:66-14:12

See also Settsu, 16:7-17:62, and Figs. 1-4, 6-9, 14, 20 & 35-36.

1.2 initializing a central processing unit of said computer system;	Settsu, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this claim limitation:

> A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

Settsu, Abstract

> The present invention relates to an information processing apparatus capable of reducing the time required for booting itself when it is powered on, and a method of booting an information processing apparatus at a high speed.

Settsu, 1:8-12.

> Referring next to FIG. 1, there is illustrated a block diagram showing the structure of an information processing apparatus according to a first embodiment of th present invention. In the figure, reference numeral 1 denotes a ROM of the information processing apparatus, 2 denotes a memory of the information processing apparatus, 3 denotes 5 a boot device of the information processing apparatus, 4 denotes a boot block in the boot device 3, 5 denotes a file system I the boot device 3, and 6 denotes a firmware or FI/W code module stored in the ROM 1. The FI/W code module 6 is directly executed on the ROM 1 so as to load data from the boot block 4 in the boot device 3 into the memory 2 and

then assume that the loaded data is a code and execute the code after setting up and running diagnostic checks on a hardware or H/W register. Further, reference numeral 7 denotes a mini operating system (OS) module complied and liked in the same way as ordinary program files and located in the boot block 4 within the boot device, the mini OS module having OS functions required for bootstrap processing, and 8 denotes an OS main body module located in the file system 5 within the boot device and provided with OS functions except the OS functions included in the mini OS module 7. When the information processing apparatus is powered on, it goes through initialization and transfers control to the F/W code module 6 stored in the ROM 1.

Referring next to FIG. 2, there is illustrated a diagram showing the structure of the mini OS module 7 stored in the boot block of the boot device of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the mini OS module 7 consists of a mini kernel module 9 which is a basic part of the OS, a boot device driver module 10 for performing I/O operations on the boot device 3, and an OS loading and initialization processing module 11 for loading the OS main body module 8 from the boot device 3 into the memory 2 and for executing the initialization of the OS main body module 8.

Settsu, 7:65-8:35

Referring next to FIG. 3, there is illustrated a diagram showing the structure of the OS main body module 8 of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the OS main body module 8 consists of a kernel module 15 having kernel functions except the functions included in the mini kernel module 9 of FIG. 2, and a device driver module 16 for performing I/O operations on devices (not shown) except the boot device 3. . . . The OS main body module 8 is located within the file system 5 of the boot device 3 and is loaded into the memory 2 by the mini OS module 7. The OS main body module 8 then goes through initialization.

Settsu, 8:47-9:3

Referring next to FIG. 4, there is illustrated a flow chart showing operations of the mini OS module 7 of the information processing apparatus according to the first embodiment of the present invention. The F/W code module 6 loads the mini OS module 7 into the memory 2 and transfers control to the mini OS module 7. The mini OS module 7 then, in step ST101, executes initialization of the mini kernel module 9.

> In this case, the thread management module 12, the I/O management module 13, and the thread communication management module 14 are initialized and their functions are available now. Next, the mini OS module 7, in step ST102, executes initialization of the boot device driver module 10. The boot device driver module 10 registers an interrupt request from the boot device into an interrupt table of the I/O management module 13 so as to handle the interrupt request, and generates and starts execution of a thread for boot device I/O processing by using the thread management module 12.

Settsu, 9:7-21.

> The mini OS module 7, in step ST103, generates a thread for the OS loading and initialization processing module 11 by using the thread management module 12. The mini OS module 7 further, in step ST104, starts execution of the thread by using the thread management module 12. The OS loading and initialization processing module 11 is thus started up as the thread. After that, the mini OS module 7 transfers control to the OS loading and initialization processing module 11.

Settsu, 9:30-39.

> As previously mentioned, in accordance with the second embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body of the OS. Further, the plurality of functional modules are separately stored in the file system 5. In addition, the OS load processing and the OS initialization processing can be performed in parallel with each other after any one of the plurality of functional modules of the OS main body is loaded into the memory. As a result, while the CPU waits for the occurrence of an event in performing the OS load or initialization processing, the CPU does not idle but the CPU performs another processing. Accordingly, the second embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 12:1-16.

> Referring next to FIG. 13, there is illustrated a block diagram showing the structure of the mini OS module 7 of the information processing apparatus according to the fourth embodiment of the present invention. Like the mini OS module 7 of the second embodiment as shown in FIG. 6, the mini OS module 7 of the fourth embodiment is provided with a mini kernel module 9, a boot device driver module 10, and an OS

initialization processing module 31. The mini OS module 7 of the fourth embodiment further comprises an OS loading and decompression processing module 50 having a function of decompressing a loaded functional module in addition to the function of the OS load processing module 30 of the second embodiment, instead of the OS load processing module 30.

Settsu, 13:66-14:12

See also Figs. 1-4, 6-9, 12-14, 20 & 35-36.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Settsu, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this claim limitation:

> A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

Settsu, Abstract

> The present invention relates to an information processing apparatus capable of reducing the time required for booting itself when it is powered on, and a method of booting an information processing apparatus at a high speed.

Settsu, 1:8-12.

> In accordance with an aspect of the present invention, there is provided an information processing apparatus comprising: a boot device divided into a boot block in which a mini operating system (OS) module having a function required for bootstrap processing is located and a file system

> in which an operating system (OS) main body module having functions other than the function of bootstrap. . . .

Settsu, 1:51-57

> In accordance with another preferred embodiment of the present invention, the plurality of functional modules, into which the OS main body module is divided, are stored as compressed files in the file system and the loading and initialization processing module of the mini OS module is divided into an OS loading and decompression processing module and an OS initialization module. Further, the mini OS module generates and starts execution of a thread for the OS loading and decompression processing module after the mini OS module initializes the mini kernel module and the boot device driver module. After the thread for the OS loading and decompression processing module is started, the OS loading and decompression processing module loads each of the plurality of functional modules into the memory and decompresses the loaded functional module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is executed, the OS initialization module initializes each of the plurality of functional modules loaded into the memory and decompressed.

Settsu, 3:6-25

> Referring next to FIG. 1, there is illustrated a block diagram showing the structure of an information processing apparatus according to a first embodiment of th present invention. In the figure, reference numeral 1 denotes a ROM of the information processing apparatus, 2 denotes a memory of the information processing apparatus, 3 denotes 5 a boot device of the information processing apparatus, 4 denotes a boot block in the boot device 3, 5 denotes a file system I the boot device 3, and 6 denotes a firmware or FI/W code module stored in the ROM 1. The FI/W code module 6 is directly executed on the ROM 1 so as to load data from the boot block 4 in the boot device 3 into the memory 2 and then assume that the loaded data is a code and execute the code after setting up and running diagnostic checks on a hardware or H/W register. Further, reference numeral 7 denotes a mini operating system (OS) module complied and liked in the same way as ordinary program files and located in the boot block 4 within the boot device, the mini OS module having OS functions required for bootstrap processing, and 8 denotes an OS main body module located in the file system 5 within the boot device and provided with OS functions except the OS functions included in the mini OS module 7. When the information processing

apparatus is powered on, it goes through initialization and transfers control to the F/W code module 6 stored in the ROM 1.

Referring next to FIG. 2, there is illustrated a diagram showing the structure of the mini OS module 7 stored in the boot block of the boot device of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the mini OS module 7 consists of a mini kernel module 9 which is a basic part of the OS, a boot device driver module 10 for performing I/O operations on the boot device 3, and an OS loading and initialization processing module 11 for loading the OS main body module 8 from the boot device 3 into the memory 2 and for executing the initialization of the OS main body module 8.

Settsu, 7:65-8:35

Referring next to FIG. 3, there is illustrated a diagram showing the structure of the OS main body module 8 of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the OS main body module 8 consists of a kernel module 15 having kernel functions except the functions included in the mini kernel module 9 of FIG. 2, and a device driver module 16 for performing I/O operations on devices (not shown) except the boot device 3. . . . The OS main body module 8 is located within the file system 5 of the boot device 3 and is loaded into the memory 2 by the mini OS module 7. The OS main body module 8 then goes through initialization.

Settsu, 8:47-9:3

Referring next to FIG. 4, there is illustrated a flow chart showing operations of the mini OS module 7 of the information processing apparatus according to the first embodiment of the present invention. The F/W code module 6 loads the mini OS module 7 into the memory 2 and transfers control to the mini OS module 7. The mini OS module 7 then, in step ST101, executes initialization of the mini kernel module 9. In this case, the thread management module 12, the I/O management module 13, and the thread communication management module 14 are initialized and their functions are available now. Next, the mini OS module 7, in step ST102, executes initialization of the boot device driver module 10. The boot device driver module 10 registers an interrupt request from the boot device into an interrupt table of the I/O management module 13 so as to handle the interrupt request, and generates and starts execution of a thread for boot device I/O processing by using the thread management module 12.

Settsu, 9:7-21.

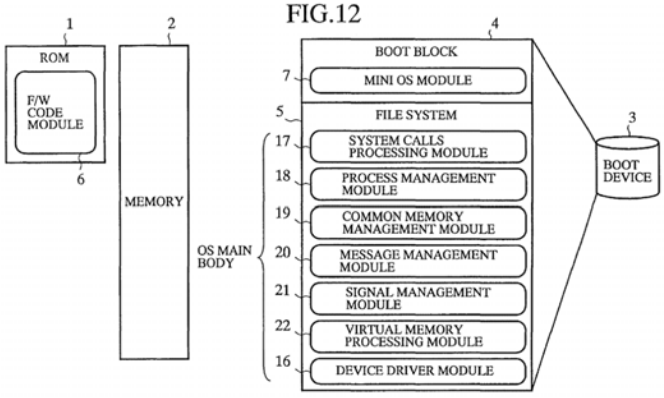
> The mini OS module 7, in step ST103, generates a thread for the OS loading and initialization processing module 11 by using the thread management module 12. The mini OS module 7 further, in step ST104, starts execution of the thread by using the thread management module 12. The OS loading and initialization processing module 11 is thus started up as the thread. After that, the mini OS module 7 transfers control to the OS loading and initialization processing module 11.

Settsu, 9:30-39.

> As previously mentioned, in accordance with the second embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body of the OS. Further, the plurality of functional modules are separately stored in the file system 5. In addition, the OS load processing and the OS initialization processing can be performed in parallel with each other after any one of the plurality of functional modules of the OS main body is loaded into the memory. As a result, while the CPU waits for the occurrence of an event in performing the OS load or initialization processing, the CPU does not idle but the CPU performs another processing. Accordingly, the second embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 12:1-16.

FIG.12



Settsu, Fig. 12

Referring next to FIG. 12, there is illustrated a block diagram showing the structure of an information processing apparatus according to a fourth embodiment of the present invention. In the figure, the same reference numerals as shown in FIG. 5 designate the same or like elements, and therefore the description of those elements will be omitted hereinafter. Like the OS of the second embodiment, the OS of the second embodiment is divided into a mini OS module 7 and a main body of the OS, and the main body is further divided into a plurality of functional modules, such as a system call processing module 17, a process management module 18, a common memory management module 19, a message management module 20, a signal management module 21, a virtual memory processing module 22, and a device driver module 16. The plurality of functional modules are separately stored as compressed files in a file system 5 of a boot device 3.

Settsu, 13:55-65

Referring next to FIG. 13, there is illustrated a block diagram showing the structure of the mini OS module 7 of the information processing apparatus according to the fourth embodiment of the present invention. Like the mini OS module 7 of the second embodiment as shown in FIG. 6, the mini OS module 7 of the fourth embodiment is provided with a mini kernel module 9, a boot device driver module 10, and an OS initialization processing module 31. The mini OS module 7 of the fourth embodiment further comprises an OS loading and decompression processing module 50 having a function of decompressing a loaded functional module in addition to the function of the OS load processing module 30 of the second embodiment, instead of the OS load processing module 30.

Settsu, 13:66-14:12

Referring next to FIG. 14, there is illustrated a flow chart showing operations of the OS loading and decompression processing module 50 of the information processing apparatus according to the fourth embodiment of the present invention. The OS loading and decompression processing module 50 to which control from the mini OS module 7 has been transferred, in step ST181, loads the main body of the OS stored in the file system 5 of the boot device 3, such as the system call processing module 17, the process management module 18, the common memory management module 19, the message management module 20, the signal management module 21, the virtual memory processing module 22, and the device driver module 16, into the memory 2. In performing step ST181, the OS loading and decompression processing module 50 loads any one of those functional

> modules 16 to 22 first. The OS loading and decompression processing module 50 then, in step ST182, decompresses one functional module loaded into the memory. Since the plurality of functional modules 16 to 22 are stored as compresses files in the file system 5 of the boot device, the functional module loaded into the memory 2 is compressed data. Therefore, in performing step ST182, the OS loading and decompression processing module 50 decompresses the compressed data so as to convert it into executable code and data.

Settsu, 14:13-37

> As previously mentioned, in accordance with the fourth embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body, and the plurality of functional modules are stored as compressed files in the file system **5** of the boot device. Further, the OS loading and decompression processing module **50** decompresses each of the plurality of functional modules each time it loads each of them into memory. As a result, the time required for I/O processing can be reduced. Accordingly, the fourth embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 14:58-15:5

See also Settsu, 8:21-35, 8:66-9:11, 11:7-9, 11:18-39, 13:49-15:5, 16:7-17:62, and Figs. 1-4, 6-9, 13-14, 20 & 35-36.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Settsu, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this claim limitation:

> A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

Settsu, Abstract

> The present invention relates to an information processing apparatus capable of reducing the time required for booting itself when it is powered on, and a method of booting an information processing apparatus at a high speed.

Settsu, 1:8-12.

> In accordance with an aspect of the present invention, there is provided an information processing apparatus comprising: a boot device divided into a boot block in which a mini operating system (OS) module having a function required for bootstrap processing is located and a file system

> in which an operating system (OS) main body module having functions other than the function of bootstrap. . . .

Settsu, 1:51-57

> In accordance with another preferred embodiment of the present invention, the plurality of functional modules, into which the OS main body module is divided, are stored as compressed files in the file system and the loading and initialization processing module of the mini OS module is divided into an OS loading and decompression processing module and an OS initialization module. Further, the mini OS module generates and starts execution of a thread for the OS loading and decompression processing module after the mini OS module initializes the mini kernel module and the boot device driver module. After the thread for the OS loading and decompression processing module is started, the OS loading and decompression processing module loads each of the plurality of functional modules into the memory and decompresses the loaded functional module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is executed, the OS initialization module initializes each of the plurality of functional modules loaded into the memory and decompressed.

Settsu, 3:6-25

> Referring next to FIG. 1, there is illustrated a block diagram showing the structure of an information processing apparatus according to a first embodiment of th present invention. In the figure, reference numeral 1 denotes a ROM of the information processing apparatus, 2 denotes a memory of the information processing apparatus, 3 denotes 5 a boot device of the information processing apparatus, 4 denotes a boot block in the boot device 3, 5 denotes a file system I the boot device 3, and 6 denotes a firmware or FI/W code module stored in the ROM 1. The FI/W code module 6 is directly executed on the ROM 1 so as to load data from the boot block 4 in the boot device 3 into the memory 2 and then assume that the loaded data is a code and execute the code after setting up and running diagnostic checks on a hardware or H/W register. Further, reference numeral 7 denotes a mini operating system (OS) module complied and liked in the same way as ordinary program files and located in the boot block 4 within the boot device, the mini OS module having OS functions required for bootstrap processing, and 8 denotes an OS main body module located in the file system 5 within the boot device and provided with OS functions except the OS functions included in the mini OS module 7. When the information processing

apparatus is powered on, it goes through initialization and transfers control to the F/W code module 6 stored in the ROM 1.

Referring next to FIG. 2, there is illustrated a diagram showing the structure of the mini OS module 7 stored in the boot block of the boot device of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the mini OS module 7 consists of a mini kernel module 9 which is a basic part of the OS, a boot device driver module 10 for performing I/O operations on the boot device 3, and an OS loading and initialization processing module 11 for loading the OS main body module 8 from the boot device 3 into the memory 2 and for executing the initialization of the OS main body module 8.

Settsu, 7:65-8:35

Referring next to FIG. 3, there is illustrated a diagram showing the structure of the OS main body module 8 of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the OS main body module 8 consists of a kernel module 15 having kernel functions except the functions included in the mini kernel module 9 of FIG. 2, and a device driver module 16 for performing I/O operations on devices (not shown) except the boot device 3. . . . The OS main body module 8 is located within the file system 5 of the boot device 3 and is loaded into the memory 2 by the mini OS module 7. The OS main body module 8 then goes through initialization.

Settsu, 8:47-9:3

Referring next to FIG. 4, there is illustrated a flow chart showing operations of the mini OS module 7 of the information processing apparatus according to the first embodiment of the present invention. The F/W code module 6 loads the mini OS module 7 into the memory 2 and transfers control to the mini OS module 7. The mini OS module 7 then, in step ST101, executes initialization of the mini kernel module 9. In this case, the thread management module 12, the I/O management module 13, and the thread communication management module 14 are initialized and their functions are available now. Next, the mini OS module 7, in step ST102, executes initialization of the boot device driver module 10. The boot device driver module 10 registers an interrupt request from the boot device into an interrupt table of the I/O management module 13 so as to handle the interrupt request, and generates and starts execution of a thread for boot device I/O processing by using the thread management module 12.

Settsu, 9:7-21.

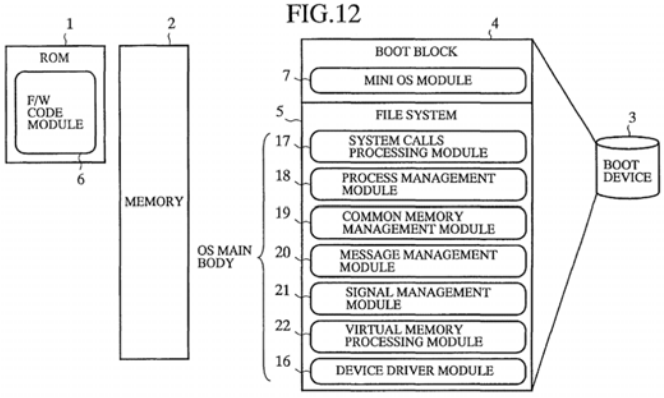
> The mini OS module 7, in step ST103, generates a thread for the OS loading and initialization processing module 11 by using the thread management module 12. The mini OS module 7 further, in step ST104, starts execution of the thread by using the thread management module 12. The OS loading and initialization processing module 11 is thus started up as the thread. After that, the mini OS module 7 transfers control to the OS loading and initialization processing module 11.

Settsu, 9:30-39.

> As previously mentioned, in accordance with the second embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body of the OS. Further, the plurality of functional modules are separately stored in the file system 5. In addition, the OS load processing and the OS initialization processing can be performed in parallel with each other after any one of the plurality of functional modules of the OS main body is loaded into the memory. As a result, while the CPU waits for the occurrence of an event in performing the OS load or initialization processing, the CPU does not idle but the CPU performs another processing. Accordingly, the second embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 12:1-16.

FIG.12



Settsu, Fig. 12

Referring next to FIG. 12, there is illustrated a block diagram showing the structure of an information processing apparatus according to a fourth embodiment of the present invention. In the figure, the same reference numerals as shown in FIG. 5 designate the same or like elements, and therefore the description of those elements will be omitted hereinafter. Like the OS of the second embodiment, the OS of the second embodiment is divided into a mini OS module 7 and a main body of the OS, and the main body is further divided into a plurality of functional modules, such as a system call processing module 17, a process management module 18, a common memory management module 19, a message management module 20, a signal management module 21, a virtual memory processing module 22, and a device driver module 16. The plurality of functional modules are separately stored as compressed files in a file system 5 of a boot device 3.

Settsu, 13:55-65

Referring next to FIG. 13, there is illustrated a block diagram showing the structure of the mini OS module 7 of the information processing apparatus according to the fourth embodiment of the present invention. Like the mini OS module 7 of the second embodiment as shown in FIG. 6, the mini OS module 7 of the fourth embodiment is provided with a mini kernel module 9, a boot device driver module 10, and an OS initialization processing module 31. The mini OS module 7 of the fourth embodiment further comprises an OS loading and decompression processing module 50 having a function of decompressing a loaded functional module in addition to the function of the OS load processing module 30 of the second embodiment, instead of the OS load processing module 30.

Settsu, 13:66-14:12

Referring next to FIG. 14, there is illustrated a flow chart showing operations of the OS loading and decompression processing module 50 of the information processing apparatus according to the fourth embodiment of the present invention. The OS loading and decompression processing module 50 to which control from the mini OS module 7 has been transferred, in step ST181, loads the main body of the OS stored in the file system 5 of the boot device 3, such as the system call processing module 17, the process management module 18, the common memory management module 19, the message management module 20, the signal management module 21, the virtual memory processing module 22, and the device driver module 16, into the memory 2. In performing step ST181, the OS loading and decompression processing module 50 loads any one of those functional

> modules 16 to 22 first. The OS loading and decompression processing module 50 then, in step ST182, decompresses one functional module loaded into the memory. Since the plurality of functional modules 16 to 22 are stored as compresses files in the file system 5 of the boot device, the functional module loaded into the memory 2 is compressed data. Therefore, in performing step ST182, the OS loading and decompression processing module 50 decompresses the compressed data so as to convert it into executable code and data.

Settsu, 14:13-37

> As previously mentioned, in accordance with the fourth embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body, and the plurality of functional modules are stored as compressed files in the file system **5** of the boot device. Further, the OS loading and decompression processing module **50** decompresses each of the plurality of functional modules each time it loads each of them into memory. As a result, the time required for I/O processing can be reduced. Accordingly, the fourth embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 14:58-15:5

See also Settsu, 8:21-35, 8:66-9:11, 11:7-9, 11:18-39, 13:49-15:5, 16:7-17:62, and Figs. 1-4, 6-9, 13-14, 20 & 35-36.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Settsu, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this claim limitation:

> A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

Settsu, Abstract

> The present invention relates to an information processing apparatus capable of reducing the time required for booting itself when it is powered on, and a method of booting an information processing apparatus at a high speed.

Settsu, 1:8-12.

> In accordance with an aspect of the present invention, there is provided an information processing apparatus comprising: a boot device divided into a boot block in which a mini operating system (OS) module having a function required for bootstrap processing is located and a file system

> in which an operating system (OS) main body module having functions other than the function of bootstrap. . . .

Settsu, 1:51-57

> In accordance with another preferred embodiment of the present invention, the plurality of functional modules, into which the OS main body module is divided, are stored as compressed files in the file system and the loading and initialization processing module of the mini OS module is divided into an OS loading and decompression processing module and an OS initialization module. Further, the mini OS module generates and starts execution of a thread for the OS loading and decompression processing module after the mini OS module initializes the mini kernel module and the boot device driver module. After the thread for the OS loading and decompression processing module is started, the OS loading and decompression processing module loads each of the plurality of functional modules into the memory and decompresses the loaded functional module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is executed, the OS initialization module initializes each of the plurality of functional modules loaded into the memory and decompressed.

Settsu, 3:6-25

> Referring next to FIG. 1, there is illustrated a block diagram showing the structure of an information processing apparatus according to a first embodiment of th present invention. In the figure, reference numeral 1 denotes a ROM of the information processing apparatus, 2 denotes a memory of the information processing apparatus, 3 denotes 5 a boot device of the information processing apparatus, 4 denotes a boot block in the boot device 3, 5 denotes a file system I the boot device 3, and 6 denotes a firmware or FI/W code module stored in the ROM 1. The FI/W code module 6 is directly executed on the ROM 1 so as to load data from the boot block 4 in the boot device 3 into the memory 2 and then assume that the loaded data is a code and execute the code after setting up and running diagnostic checks on a hardware or H/W register. Further, reference numeral 7 denotes a mini operating system (OS) module complied and liked in the same way as ordinary program files and located in the boot block 4 within the boot device, the mini OS module having OS functions required for bootstrap processing, and 8 denotes an OS main body module located in the file system 5 within the boot device and provided with OS functions except the OS functions included in the mini OS module 7. When the information processing

apparatus is powered on, it goes through initialization and transfers control to the F/W code module 6 stored in the ROM 1.

Referring next to FIG. 2, there is illustrated a diagram showing the structure of the mini OS module 7 stored in the boot block of the boot device of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the mini OS module 7 consists of a mini kernel module 9 which is a basic part of the OS, a boot device driver module 10 for performing I/O operations on the boot device 3, and an OS loading and initialization processing module 11 for loading the OS main body module 8 from the boot device 3 into the memory 2 and for executing the initialization of the OS main body module 8.

Settsu, 7:65-8:35

Referring next to FIG. 3, there is illustrated a diagram showing the structure of the OS main body module 8 of the information processing apparatus according to the first embodiment of the present invention. As shown in the figure, the OS main body module 8 consists of a kernel module 15 having kernel functions except the functions included in the mini kernel module 9 of FIG. 2, and a device driver module 16 for performing I/O operations on devices (not shown) except the boot device 3. . . . The OS main body module 8 is located within the file system 5 of the boot device 3 and is loaded into the memory 2 by the mini OS module 7. The OS main body module 8 then goes through initialization.

Settsu, 8:47-9:3

Referring next to FIG. 4, there is illustrated a flow chart showing operations of the mini OS module 7 of the information processing apparatus according to the first embodiment of the present invention. The F/W code module 6 loads the mini OS module 7 into the memory 2 and transfers control to the mini OS module 7. The mini OS module 7 then, in step ST101, executes initialization of the mini kernel module 9. In this case, the thread management module 12, the I/O management module 13, and the thread communication management module 14 are initialized and their functions are available now. Next, the mini OS module 7, in step ST102, executes initialization of the boot device driver module 10. The boot device driver module 10 registers an interrupt request from the boot device into an interrupt table of the I/O management module 13 so as to handle the interrupt request, and generates and starts execution of a thread for boot device I/O processing by using the thread management module 12.

Settsu Claim 1.5

"utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

Settsu, 9:7-21.

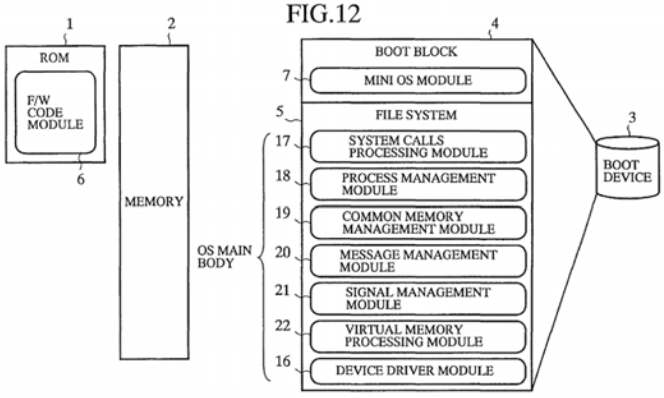
> The mini OS module 7, in step ST103, generates a thread for the OS loading and initialization processing module 11 by using the thread management module 12. The mini OS module 7 further, in step ST104, starts execution of the thread by using the thread management module 12. The OS loading and initialization processing module 11 is thus started up as the thread. After that, the mini OS module 7 transfers control to the OS loading and initialization processing module 11.

Settsu, 9:30-39.

> As previously mentioned, in accordance with the second embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body of the OS. Further, the plurality of functional modules are separately stored in the file system 5. In addition, the OS load processing and the OS initialization processing can be performed in parallel with each other after any one of the plurality of functional modules of the OS main body is loaded into the memory. As a result, while the CPU waits for the occurrence of an event in performing the OS load or initialization processing, the CPU does not idle but the CPU performs another processing. Accordingly, the second embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 12:1-16.

FIG.12

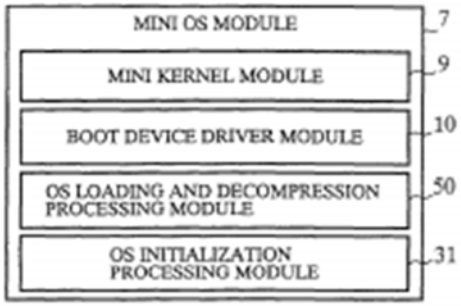


Settsu, Fig. 12

Referring next to FIG. 12, there is illustrated a block diagram showing the structure of an information processing apparatus according to a fourth embodiment of the present invention. In the figure, the same reference numerals as shown in FIG. 5 designate the same or like elements, and therefore the description of those elements will be omitted hereinafter. Like the OS of the second embodiment, the OS of the second embodiment is divided into a mini OS module 7 and a main body of the OS, and the main body is further divided into a plurality of functional modules, such as a system call processing module 17, a process management module 18, a common memory management module 19, a message management module 20, a signal management module 21, a virtual memory processing module 22, and a device driver module 16. The plurality of functional modules are separately stored as compressed files in a file system 5 of a boot device 3.

Settsu, 13:55-65

FIG.13



Settsu, Fig. 13.

Referring next to FIG. 13, there is illustrated a block diagram showing the structure of the mini OS module 7 of the information processing apparatus according to the fourth embodiment of the present invention. Like the mini OS module 7 of the second embodiment as shown in FIG. 6, the mini OS module 7 of the fourth embodiment is provided with a mini kernel module 9, a boot device driver module 10, and an OS initialization processing module 31. The mini OS module 7 of the fourth embodiment further comprises an OS loading and decompression processing module 50 having a function of decompressing a loaded functional module in addition to the function of the OS load processing module 30 of the second embodiment, instead of the OS load processing module 30.

Settsu, 13:66-14:12

Settsu Claim 1.5

"utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

> Referring next to FIG. 14, there is illustrated a flow chart showing operations of the OS loading and decompression processing module 50 of the information processing apparatus according to the fourth embodiment of the present invention. The OS loading and decompression processing module 50 to which control from the mini OS module 7 has been transferred, in step ST181, loads the main body of the OS stored in the file system 5 of the boot device 3, such as the system call processing module 17, the process management module 18, the common memory management module 19, the message management module 20, the signal management module 21, the virtual memory processing module 22, and the device driver module 16, into the memory 2. In performing step ST181, the OS loading and decompression processing module 50 loads any one of those functional modules 16 to 22 first. The OS loading and decompression processing module 50 then, in step ST182, decompresses one functional module loaded into the memory. Since the plurality of functional modules 16 to 22 are stored as compresses files in the file system 5 of the boot device, the functional module loaded into the memory 2 is compressed data. Therefore, in performing step ST182, the OS loading and decompression processing module 50 decompresses the compressed data so as to convert it into executable code and data.

Settsu, 14:13-37

> As previously mentioned, in accordance with the fourth embodiment of the present invention, the main body of the OS is divided into a plurality of functional modules according a plurality of functions to be performed by the main body, and the plurality of functional modules are stored as compressed files in the file system **5** of the boot device. Further, the OS loading and decompression processing module **50** decompresses each of the plurality of functional modules each time it loads each of them into memory. As a result, the time required for I/O processing can be reduced. Accordingly, the fourth embodiment of the present invention provides an advantage of being able to further reduce the time required for booting up the information processing apparatus.

Settsu, 14:58-15:5

See also Settsu, 8:21-35, 8:66-9:11, 11:7-9, 11:18-39, 13:49-15:5, 16:7-17:62, and Figs. 1-4, 6-9, 14, 20 & 35-36.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Settsu, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Settsu, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The present invention is made to overcome the above problems. It is therefore an object of the present invention to provide an information processing apparatus and a method capable of reducing the time required for booting up itself when it is powered on, and also reducing the time required to start execution of applications to be started automatically when the information processing apparatus is booted up.

Settsu, 1:44-50

> In accordance with another preferred embodiment of the present invention, the OS loading processing module of the mini OS module is an application (AP) execution and OS loading processing module for starting execution of at least a predetermined application module which is located in the file system and which can automatically be started and run on the operating system when booting up the information processing apparatus, and for loading each of the plurality of functional modules into the memory. Further, the predetermined application module includes a function definition file in which some functional modules required for the application module to run on the operating system are listed. After the mini OS module is loaded into the memory, the mini OS module initializes the mini kernel module and the boot device driver module and then generates and starts execution of a thread for the AP execution and OS loading processing module. After the thread for the AP execution and OS loading processing module is started, the AP execution and OS loading processing module loads the application

nodule from the file system into the memory and further loads some functional modules required for the application module into the memory according to the function definition file included in the application module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is started, the OS initialization module then initializes each of the some functional modules loaded into the memory. And, after the initialization of all of the some functional modules is completed, the application execution and OS loading processing module further loads the remainder of all functional modules included in the OS main body module into the memory and initializes the remainder using the OS initialization processing module while starting execution of the application module as a process.

Settsu, 3:48-4:14

Settsu, Fig. 18.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Settsu, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The present invention is made to overcome the above problems. It is therefore an object of the present invention to provide an information processing apparatus and a method capable of reducing the time required for booting up itself when it is powered on, and also reducing the time required to start execution of applications to be started automatically when the information processing apparatus is booted up.

Settsu, 1:44-50

> In accordance with another preferred embodiment of the present invention, the OS loading processing module of the mini OS module is an application (AP) execution and OS loading processing module for starting execution of at least a predetermined application module which is located in the file system and which can automatically be started and run on the operating system when booting up the information processing apparatus, and for loading each of the plurality of functional modules into the memory. Further, the predetermined application module includes a function definition file in which some functional modules required for the application module to run on the operating system are listed. After the mini OS module is loaded into the memory, the mini OS module initializes the mini kernel module and the boot device driver module and then generates and starts execution of a thread for the AP execution and OS loading processing module. After the thread for the AP execution and OS loading processing module is started, the AP

execution and OS loading processing module loads the application nodule from the file system into the memory and further loads some functional modules required for the application module into the memory according to the function definition file included in the application module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is started, the OS initialization module then initializes each of the some functional modules loaded into the memory. And, after the initialization of all of the some functional modules is completed, the application execution and OS loading processing module further loads the remainder of all functional modules included in the OS main body module into the memory and initializes the remainder using the OS initialization processing module while starting execution of the application module as a process.

Settsu, 3:48-4:14

Settsu, Fig. 18.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Settsu, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Settsu, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

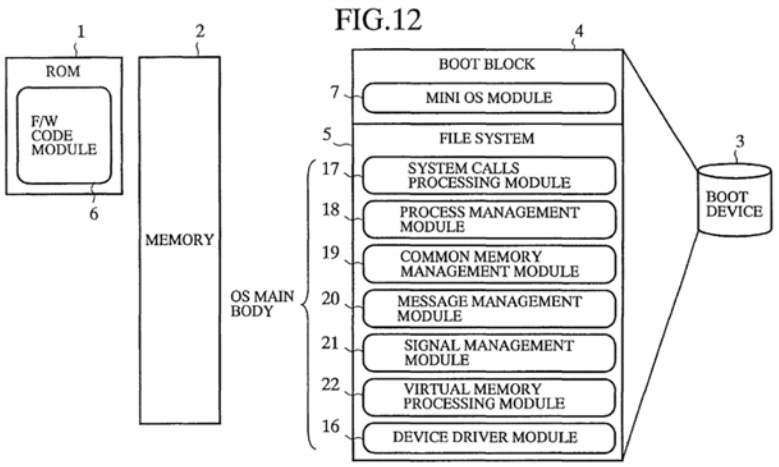
See also

> A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

Settsu, Abstract

> In accordance with another preferred embodiment of the present invention, the mini OS module further includes an address resolve table used for linking the mini OS module with the OS main body module. Further, after the mini OS module generates and starts execution of a thread for the OS loading and initialization processing module, the OS loading and initialization processing module loads the OS main body module into the memory and then initializes it, loads a first process to be executed first, into the memory, loads code portions of the mini kernel module and the boot device driver module into the memory, and writes addresses of the code portions loaded into the memory into the address resolve table.

Settsu, 5:39-51



Settsu, Fig. 12

> Referring next to FIG. 12, there is illustrated a block diagram showing the structure of an information processing apparatus according to a fourth embodiment of the present invention. In the figure, the same reference numerals as shown in FIG. 5 designate the same or like elements, and therefore the description of those elements will be omitted hereinafter. Like the OS of the second embodiment, the OS of the second embodiment is divided into a mini OS module 7 and a main body of the OS, and the main body is further divided into a plurality of functional modules, such as a system call processing module 17, a process management module 18, a common memory management module 19, a message management module 20, a signal management module 21, a virtual memory processing module 22, and a device driver module 16. The plurality of functional modules are separately stored as compressed files in a file system 5 of a boot device 3.

Settsu, 13:55-65

> The OS loading and decompression processing module 50 then, in step ST185, checks whether or not the whole of the main body of the OS has been loaded, that is, whether or not all the functional modules 16 to 22 have been loaded into the memory 2. If all the functional modules 16 to 22 have not been loaded into the memory 2 yet, the OS loading and decompression processing module 50 returns to step ST181 in which it continues to load the remaining functional modules of the OS main body.

Settsu, 14:44-52

See also Settsu, 16:7-17:62 and Figs. 1-4, 6-9, 13-14, 20 & 35-36.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Settsu, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> In accordance with another preferred embodiment of the present invention, the plurality of functional modules, into which the OS main body module is divided, are stored as compressed files in the file system and the loading and initialization processing module of the mini OS module is divided into an OS loading and decompression processing module and an OS initialization module. Further, the mini OS module generates and starts execution of a thread for the OS loading and decompression processing module after the mini OS module initializes the mini kernel module and the boot device driver module. After the thread for the OS loading and decompression processing module is started, the OS loading and decompression processing module loads each of the plurality of functional modules into the memory and decompresses the loaded functional module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is executed, the OS initialization module initializes each of the plurality of functional modules loaded into the memory and decompressed.

Settsu, 3:6-25

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Settsu, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Settsu discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> In accordance with another preferred embodiment of the present invention, the plurality of functional modules, into which the OS main body module is divided, are stored as compressed files in the file system and the loading and initialization processing module of the mini OS module is divided into an OS loading and decompression processing module and an OS initialization module. Further, the mini OS module generates and starts execution of a thread for the OS loading and decompression processing module after the mini OS module initializes the mini kernel module and the boot device driver module. After the thread for the OS loading and decompression processing module is started, the OS loading and decompression processing module loads each of the plurality of functional modules into the memory and decompresses the loaded functional module, and then generates and starts execution of a thread for the OS initialization module. After the thread for the OS initialization module is executed, the OS initialization module initializes each of the plurality of functional modules loaded into the memory and decompressed.

Settsu, 3:6-25

11.1. a processor;	Settsu, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Settsu, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Settsu, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Settsu, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Settsu, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Settsu, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Settsu, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Settsu, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B17
Invalidity of U.S. Patent 8,090,936 based on Settsu

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Settsu, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Settsu discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B18
Invalidity of U.S. Patent 8,090,936 based on Shinjo

U.S. Patent No. 5,269,022 to Shinjo ("Shinjo") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Shinjo, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

> The present invention relates to a method and an apparatus for booting a computer system.

Shinjo, 1:9-10

> In a computer system, generally, whenever the system is booted, a boot process for loading firmware, an initial program loader (IPL) program and an initialize (INZ) program, an initial program loader process, and an initialization process are executed.

Shinjo, 1:11-16

> According to one aspect of the present invention, there is provided a method for booting a computer system, the method comprising the steps of: setting a boot mode for booting the computer system; determining whether or not the set boot mode is a normal mode; determining whether or not a flag is set when the boot mode is the normal mode, the flag representing whether or not backup data is restorable; storing main memory data to be stored in a main memory immediately after the computer system is booted, as the backup data, into a backup memory when the flag is reset; and restoring the backup data stored in the backup memory, as the main memory data, into the main memory when the flag is set.

Shinjo, 1:35-48

> According to another aspect of the present invention, there is provided an apparatus for booting a computer system, the apparatus comprising: a main

memory for storing main memory data; means for setting a boot mode for booting the computer system; determining means for determining whether or not the boot mode is a normal mode; a flag to be set/reset in accordance with a determination result by the determining means; and a backup memory for storing the main memory data to be stored in the main memory immediately after the computer system is booted, as backup data when the boot mode is the normal mode and the flag is reset, and wherein the backup data stored in the backup memory is restored as the main memory data into the main memory when the boot mode is the normal mode and the flag is set.

Shinjo, 1:49-64

The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-68

When a boot command is output from the boot mode setting unit 17 to the CPU 11, an operation using the initial program loader (IPL) program is started by the CPU 11.

Shinjo, 3:4-7

In step S1, it is determined whether or not the boot mode designated by the boot command output from the boot mode setting unit 17 is the maintenance mode. If the boot mode is not the maintenance mode, i.e., if it is the normal mode, it is determined whether or not the backup flag 15 of the backup memory 13 is set (step S2). That is, it is determined whether a boot process is executed in the quick start mode or saving mode.

Shinjo, 3:8-15

In step S2, when the backup flag 15 is reset, it is determined that the backup data stored in the backup memory 13 cannot be restored. Since this determination is made in the first boot process, the same boot process as a conventional one is executed. In other words, the operating system (OS) initialization process and the application initialization process are executed by the initialization program (steps S4 and S5).

Shinjo, 3:16-23

> Since the backup flag 15 has been set, the boot process is started in the quick start mode through steps S1 and S2 after the reboot. That is, in step S6, the backup data stored in the backup memory 13 is restored into the main memory 12 as the main memory data. Since the computer system is thus restored to the state immediately after the boot process and the running environment is set, the boot process of the computer system can be completed without executing the OS initialization process of step S3 or the application initialization process of step S4.

Shinjo, 3:35-45

> In the computer system according to the first embodiment wherein the main memory data stored in the main memory 12 is saved into the backup memory 13 as the backup data, when the system power source is turned off, the backup data is erased. It is thus necessary to boot the system in the same manner as the conventional apparatus and save the main memory data stored in the main memory 12 into the backup memory 13 as the backup data immediately after the system is booted. By backing up the backup memory 13 by a battery or the like, such boot process in the system is executed only at the first time.

Shinjo, 3:65-4:8

> As described above, according to the present invention, when the system is first booted, the saving mode is selected. Immediately after the system is booted, the main memory data stored in the main memory is saved as backup data into the backup memory (backup file), and the backup flag is set. The system is then rebooted. Therefore, when the system is next booted in the normal mode, the quick start mode is selected and the backup data saved into the backup memory (backup file) is restored as the main memory data in the main memory. The boot process of the system is completed only by the above process, and the state immediately after the system is booted is restored.

Shinjo, 4:45-67

1.2 initializing a central processing unit of said computer system;	Shinjo, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

> The CPU 11 controls the entire computer system.

Shinjo, 2:32

> When a boot command is output from the boot mode setting unit 17 to the CPU 11, an operation using the initial program loader (IPL) program is started by the CPU 11.

Shijno, 3:4-7

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Shinjo, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

> In the initialization process, a resident load module is loaded into a main memory, various control blocks are produced, the running environment of an operating system (OS) is set, and the running environment of an application system is set (for example, a control process is produced).

Shinjo, 1:16-21

> According to one aspect of the present invention, there is provided a method for booting a computer system, the method comprising the steps of: setting a boot mode for booting the computer system; determining whether or not the set boot mode is a normal mode; determining whether or not a flag is set when the boot mode is the normal mode, the flag representing whether or not backup data is restorable; storing main memory data to be stored in a main memory immediately after the computer system is booted, as the backup data, into a backup memory when the flag is reset; and restoring the backup data stored in the backup memory, as the main memory data, into the main memory when the flag is set.

Shinjo, 1:35-48

> In step S7, when the designated boot mode is not the maintenance mode, i.e., when it is the normal mode, the saving mode is selected, and the main memory data stored in the main memory 12 is saved into the backup memory 13 as the backup data and the backup flag 15 of the backup memory 13 is set (step S8). The process of step S8 is completed and then a reboot is executed.

Shinjo, 3:28-34

> Since the backup flag 15 has been set, the boot process is started in the quick start mode through steps S1 and S2 after the reboot. That is, in step S6, the backup data stored in the backup memory 13 is restored into the main memory 12 as the main memory data. Since the computer system is thus restored to the state immediately after the boot process and the running environment is set, the boot

process of the computer system can be completed without executing the OS initialization process of step S3 or the application initialization process of step S4.

Shinjo, 3:35-45

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Shinjo, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

> In a computer system, when the system is first booted in a normal mode, main memory data stored in a main memory immediately after the system is booted, is stored as backup data in a backup memory or the like. A backup flag representing whether or not the backup data can be restored is set and the system is rebooted. When the system is next booted in the normal mode, the backup data stored in the backup memory or the like is restored as the main memory data in the main memory. The backup flag is automatically reset in a maintenance mode.

Shinjo, Abstract

> According to one aspect of the present invention, there is provided a method for booting a computer system, the method comprising the steps of: setting a boot mode for booting the computer system; determining whether or not the set boot mode is a normal mode; determining whether or not a flag is set when the boot mode is the normal mode, the flag representing whether or not backup data is restorable; storing main memory data to be stored in a main memory immediately after the computer system is booted, as the backup data, into a backup memory when the flag is reset; and restoring the backup data stored in the backup memory, as the main memory data, into the main memory when the flag is set.

Shinjo, 1:32-48

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the

normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-2:68

In the computer system according to the first embodiment wherein the main memory data stored in the main memory 12 is saved into the backup memory 13 as the backup data, when the system power source is turned off, the backup data is erased. It is thus necessary to boot the system in the same manner as the conventional apparatus and save the main memory data stored in the main memory 12 into the backup memory 13 as the backup data immediately after the system is booted. By backing up the backup memory 13 by a battery or the like, such boot process in the system is executed only at the first time.

Shinjo, 3:65-4:8

Since the backup flag 15 has been set, the boot process is started in the quick start mode through steps S1 and S2 after the reboot. That is, in step S6, the backup data stored in the backup memory 13 is restored into the main memory 12 as the main memory data. Since the computer system is thus restored to the state immediately after the boot process and the running environment is set, the boot process of the computer system can be completed without executing the OS initialization process of step S3 or the application initialization process of step S4.

Shinjo, 3:35-45

A disk unit 19 of a computer system according to the second embodiment as shown in FIG. 3 can be used in place of the backup memory 13 shown in FIG. 1. The disk unit 19 includes a backup file 23 for storing the main memory data stored in the main memory 12 as backup data and a backup flag 25. Even through the system power source is turned off, the backup data stored in the backup file 23 is not erased. In the system according to the second embodiment, since the backup data as the main memory data is saved and restored between the main memory 12 and disk unit 19, disk access occurs. Therefore, the time required for the boot process of the system according to the first embodiment is longer than the time required for that of the system according to the first embodiment.

Shinjo, 4:9-24

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Shinjo, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-2:68

> Since the backup flag 15 has been set, the boot process is started in the quick start mode through steps S1 and S2 after the reboot. That is, in step S6, the backup data stored in the backup memory 13 is restored into the main memory 12 as the main memory data. Since the computer system is thus restored to the state immediately after the boot process and the running environment is set, the boot process of the computer system can be completed without executing the OS initialization process of step S3 or the application initialization process of step S4.

Shinjo, 3:35-45

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as

replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-2:68

In step S7, when the designated boot mode is not the maintenance mode, i.e., when it is the normal mode, the saving mode is selected, and the main memory data stored in the main memory 12 is saved into the backup memory 13 as the backup data and the backup flag 15 of the backup memory 13 is set (step S8). The process of step S8 is completed and then a reboot is executed.

Shinjo, 3:28-34

In a system according to the third embodiment as shown in FIG. 4, the memory 14 includes the backup memory 13, and the disk unit 19 includes the backup file 23. In the saving mode, therefore, the main memory data stored in the main memory 12 can be saved as backup data into the backup memory 13 and backup file 23, and the backup flags 15 and 25 can be set.

Shinjo, 4:24-31

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Shinjo, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Shinjo, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-2:68

> After the backup flag 15 is reset in step S3, the boot process is executed as in the conventional system, i.e., the OS initialization process and the application initialization process are executed (steps S4 and S5).

Shinjo, 3:54-57

> In the conventional system, a very large number of times of disk access are necessary for the OS initialization process such as setting of the running environment of an OS and for the application initialization process such as setting of the running environment of an application program, and long time is required for the boot process. In the present invention, however, the system can be booted at high speed.

Shinjo, 4:58-65

Appendix B18
Invalidity of U.S. Patent 8,090,936 based on Shinjo

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Shinjo, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:51-2:68

> After the backup flag 15 is reset in step S3, the boot process is executed as in the conventional system, i.e., the OS initialization process and the application initialization process are executed (steps S4 and S5).

Shinjo, 3:54-57

> In the conventional system, a very large number of times of disk access are necessary for the OS initialization process such as setting of the running environment of an OS and for the application initialization process such as setting of the running environment of an application program, and long time is required for the boot process. In the present invention, however, the system can be booted at high speed.

Appendix B18
Invalidity of U.S. Patent 8,090,936 based on Shinjo

Shinjo, 4:58-65

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Shinjo, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> In the initialization process, a resident load module is loaded into a main memory, various control blocks are produced, the running environment of an operating system (OS) is set, and the running environment of an application system is set (for example, a control process is produced).

Shinjo, 1:16-21

6. The method of claim 1, further comprising updating the list of boot data.	Shinjo, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The boot mode setting unit 17 is constituted of, for example, a service processor (SVP) and used to set a boot mode in the computer system. The boot mode includes a maintenance mode for software maintenance such as replacement of the programs and patch and a mode (normal mode) other than the maintenance mode. The normal mode includes a quick start mode in which a high speed boot can be executed using the backup data and a saving mode in which the main memory data stored in the main memory 12 is saved in the backup memory 13 as the backup data, immediately after a normal boot is executed. It depends upon the set/reset state of the backup flag 15 which of the quick start mode and the saving mode is selected. More specifically, in the normal mode, when the backup flag 15 is set, the quick start mode is selected and, when the backup flag 15 is reset, the saving mode is selected.

Shinjo, 2:52-68

> When software maintenance such as replacement of programs and patch is performed, the boot process is executed in the maintenance mode and thus the backup data cannot be automatically restored. When the system is next booted, the saving mode is selected again. Therefore, the update backup data can always be stored in the backup memory (backup file).

Shinjo, 5:3-9

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Shinjo, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Shinjo, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Shinjo, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Shinjo, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Shinjo, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shinjo discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also ***add disclosure of non-volatile memory to the extent not in claim 1 already***

> The disk unit 16 is constituted of, for example, a magnetic disk unit and stores various programs, data or the like which are to be booted.

Shinjo, 2:48-51

> A disk unit 19 of a computer system according to the second embodiment as shown in FIG. 3 can be used in place of the backup memory 13 shown in FIG. 1. The disk unit 19 includes a backup file 23 for storing the main memory data stored in the main memory 12 as backup data and a backup flag 25. Even through the system power source is turned off, the backup data stored in the backup file 23 is not erased. In the system according to the second embodiment, since the backup data as the main memory data is saved and restored between the main memory 12 and disk unit 19, disk access occurs. Therefore, the time required for the boot process of the system according to the first embodiment is longer than the time required for that of the system according to the first embodiment.

Shinjo, 4:9-23

Appendix B18
Invalidity of U.S. Patent 8,090,936 based on Shinjo

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Shinjo, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Shinjo, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Shinjo, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Shinjo, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Shinjo, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Shinjo, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shinjo discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B19
Invalidity of U.S. Patent 8,090,936 based on Shipman

U.S. Patent No. 5,671,413 to Shipman ("Shipman") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B19
Invalidity of U.S. Patent 8,090,936 based on Shipman

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Shipman, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

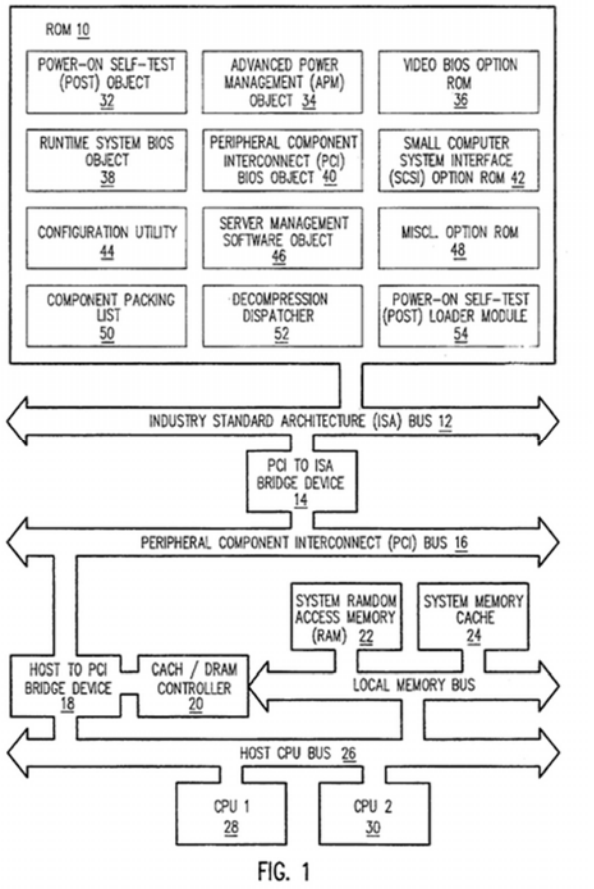


FIG. 1

Shipman, Fig. 1

> In IBM compatible personal computers a set of programs called basic input/output system (BIOS) are encoded in read-only memory (ROM). The BIOS facilitates the transfer of data and instructions between a central processing unit (CPU) and peripheral devices such as disk drives. Computer systems are designed to perform functional tests of the BIOS every time the computer is turned on. When the computer is turned on, BIOS is copied to an area of random access memory (RAM) set aside for it. Since a RAM is much faster acting than a ROM, accessing the BIOS code from RAM results in much faster initialization of the computer.

Shipman, 1:12-22

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the

> service components into random access memory (RAM) of the computer system for execution on an as needed basis, and removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.
>
> More specifically, different basic input/output system (BIOS) functions are split into components, or objects, and selected ones of the components are stored in a compressed state and these compressed components are only decompressed and executed when needed. A decompression dispatcher software takes a request from currently executing BIOS code to decompress another portion of BIOS code that may be needed. The requester can specify whether or not to just decompress the portion of code into memory or to decompress and initialize the decompressed BIOS. The decompression dispatcher can be used to dispatch different portions of BIOS code to different processors in a multi-processor system.

Shipman, 1:35-59

> The BIOS stored in a read only memory (ROM) is organized into a set of components, each being dispatched as an independent executable object after being copied to a shadow memory portion of a random access memory (RAM). Three types of components are defined, initialization components, runtime components and functional components. The initialization components are dispatched during a Power On Self Test (POST) cycle such that the initialization components are active during initialization. The initialization components are terminated prior to loading an operating system. The runtime components are maintained in the uncompressed/decompressed state and executable after loading of the operating system. The functional components are decompressed and dispatched on an as needed basis.

Shipman, 1:60-2:7

> When dispatching components the decompression dispatcher works from a packing list provided in the ROM image by a BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The packing list provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 2:14-22

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

> Compressed code is the code that resides in the ROM in a compressed state. Decompressed code is code that has been decompressed from its compressed state inside the ROM and now resides in DRAM (22) shadowed memory, in a decompressed state. Uncompressed code is code that resides inside the ROM in an uncompressed state. This is code that was never compressed.

Shipman, 3:22-28

> FIG. 13 is a flow diagram of a BIOS build process that automates compressing and packing components into a final ROM binary image. A list of binary images and associated attributes to include in the final ROM image (1300) and a number of binary image files (1302) are provided to a ROM Packing Utility (1304). The ROM Packing Utility (1304) creates a packed ROM image (1306) and a packing list (1308) that are merged by a merge utility (1310) into a final ROM image (1312) and a packing map (1314) that are stored in the ROM (10) shown in FIG. 1.

Shipman, 3:55-65

> Packing List--The Packing List is generated by the Packing Utility and describes all the attributes of the packed ROM image. This list is used by the decompression dispatcher to dispatch components.

Shipman, 4:33-36

> Component Dispatching
>
> Components are dispatched through a utility called the decompression dispatcher. There are two types of dispatching supported by the decompression dispatcher, active and passive.
>
> . . .
>
> When dispatching components the decompression dispatcher works from a Packing List provided in the ROM image by the BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The Packing List provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 5:3-34

> Refer to FIG. 3 which is a flow diagram of initializing the decompression dispatcher of FIG. 2. The dispatcher data and executable code is copied (302) from the ROM to the RAM. The dispatcher interrupt vector is installed (304), the memory allocation table is cleared (306) and the component handle table is cleared (308). The compressed BIOS component information in the packing list is fetched from RAM (310). The relocation type field is examined to find what type of relocation is supported, below 1 meg. (312, 314), above 1 meg. (316, 318), or anywhere (320). If the required size is not available (322) allocation error flags are set (324). If the specified load address is not available (326) allocation error flags are set (328).
>
> If the required size is available (322), then the memory allocation table is updated (330). If the end of the component list in the packing list has not been reached (332), then the flow returns to block (310). If the end of the component list in the packing list has been reached (332), then the flow ends (334).

Shipman, 12:17-35

> Refer to FIG. 5 which is a flow diagram of how components are dispatched. The dispatcher assumes only one component from a class is being loaded, and hence the count of components is set to 1. The counter is used as an index into the packing list. The component of a class might contain a number of components referenced by a subclass

identifier. A check (504) is made to determine if all components of the class are to be loaded. If yes, then the first subclass identifier is fetched (506) as described in FIG. 7. If the subclass is found in the packing list (508) then the count of the subclass index counter is incremented (510). If the subclass is not found in the packing list (508), then the count of the subclass index counter is not incremented and the flow proceeds to get packing information for the component indexed by the count of the component counter (512). The procedure next decompresses the component at the index count and places the decompressed Code in RAM (514). The details of block (514) are shown in more detail in FIG. 12. At the end of the decompressing procedure, a check (516) is made to determine if there are more components in this class to decompressed. If yes, the component index counter is decremented (518). If no, a check (520) is made to determine if this is the last component actively dispatched or if an option ROM is present. If not, the flow ends (524). If yes, then the return address is adjusted to execute the dispatched component (522).

Shipman, 12:49-13:7

Refer to FIG. 12 which is a flow diagram of the procedure for getting a specified component transferred and stored in RAM in an uncompressed and executable state.

A RAM memory block large enough to hold the specified component is found (1202) and allocated. After the memory space is allocated (1204), the size field in the packing list is accessed to get the size information (1206). The compressed and decompressed size information is obtained from the packing list entry (1206). The Compressed Size field specifies the amount of space (in bytes) consumed by a component when it is in the ROM image. Unitialized Data components use this field to specify the maximum amount of memory required and that they be aligned on a 64K boundary. The Decompressed Size field specifies the amount of space (in bytes) consumed by a component after it has been decompressed and dispatched. As described earlier, components that are placed in the ROM uncompressed have identical Compressed and Decompressed sizes. Unitialized Data components specify this field to be identical to the Compressed Size field.

If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Appendix B19
Invalidity of U.S. Patent 8,090,936 based on Shipman

Shipman, 14:15-41

See also Shipman, Figs. 2-13

1.2 initializing a central processing unit of said computer system;	Shipman, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> In IBM compatible personal computers a set of programs called basic input/output system (BIOS) are encoded in read-only memory (ROM). The BIOS facilitates the transfer of data and instructions between a central processing unit (CPU) and peripheral devices such as disk drives. Computer systems are designed to perform functional tests of the BIOS every time the computer is turned on. When the computer is turned on, BIOS is copied to an area of random access memory (RAM) set aside for it. Since a RAM is much faster acting than a ROM, accessing the BIOS code from RAM results in much faster initialization of the computer.

Shipman, 1:12-22

> The BIOS stored in a read only memory (ROM) is organized into a set of components, each being dispatched as an independent executable object after being copied to a shadow memory portion of a random access memory (RAM). Three types of components are defined, initialization components, runtime components and functional

> components. The initialization components are dispatched during a Power On Self Test (POST) cycle such that the initialization components are active during initialization. The initialization components are terminated prior to loading an operating system. The runtime components are maintained in the uncompressed/decompressed state and executable after loading of the operating system. The functional components are decompressed and dispatched on an as needed basis.

Shipman, 1:60-2:7

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

See also Shipman, Figs. 1-13

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Shipman, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> In IBM compatible personal computers a set of programs called basic input/output system (BIOS) are encoded in read-only memory (ROM). The BIOS facilitates the transfer of data and instructions between a central processing unit (CPU) and peripheral devices such as disk drives. Computer systems are designed to perform functional tests of the BIOS every time the computer is turned on. When the computer is turned on, BIOS is copied to an area of random access memory (RAM) set aside for it. Since a RAM is much faster acting than a ROM, accessing the BIOS code from RAM results in much faster initialization of the computer.

Shipman, 1:12-22

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with

> a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.
>
> More specifically, different basic input/output system (BIOS) functions are split into components, or objects, and selected ones of the components are stored in a compressed state and these compressed components are only decompressed and executed when needed. A decompression dispatcher software takes a request from currently executing BIOS code to decompress another portion of BIOS code that may be needed. The requester can specify whether or not to just decompress the portion of code into memory or to decompress and initialize the decompressed BIOS. The decompression dispatcher can be used to dispatch different portions of BIOS code to different processors in a multi-processor system.

Shipman, 1:35-59

> The BIOS stored in a read only memory (ROM) is organized into a set of components, each being dispatched as an independent executable object after being copied to a shadow memory portion of a random access memory (RAM). Three types of components are defined, initialization components, runtime components and functional components. The initialization components are dispatched during a Power On Self Test (POST) cycle such that the initialization components are active during initialization. The initialization components are terminated prior to loading an operating system. The runtime components are maintained in the uncompressed/decompressed state and executable after loading of the operating system. The functional components are decompressed and dispatched on an as needed basis.

Shipman, 1:60-2:7

> When dispatching components the decompression dispatcher works from a packing list provided in the ROM image by a BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The packing list provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 2:14-22

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

> Compressed code is the code that resides in the ROM in a compressed state. Decompressed code is code that has been decompressed from its compressed state inside the ROM and now resides in DRAM (22) shadowed memory, in a decompressed state. Uncompressed code is code that resides inside the ROM in an uncompressed state. This is code that was never compressed.

Shipman, 3:22-28

> FIG. 13 is a flow diagram of a BIOS build process that automates compressing and packing components into a final ROM binary image. A list of binary images and associated attributes to include in the final ROM image (1300) and a number of binary image files (1302) are provided to a ROM Packing Utility (1304). The ROM Packing Utility (1304) creates a packed ROM image (1306) and a packing list (1308) that are merged by a merge utility (1310) into a final ROM image (1312) and a packing map (1314) that are stored in the ROM (10) shown in FIG. 1.

Shipman, 3:55-65

> Packing List--The Packing List is generated by the Packing Utility and describes all the attributes of the packed ROM image. This list is used by the decompression dispatcher to dispatch components.

Shipman, 4:33-36

> Component Dispatching
>
> Components are dispatched through a utility called the decompression dispatcher. There are two types of dispatching supported by the decompression dispatcher, active and passive.
>
> . . .
>
> When dispatching components the decompression dispatcher works from a Packing List provided in the ROM image by the BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The Packing List provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 5:3-34

> Refer to FIG. 3 which is a flow diagram of initializing the decompression dispatcher of FIG. 2. The dispatcher data and executable code is copied (302) from the ROM to the RAM. The dispatcher interrupt vector is installed (304), the memory allocation table is cleared (306) and the component handle table is cleared (308). The compressed BIOS component information in the packing list is fetched from RAM (310). The relocation type field is examined to find what type of relocation is supported, below 1 meg. (312, 314), above 1 meg. (316, 318), or anywhere (320). If the required size is not available (322) allocation error flags are set (324). If the specified load address is not available (326) allocation error flags are set (328).
>
> If the required size is available (322), then the memory allocation table is updated (330). If the end of the component list in the packing list has not been reached (332), then the flow returns to block (310). If the end of the component list in the packing list has been reached (332), then the flow ends (334).

Shipman, 12:17-35

> Refer to FIG. 5 which is a flow diagram of how components are dispatched. The dispatcher assumes only one component from a class is being loaded, and hence the count of components is set to 1. The counter is used as an index into the packing list. The component of a

class might contain a number of components referenced by a subclass identifier. A check (504) is made to determine if all components of the class are to be loaded. If yes, then the first subclass identifier is fetched (506) as described in FIG. 7. If the subclass is found in the packing list (508) then the count of the subclass index counter is incremented (510). If the subclass is not found in the packing list (508), then the count of the subclass index counter is not incremented and the flow proceeds to get packing information for the component indexed by the count of the component counter (512). The procedure next decompresses the component at the index count and places the decompressed Code in RAM (514). The details of block (514) are shown in more detail in FIG. 12. At the end of the decompressing procedure, a check (516) is made to determine if there are more components in this class to decompressed. If yes, the component index counter is decremented (518). If no, a check (520) is made to determine if this is the last component actively dispatched or if an option ROM is present. If not, the flow ends (524). If yes, then the return address is adjusted to execute the dispatched component (522).

Shipman, 12:49-13:7

Refer to FIG. 12 which is a flow diagram of the procedure for getting a specified component transferred and stored in RAM in an uncompressed and executable state.

A RAM memory block large enough to hold the specified component is found (1202) and allocated. After the memory space is allocated (1204), the size field in the packing list is accessed to get the size information (1206). The compressed and decompressed size information is obtained from the packing list entry (1206). The Compressed Size field specifies the amount of space (in bytes) consumed by a component when it is in the ROM image. Unitialized Data components use this field to specify the maximum amount of memory required and that they be aligned on a 64K boundary. The Decompressed Size field specifies the amount of space (in bytes) consumed by a component after it has been decompressed and dispatched. As described earlier, components that are placed in the ROM uncompressed have identical Compressed and Decompressed sizes. Unitialized Data components specify this field to be identical to the Compressed Size field.

If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an

uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Shipman, 14:15-41

See also Shipman, Figs. 1-13

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Shipman, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> In IBM compatible personal computers a set of programs called basic input/output system (BIOS) are encoded in read-only memory (ROM). The BIOS facilitates the transfer of data and instructions between a central processing unit (CPU) and peripheral devices such as disk drives. Computer systems are designed to perform functional tests of the BIOS every time the computer is turned on. When the computer is turned on, BIOS is copied to an area of random access memory (RAM) set aside for it. Since a RAM is much faster acting than a ROM, accessing the BIOS code from RAM results in much faster initialization of the computer.

Shipman, 1:12-22

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the

service components into random access memory (RAM) of the computer system for execution on an as needed basis, and removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

More specifically, different basic input/output system (BIOS) functions are split into components, or objects, and selected ones of the components are stored in a compressed state and these compressed components are only decompressed and executed when needed. A decompression dispatcher software takes a request from currently executing BIOS code to decompress another portion of BIOS code that may be needed. The requester can specify whether or not to just decompress the portion of code into memory or to decompress and initialize the decompressed BIOS. The decompression dispatcher can be used to dispatch different portions of BIOS code to different processors in a multi-processor system.

Shipman, 1:35-59

The BIOS stored in a read only memory (ROM) is organized into a set of components, each being dispatched as an independent executable object after being copied to a shadow memory portion of a random access memory (RAM). Three types of components are defined, initialization components, runtime components and functional components. The initialization components are dispatched during a Power On Self Test (POST) cycle such that the initialization components are active during initialization. The initialization components are terminated prior to loading an operating system. The runtime components are maintained in the uncompressed/decompressed state and executable after loading of the operating system. The functional components are decompressed and dispatched on an as needed basis.

Shipman, 1:60-2:7

When dispatching components the decompression dispatcher works from a packing list provided in the ROM image by a BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The packing list provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 2:14-22

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

> Compressed code is the code that resides in the ROM in a compressed state. Decompressed code is code that has been decompressed from its compressed state inside the ROM and now resides in DRAM (22) shadowed memory, in a decompressed state. Uncompressed code is code that resides inside the ROM in an uncompressed state. This is code that was never compressed.

Shipman, 3:22-28

> FIG. 13 is a flow diagram of a BIOS build process that automates compressing and packing components into a final ROM binary image. A list of binary images and associated attributes to include in the final ROM image (1300) and a number of binary image files (1302) are provided to a ROM Packing Utility (1304). The ROM Packing Utility (1304) creates a packed ROM image (1306) and a packing list (1308) that are merged by a merge utility (1310) into a final ROM image (1312) and a packing map (1314) that are stored in the ROM (10) shown in FIG. 1.

Shipman, 3:55-65

> Packing List--The Packing List is generated by the Packing Utility and describes all the attributes of the packed ROM image. This list is used by the decompression dispatcher to dispatch components.

Shipman, 4:33-36

> Component Dispatching
>
> Components are dispatched through a utility called the decompression dispatcher. There are two types of dispatching supported by the decompression dispatcher, active and passive.
>
> . . .
>
> When dispatching components the decompression dispatcher works from a Packing List provided in the ROM image by the BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The Packing List provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 5:3-34

> Refer to FIG. 3 which is a flow diagram of initializing the decompression dispatcher of FIG. 2. The dispatcher data and executable code is copied (302) from the ROM to the RAM. The dispatcher interrupt vector is installed (304), the memory allocation table is cleared (306) and the component handle table is cleared (308). The compressed BIOS component information in the packing list is fetched from RAM (310). The relocation type field is examined to find what type of relocation is supported, below 1 meg. (312, 314), above 1 meg. (316, 318), or anywhere (320). If the required size is not available (322) allocation error flags are set (324). If the specified load address is not available (326) allocation error flags are set (328).
>
> If the required size is available (322), then the memory allocation table is updated (330). If the end of the component list in the packing list has not been reached (332), then the flow returns to block (310). If the end of the component list in the packing list has been reached (332), then the flow ends (334).

Shipman, 12:17-35

> Refer to FIG. 5 which is a flow diagram of how components are dispatched. The dispatcher assumes only one component from a class is being loaded, and hence the count of components is set to 1. The counter is used as an index into the packing list. The component of a class might contain a number of components referenced by a subclass

identifier. A check (504) is made to determine if all components of the class are to be loaded. If yes, then the first subclass identifier is fetched (506) as described in FIG. 7. If the subclass is found in the packing list (508) then the count of the subclass index counter is incremented (510). If the subclass is not found in the packing list (508), then the count of the subclass index counter is not incremented and the flow proceeds to get packing information for the component indexed by the count of the component counter (512). The procedure next decompresses the component at the index count and places the decompressed Code in RAM (514). The details of block (514) are shown in more detail in FIG. 12. At the end of the decompressing procedure, a check (516) is made to determine if there are more components in this class to decompressed. If yes, the component index counter is decremented (518). If no, a check (520) is made to determine if this is the last component actively dispatched or if an option ROM is present. If not, the flow ends (524). If yes, then the return address is adjusted to execute the dispatched component (522).

Shipman, 12:49-13:7

Refer to FIG. 12 which is a flow diagram of the procedure for getting a specified component transferred and stored in RAM in an uncompressed and executable state.

A RAM memory block large enough to hold the specified component is found (1202) and allocated. After the memory space is allocated (1204), the size field in the packing list is accessed to get the size information (1206). The compressed and decompressed size information is obtained from the packing list entry (1206). The Compressed Size field specifies the amount of space (in bytes) consumed by a component when it is in the ROM image. Unitialized Data components use this field to specify the maximum amount of memory required and that they be aligned on a 64K boundary. The Decompressed Size field specifies the amount of space (in bytes) consumed by a component after it has been decompressed and dispatched. As described earlier, components that are placed in the ROM uncompressed have identical Compressed and Decompressed sizes. Unitialized Data components specify this field to be identical to the Compressed Size field.

If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Shipman, 14:15-41

See also Shipman, Figs. 1-13

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Shipman, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> In IBM compatible personal computers a set of programs called basic input/output system (BIOS) are encoded in read-only memory (ROM). The BIOS facilitates the transfer of data and instructions between a central processing unit (CPU) and peripheral devices such as disk drives. Computer systems are designed to perform functional tests of the BIOS every time the computer is turned on. When the computer is turned on, BIOS is copied to an area of random access memory (RAM) set aside for it. Since a RAM is much faster acting than a ROM, accessing the BIOS code from RAM results in much faster initialization of the computer.

Shipman, 1:12-22

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.
>
> More specifically, different basic input/output system (BIOS) functions are split into components, or objects, and selected ones of the components are stored in a compressed state and these compressed components are only decompressed and executed when needed. A decompression dispatcher software takes a request from currently executing BIOS code to decompress another portion of BIOS code that may be needed. The requester can specify whether or not to just decompress the portion of code into memory or to decompress and initialize the decompressed BIOS. The decompression dispatcher can be used to dispatch different portions of BIOS code to different processors in a multi-processor system.

Shipman, 1:35-59

> The BIOS stored in a read only memory (ROM) is organized into a set of components, each being dispatched as an independent executable object after being copied to a shadow memory portion of a random access memory (RAM). Three types of components are defined, initialization components, runtime components and functional components. The initialization components are dispatched during a Power On Self Test (POST) cycle such that the initialization components are active during initialization. The initialization components are terminated prior to loading an operating system. The runtime components are maintained in the uncompressed/decompressed state and executable after loading of the operating system. The functional components are decompressed and dispatched on an as needed basis.

Shipman, 1:60-2:7

> When dispatching components the decompression dispatcher works from a packing list provided in the ROM image by a BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The

> packing list provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 2:14-22

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

> Compressed code is the code that resides in the ROM in a compressed state. Decompressed code is code that has been decompressed from its compressed state inside the ROM and now resides in DRAM (22) shadowed memory, in a decompressed state. Uncompressed code is code that resides inside the ROM in an uncompressed state. This is code that was never compressed.

Shipman, 3:22-28

> FIG. 13 is a flow diagram of a BIOS build process that automates compressing and packing components into a final ROM binary image. A list of binary images and associated attributes to include in the final ROM image (1300) and a number of binary image files (1302) are provided to a ROM Packing Utility (1304). The ROM Packing Utility (1304) creates a packed ROM image (1306) and a packing list (1308) that are merged by a merge utility (1310) into a final ROM image (1312) and a packing map (1314) that are stored in the ROM (10) shown in FIG. 1.

Shipman, 3:55-65

> Packing List--The Packing List is generated by the Packing Utility and describes all the attributes of the packed ROM image. This list is used by the decompression dispatcher to dispatch components.

Shipman, 4:33-36

> Component Dispatching
>
> Components are dispatched through a utility called the decompression dispatcher. There are two types of dispatching supported by the decompression dispatcher, active and passive.
>
> . . .
>
> When dispatching components the decompression dispatcher works from a Packing List provided in the ROM image by the BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The Packing List provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 5:3-34

> Refer to FIG. 3 which is a flow diagram of initializing the decompression dispatcher of FIG. 2. The dispatcher data and executable code is copied (302) from the ROM to the RAM. The dispatcher interrupt vector is installed (304), the memory allocation table is cleared (306) and the component handle table is cleared (308). The compressed BIOS component information in the packing list is fetched from RAM (310). The relocation type field is examined to find what type of relocation is supported, below 1 meg. (312, 314), above 1 meg. (316, 318), or anywhere (320). If the required size is not available (322) allocation error flags are set (324). If the specified load address is not available (326) allocation error flags are set (328).
>
> If the required size is available (322), then the memory allocation table is updated (330). If the end of the component list in the packing list has not been reached (332), then the flow returns to block (310). If the end of the component list in the packing list has been reached (332), then the flow ends (334).

Shipman, 12:17-35

> Refer to FIG. 5 which is a flow diagram of how components are dispatched. The dispatcher assumes only one component from a class is being loaded, and hence the count of components is set to 1. The counter is used as an index into the packing list. The component of a class might contain a number of components referenced by a subclass identifier. A check (504) is made to determine if all components of the class are to be loaded. If yes, then the first subclass identifier is fetched (506) as described in FIG. 7. If the subclass is found in the packing list (508) then the count of the subclass index counter is incremented (510). If the subclass is not found in the packing list (508), then the count of the subclass index counter is not incremented and the flow proceeds to get packing information for the component indexed by the count of the component counter (512). The procedure next decompresses the component at the index count and places the decompressed Code in RAM (514). The details of block (514) are shown in more detail in FIG. 12. At the end of the decompressing procedure, a check (516) is made to determine if there are more components in this class to decompressed. If yes, the component index counter is decremented (518). If no, a check (520) is made to determine if this is the last component actively dispatched or if an option ROM is present. If not, the flow ends (524). If yes, then the return address is adjusted to execute the dispatched component (522).

Shipman, 12:49-13:7

> Refer to FIG. 12 which is a flow diagram of the procedure for getting a specified component transferred and stored in RAM in an uncompressed and executable state.
>
> A RAM memory block large enough to hold the specified component is found (1202) and allocated. After the memory space is allocated (1204), the size field in the packing list is accessed to get the size information (1206). The compressed and decompressed size information is obtained from the packing list entry (1206). The Compressed Size field specifies the amount of space (in bytes) consumed by a component when it is in the ROM image. Unitialized Data components use this field to specify the maximum amount of memory required and that they be aligned on a 64K boundary. The Decompressed Size field specifies the amount of space (in bytes) consumed by a component after it has been decompressed and dispatched. As described earlier, components that are placed in the ROM uncompressed have identical Compressed and Decompressed sizes. Unitialized Data components specify this field to be identical to the Compressed Size field.

If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Shipman, 14:15-41

See also Shipman, Figs. 1-13

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Shipman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Shipman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> As more new features are designed into computers, more code has to be stored into the BIOS stored on the ROM. The size of the ROM must be increased to accommodate the additional code, resulting in additional manufacturing costs. It is desirable to decreases product cost by reducing the size of a ROM required to store the BIOS and all of its components. This can be done by decompressing the code as it is stored on the ROM and then decompressing the code at the time it is copied to the RAM.

Shipman, 1:23-32

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Shipman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> As more new features are designed into computers, more code has to be stored into the BIOS stored on the ROM. The size of the ROM must be increased to accommodate the additional code, resulting in additional manufacturing costs. It is desirable to decreases product cost by reducing the size of a ROM required to store the BIOS and all of its components. This can be done by decompressing the code as it is stored on the ROM and then decompressing the code at the time it is copied to the RAM.

Shipman, 1:23-32

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Shipman, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional components.

Shipman, 2:63-3:12

> Refer to FIG. 1 which is a block diagram of a computer system in which the present invention is embodied. The computer includes a read only memory, or ROM, (10), a dynamic random access memory, or DRAM, (22), a system memory cache (24), a cache/DRAM controller (20), and central processing units (28, 30). A set of programs called basic input/output system (BIOS) are encoded in ROM (10). The BIOS facilitates the transfer of data and instructions between the central processing units (28, 30) and peripheral devices. In the present invention the BIOS is organized into a set of BIOS components (32 through 54), each of which is capable of being dispatched as an independent executable object. Each BIOS component is a self-contained (link independent) binary image that performs a certain task or function. The BIOS components fall into three major types: initialization components, runtime components and functional

components.

Shipman, 2:63-3:12

System random access memory, RAM, (22) and system memory cache (24) am connected to the Cache/DRAM Controller (20) via a local memory bus. The CPU 1 (28) and the CPU 2 (30) is connected to the host to PCI bridge device (18) via the Host CPU Bus (26).

Shipman, 3:50-54

6. The method of claim 1, further comprising updating the list of boot data.	Shipman, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> When dispatching components the decompression dispatcher works from a packing list provided in the ROM image by a BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The packing list provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 2:14-22

> FIG. 13 is a flow diagram of a BIOS build process that automates compressing and packing components into a final ROM binary image. A list of binary images and associated attributes to include in the final ROM image (1300) and a number of binary image files (1302) are provided to a ROM Packing Utility (1304). The ROM Packing Utility (1304) creates a packed ROM image (1306) and a packing list (1308) that are merged by a merge utility (1310) into a final ROM image

(1312) and a packing map (1314) that are stored in the ROM (10) shown in FIG. 1.

Shipman, 3:55-65

Packing List--The Packing List is generated by the Packing Utility and describes all the attributes of the packed ROM image. This list is used by the decompression dispatcher to dispatch components.

Shipman, 4:33-36

Component Dispatching

Components are dispatched through a utility called the decompression dispatcher. There are two types of dispatching supported by the decompression dispatcher, active and passive.

. . .

When dispatching components the decompression dispatcher works from a Packing List provided in the ROM image by the BIOS build process. To optimize device usage, the binary images that comprise the different components are packed into the final ROM image. The Packing List provides a detailed description of the packing as well as other important pieces of information regarding the types of components that have been packed into the ROM.

Shipman, 5:3-34

Refer to FIG. 3 which is a flow diagram of initializing the decompression dispatcher of FIG. 2. The dispatcher data and executable code is copied (302) from the ROM to the RAM. The dispatcher interrupt vector is installed (304), the memory allocation table is cleared (306) and the component handle table is cleared (308). The compressed BIOS component information in the packing list is fetched from RAM (310). The relocation type field is examined to find what type of relocation is supported, below 1 meg. (312, 314), above 1 meg. (316, 318), or anywhere (320). If the required size is not available (322) allocation error flags are set (324). If the specified load address is not available (326) allocation error flags are set (328).

If the required size is available (322), then the memory allocation table is updated (330). If the end of the component list in the packing list has not been reached (332), then the flow returns to block (310). If the end

of the component list in the packing list has been reached (332), then the flow ends (334).

Shipman, 12:17-35

Refer to FIG. 5 which is a flow diagram of how components are dispatched. The dispatcher assumes only one component from a class is being loaded, and hence the count of components is set to 1. The counter is used as an index into the packing list. The component of a class might contain a number of components referenced by a subclass identifier. A check (504) is made to determine if all components of the class are to be loaded. If yes, then the first subclass identifier is fetched (506) as described in FIG. 7. If the subclass is found in the packing list (508) then the count of the subclass index counter is incremented (510). If the subclass is not found in the packing list (508), then the count of the subclass index counter is not incremented and the flow proceeds to get packing information for the component indexed by the count of the component counter (512). The procedure next decompresses the component at the index count and places the decompressed Code in RAM (514). The details of block (514) are shown in more detail in FIG. 12. At the end of the decompressing procedure, a check (516) is made to determine if there are more components in this class to decompressed. If yes, the component index counter is decremented (518). If no, a check (520) is made to determine if this is the last component actively dispatched or if an option ROM is present. If not, the flow ends (524). If yes, then the return address is adjusted to execute the dispatched component (522).

Shipman, 12:49-13:7

Refer to FIG. 12 which is a flow diagram of the procedure for getting a specified component transferred and stored in RAM in an uncompressed and executable state.

A RAM memory block large enough to hold the specified component is found (1202) and allocated. After the memory space is allocated (1204), the size field in the packing list is accessed to get the size information (1206). The compressed and decompressed size information is obtained from the packing list entry (1206). The Compressed Size field specifies the amount of space (in bytes) consumed by a component when it is in the ROM image. Unitialized Data components use this field to specify the maximum amount of memory required and that they be aligned on a 64K boundary. The Decompressed Size field specifies the amount of space (in bytes) consumed by a component after it has been decompressed and dispatched. As described earlier, components that are

placed in the ROM uncompressed have identical Compressed and Decompressed sizes. Unitialized Data components specify this field to be identical to the Compressed Size field.

If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Shipman, 14:15-41

See also Shipman, Figs. 1-13

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Shipman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Shipman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Shipman, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Shipman, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Shipman, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Shipman discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and optionally removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented without requiring more non-volatile storage.

Shipman, Abstract

> The services to be provided by a basic input/output system (BIOS) of a computer system are implemented via a number of independently executable service components. Additionally, the BIOS is provided with a decompression dispatcher for decompressing and dispatching the service components into random access memory (RAM) of the computer system for execution on an as needed basis, and removing the dispatched service components when they are no longer needed. As a result, the service components may be stored in a non-volatile storage in a compressed state, allowing more services to be implemented

without requiring more non-volatile storage.

Shipman, 1:35-46

> The ROM (10) is a non-volatile device used to store the BIOS components, some in a compressed state and some in an uncompressed state. The DRAM (22) stores a shadow RAM binary image of selected components stored in the ROM. For those components stored in the compressed state, they are decompressed before storing in the shadow RAM. In Intel 80386 and 80486 computers, shadow RAM is a portion of upper memory area set aside for programs retrieved from ROM.

Shipman, 3:13-21

> If the component is not compressed, then the component is copied (1212) to RAM from the non-volatile storage device used to store the compressed or uncompressed BIOS code. If the component is compressed, then the component is decompressed (1210), stored in an uncompressed state in the shadowed memory portion of RAM, and the procedure stops (1216).

Shipman, 14:35-41

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Shipman, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Shipman, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Shipman, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Shipman, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Shipman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B19
Invalidity of U.S. Patent 8,090,936 based on Shipman

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Shipman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Shipman discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B20
Invalidity of U.S. Patent 8,090,936 based on Sukegawa

U.S. Patent No. 5,860,083 to Sukegawa ("Sukegawa") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

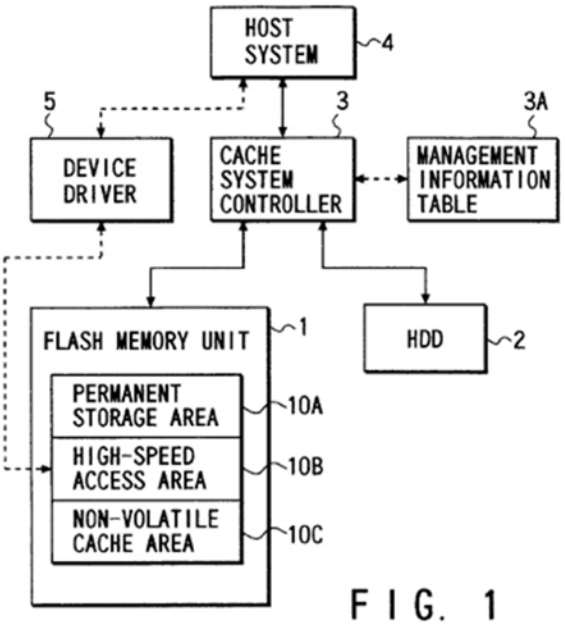
The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

In addition, Apple incorporates by reference, as if set forth fully herein, all arguments related to Sukegawa in pending inter partes review petitions IPR2016-1365, IPR2016-01366, IPR2016-01737, and IPR2016-01738.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Sukegawa, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this claim limitation:



F I G. 1

Sukegawa, Fig. 1

> "(Data Storage System)
> It is assumed that the data storage system of the present invention is applied to a computer system such as a personal computer. Thus, as shown in FIG. 1, this data storage system has a flash memory unit 1 constituted by a flash EEPROM, a disk drive or an HDD (Hard Disk Drive) 2, and a controller. The controller includes a cache system controller 3 (hereinafter called "controller") for performing a cache function of using the flash memory unit 1 as a cache memory, and a device driver (software) 5 with no cache function.
>
> The device driver 5 has a function of controlling the flash memory unit 1 under the management of the OS (Operation System) of a host system 4. In this invention, in particular, the device driver 5 performs the access control (to be described later) of a highspeed access area lOB or a specific storage area in the flash memory unit 1. The controller 3 performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD 2 via respective device drivers (i.e. a flash memory driver and a hard disk driver)."

Sukegawa, 4:1-21.

> "The host system 4 refers to a computer body comprising a CPU of the computer system, a main memory storing the OS and an application program (AP), and other various structural elements. The controller 3 is provided between the host system 4 and the storage units 1 and 2. The controller 3 controls the flash memory unit 1 and HDD 2, as an integrated storage system, in accordance with access requests (read/write commands) issued from the host system 4 to the HDD."

Sukegawa, 4:22-30.

> "According to this system, for example, control information necessary for starting an application program (AP) and an OS, which are frequently used, is stored in the first storage area. Thus, the storage area of the flash memory can be effectively used in accordance with the function and the condition of use of data."

Sukegawa, 2:65-3:3.

> "(First Modification of the First Embodiment)
> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area lOA of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of

> operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

see also Sukegawa 5:1-7:2, 7:28-55, 9:1-10, 10:33-52, 11:7-19, Fig. 4, Fig. 5.

1.2 initializing a central processing unit of said computer system;	Sukegawa, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this claim limitation:

> "The host system 4 refers to a computer body comprising a CPU of the computer system, a main memory storing the OS and an application program (AP), and other various structural elements. The controller 3 is provided between the host system 4 and the storage units 1 and 2. The controller 3 controls the flash memory unit 1 and HDD 2, as an integrated storage system, in accordance with access requests (read/write commands) issued from the host system 4 to the HDD."

Sukegawa, 4:22-30.

> "(First Modification of the First Embodiment)
> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area lOA of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

See also Sukegawa, 6:19-58.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Sukegawa, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this claim limitation:

> "(Data Storage System)
> It is assumed that the data storage system of the present invention is applied to a computer system such as a personal computer. Thus, as shown in FIG. 1, this data storage system has a flash memory unit 1 constituted by a flash EEPROM, a disk drive or an HDD (Hard Disk Drive) 2, and a controller. The controller includes a cache system controller 3 (hereinafter called "controller") for performing a cache function of using the flash memory unit 1 as a cache memory, and a device driver (software) 5 with no cache function.
>
> The device driver 5 has a function of controlling the flash memory unit 1 under the management of the OS (Operation System) of a host system 4. In this invention, in particular, the device driver 5 performs the access control (to be described later) of a highspeed access area lOB or a specific storage area in the flash memory unit 1. The controller 3 performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD 2 via respective device drivers (i.e. a flash memory driver and a hard disk driver)."

Sukegawa, 4:1-21.

> "The first embodiment relates to a system wherein the permanent storage area 10A of flash memory unit 1 is used as a cache memory area."

Sukegawa, 5:10-12.

> "(First Modification of the First Embodiment)

> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area lOA of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

> "In the system of the present invention, when the host system 4 issues the read command to the HDD 2, the controller 3 determines whether the data to be accessed (e.g. AP control information as mentioned above) is stored in the permanent storage area 10A or non-volatile cache area 10C, which is the cache memory area (or whether the cache memory area is "hit") (steps S20 and S21), as shown in FIGS. 5. If the data to be accessed is "hit", the controller 3 reads the data from the permanent storage area 10A or non-volatile cache area 10 and transfers the read-out data to the host system 4 ("YES" in step S21; step S25). On the other hand, if the cache memory area is not "hit", the controller 3 accesses the HDD 2, reads out the data to be accessed and transfers the read-out data to the host system 4. In this case, as described above,

> if the data to be accessed is the permanent data designated by the user, the controller 3 stores it in the permanent storage area 10A ("NO" in step S21; steps S22 and 23). If the data to be accessed is not the permanent data designated by the user, the controller 3 stores the data in the non-volatile cache area 10C which is used as an ordinary cache memory area ("NO" in step S22; step S24). Thus, the data read out from the HDD 2 is stored at least once in the permanent storage area 10A and nonvolatile cache area 10C in flash memory unit 1. In particular, it is assumed that the non-volatile cache area 10C is used independently by the controller 3 and not directly used by the user's intent."

Sukegawa, 7:28-55

see also Sukegawa 5:1-7:2, 9:1-10, 10:33-52, 11:7-19, Fig 1, Fig. 4, Fig. 5.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Sukegawa, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this claim limitation:

> "According to this system, for example, control information necessary for starting an application program (AP) and an OS, which are frequently used, is stored in the first storage area. Thus, the storage area of the flash memory can be effectively used in accordance with the function and the condition of use of data."

Sukegawa, 2:65-3:3.

> "(Data Storage System)
> It is assumed that the data storage system of the present invention is applied to a computer system such as a personal computer. Thus, as shown in FIG. 1, this data storage system has a flash memory unit 1 constituted by a flash EEPROM, a disk drive or an HDD (Hard Disk Drive) 2, and a controller. The controller includes a cache system controller 3 (hereinafter called "controller") for performing a cache function of using the flash memory unit 1 as a cache memory, and a device driver (software) 5 with no cache function.
>
> The device driver 5 has a function of controlling the flash memory unit 1 under the management of the OS (Operation System) of a host system 4. In this invention, in particular, the device driver 5 performs the access control (to be described later) of a highspeed access area lOB or a specific storage area in the flash memory unit 1. The controller 3 performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD 2 via respective device drivers (i.e. a flash memory driver and a hard disk driver).
>
> The host system 4 refers to a computer body comprising a CPU of the computer system, a main memory storing the OS and an application

> program (AP), and other various structural elements. The controller 3 is provided between the host system 4 and the storage units 1 and 2. The controller 3 controls the flash memory unit 1 and HDD 2, as an integrated storage system, in accordance with access requests (read/write commands) issued from the host system 4 to the HDD."

Sukegawa, 4:1-30.

> "The first embodiment relates to a system wherein the permanent storage area 10A of flash memory unit 1 is used as a cache memory area."

Sukegawa, 5:10-12.

> "(First Modification of the First Embodiment)
> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area lOA of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

See also Sukegawa 5:1-7:2, 7:28-55, 9:1-10, 10:33-52, 11:7-19, Fig 1, Fig. 4, Fig. 5.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Sukegawa, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this claim limitation:

> "(Data Storage System)
> It is assumed that the data storage system of the present invention is applied to a computer system such as a personal computer. Thus, as shown in FIG. 1, this data storage system has a flash memory unit 1 constituted by a flash EEPROM, a disk drive or an HDD (Hard Disk Drive) 2, and a controller. The controller includes a cache system controller 3 (hereinafter called "controller") for performing a cache function of using the flash memory unit 1 as a cache memory, and a device driver (software) 5 with no cache function.
>
> The device driver 5 has a function of controlling the flash memory unit 1 under the management of the OS (Operation System) of a host system 4. In this invention, in particular, the device driver 5 performs the access control (to be described later) of a highspeed access area lOB or a specific storage area in the flash memory unit 1. The controller 3 performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD 2 via respective device drivers (i.e. a flash memory driver and a hard disk driver).
>
> The host system 4 refers to a computer body comprising a CPU of the computer system, a main memory storing the OS and an application program (AP), and other various structural elements. The controller 3 is provided between the host system 4 and the storage units 1 and 2. The controller 3 controls the flash memory unit 1 and HDD 2, as an integrated storage system, in accordance with access requests (read/write commands) issued from the host system 4 to the HDD."

Sukegawa, 4:1-30.

> "The first embodiment relates to a system wherein the permanent storage area 10A of flash memory unit 1 is used as a cache memory area."

Sukegawa, 5:10-12.

> "(First Modification of the First Embodiment)
> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area lOA of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

See also Sukegawa 5:1-7:2, 7:28-55, 9:1-10, 10:33-52, 11:7-19, Fig 1, Fig. 4, Fig. 5.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Sukegawa, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Sukegawa, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The first embodiment relates to a system wherein the permanent storage area lOA of flash memory unit 1 is used as a cache memory area. In this embodiment, it is supposed that the user desires to start a frequently used application program (AP) at high speed at all times.
>
> The user starts a data storage utility program of the cache system controller 3 via a user interface provided in the host system 4 (step S1). The data storage utility program reads specified data from the HDD 2 and stores the read data in a specified storage area in the flash memory unit 1. In this case, it is assumed that the user sets the permanent storage area 10A in the flash memory unit 1 as the data storage area, at the time of instructing the start of the data storage utility program (step S2).
>
> Then, the user instructs the host system 4 to start the AP (step S3). The host system 4, upon receiving the AP start instruction, issues a read command to the controller 3 in order to read control information necessary for the start of the AP from the HDD 2.
>
> The controller 3 controls the HDD 2, reads out the control information necessary for the start of the AP and transfers the read-out control information to the host system 4 (step S4). At this time, according to the started-up data storage utility program, the controller 3 stores the AP control information read out from the HDD 2 in the permanent storage area 10A of flash memory unit 1 (step S5). When the AP has been prepared to start, the data storage utility program is stopped by the

instruction from the user ("YES" in step S6; step S7). Through these operations, the control information necessary for starting the AP is stored in the permanent storage area 10A in the flash memory unit 1.

Sukegawa 5:10-40

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Sukegawa, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The first embodiment relates to a system wherein the permanent storage area lOA of flash memory unit 1 is used as a cache memory area. In this embodiment, it is supposed that the user desires to start a frequently used application program (AP) at high speed at all times.
>
> The user starts a data storage utility program of the cache system controller 3 via a user interface provided in the host system 4 (step S1). The data storage utility program reads specified data from the HDD 2 and stores the read data in a specified storage area in the flash memory unit 1. In this case, it is assumed that the user sets the permanent storage area 10A in the flash memory unit 1 as the data storage area, at the time of instructing the start of the data storage utility program (step S2).
>
> Then, the user instructs the host system 4 to start the AP (step S3). The host system 4, upon receiving the AP start instruction, issues a read command to the controller 3 in order to read control information necessary for the start of the AP from the HDD 2.
>
> The controller 3 controls the HDD 2, reads out the control information necessary for the start of the AP and transfers the read-out control information to the host system 4 (step S4). At this time, according to the started-up data storage utility program, the controller 3 stores the AP control information read out from the HDD 2 in the permanent storage area 10A of flash memory unit 1 (step S5). When the AP has been

prepared to start, the data storage utility program is stopped by the instruction from the user ("YES" in step S6; step S7). Through these operations, the control information necessary for starting the AP is stored in the permanent storage area 10A in the flash memory unit 1.

Sukegawa 5:10-40

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Sukegawa, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> It is assumed that the data storage system of the present invention is applied to a computer system such as a personal computer. Thus, as shown in FIG. 1, this data storage system has a flash memory unit 1 constituted by a flash EEPROM, a disk drive or an HDD (Hard Disk Drive) 2, and a controller. The controller includes a cache system controller 3 (hereinafter called "controller") for performing a cache function of using the flash memory unit 1 as a cache memory, and a device driver (software) 5 with no cache function.
>
> The device driver 5 has a function of controlling the flash memory unit 1 under the management of the OS (Operation System) of a host system 4. In this invention, in particular, the device driver 5 performs the access control (to be described later) of a highspeed access area 10B or a specific storage area in the flash memory unit 1. The controller 3 performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD 2 via respective device drivers (i.e. a flash memory driver and a hard disk driver).

Sukegawa 4:1-21

6. The method of claim 1, further comprising updating the list of boot data.	Sukegawa, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Sukegawa discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

Sukegawa discloses this claim limitation:

> "The host system 4 refers to a computer body comprising a CPU of the computer system, a main memory storing the OS and an application program (AP), and other various structural elements. The controller 3 is provided between the host system 4 and the storage units 1 and 2. The controller 3 controls the flash memory unit 1 and HDD 2, as an integrated storage system, in accordance with access requests (read/write commands) issued from the host system 4 to the HDD."

Sukegawa, 4:22-30.

> "According to this system, for example, control information necessary for starting an application program (AP) and an OS, which are frequently used, is stored in the first storage area. Thus, the storage area of the flash memory can be effectively used in accordance with the function and the condition of use of data."

Sukegawa, 2:65-3:3;

> "(First Modification of the First Embodiment)
> FIG. 4 is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area 10A of flash memory unit 1, for example, when the OS of the host system 4 is started in a series of operations from the turn-on of power to the completion of the starting operation."

Sukegawa, 6:18-26.

> "According to the data storage utility program, the control information, which is pre-stored in the HDD 2 and necessary for starting the OS, is read out and stored in the permanent storage area 10A (steps S16 and S17). The controller 3 transfers to the host system 4 the control information necessary for starting the OS read out from the HDD. Based on the control information, the host system 4 starts the OS."

Sukegawa, 6:35-42.

> "According to this system, when the OS is automatically started by the control information read out from the HDD 2 at the time of turning-on of power, the control information is stored in the permanent storage area 10 used as the cache memory area for the HDD 2. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD 2 but from the permanent storage area 10 or cache memory area, and the read-out control information is transferred to the host system 4. Thus, the control information can be accessed from the permanent storage area 10A in the flash memory unit 1 having a higher access speed than the HDD 2. As a result, the OS can be started at higher speed."

Sukegawa, 6:45-58.

> "The permanent storage area 10A and non-volatile cache area 10C function as cache memory areas of the HDD 2. Normally, each time the data in the cache memory area is updated, the updated data is written in the HDD 2. In this embodiment, the user can set the mode of each of the areas 10A and 10C, thereby determining whether or not the updated data should be written in the HDD 2 each time the data is updated."

Sukegawa, 9:19-29.

See also Sukegawa 5:1-7:2, 7:28-55, 9:1-10, 9:53-10:52, 11:7-19, Fig 1, Fig. 4, Fig. 5

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Sukegawa, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Sukegawa, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Sukegawa, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Sukegawa, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Sukegawa, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Sukegawa, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Sukegawa, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Sukegawa, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Sukegawa, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Sukegawa, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Sukegawa, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Sukegawa discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B21
Invalidity of U.S. Patent 8,090,936 based on Surine

U.S. Patent No. 6,212,632 to Surine ("Surine") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Surine, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

> "At power-up, boot code stored in the non-volatile memory is executed and begins instantiating the initial operating environment of the embedded computer system. A function pointer table is instantiated in the volatile memory, wherein the function pointer table includes a plurality of entries for a corresponding plurality of instantiated functions, wherein at least one entry is for operating system code stored in the non-volatile memory."

Surine, Abstract.

> "At power-up, boot code stored in the ROM is executed and begins instantiating the initial operating environment of the embedded system. A function pointer table is instantiated in the RAM. The function pointer table has entries, or function pointers, for each instantiated function such that they can each call each other and pass execution. The function pointer table has entries for functions which are instantiated in ROM and entries for functions which are instantiated in RAM."

Surine, 2:66-3:7.

> "Referring now to FIG. 4, a memory diagram depicting the contents of ROM 310 and RAM 315 is shown. As shown in FIG. 4, ROM 310 stores software including boot code 401, compressed high-use functions 402,

and operating system code 403."

Surine, 5:15-19.

> "Function pointer table 510 includes a plurality of entries, or function pointers, which, when called by processing unit 305, redirect program execution to the memory address of a selected routine. The function pointers of function pointer table allow instantiated functions, whether executing from ROM 310 or RAM 315, to call one another. Initially, the function pointers are initialized to addresses in ROM 310, but after copying to RAM 315, the high-use function pointers are updated to addresses in RAM 315."

Surine, 6:24-29. *See also* Surine, 7:5-21, 8:54-56.

1.2 initializing a central processing unit of said computer system;	Surine, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

> "At power-up, boot code stored in the non-volatile memory is executed and begins instantiating the initial operating environment of the embedded computer system. A function pointer table is instantiated in the volatile memory, wherein the function pointer table includes a plurality of entries for a corresponding plurality of instantiated functions, wherein at least one entry is for operating system code stored in the non-volatile memory."

Surine, Abstract.

> "At boot time, or power-up, boot code 202 executes, decompresses compressed code 201 into camera system code 203 and loads camera system code 203 into RAM 102."

Surine, 2:4-7, Fig. 2.

> "At system 300 power-up, boot code 401 is executed and begins setting up the initial software environment of embedded system 300. Boot code 401 initializes the initial software environment of system 300 by setting up capture buffer 414, display buffer 415, and working memory 420."

Surine, 5:31-35.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Surine, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

> "Typically, as shown in FIG. 2, the camera system code 203 (e.g., operating system software and its associated data structures, resources, etc.) is stored as compressed code 201 in non-volatile ROM 104."

Surine, 2:1-4, Fig. 2.

> "At boot time, or power-up, boot code 202 executes, decompresses compressed code 201 into camera system code 203 and loads camera system code 203 into RAM 102."

Surine, 2:4-7, Fig. 2.

> "Consequently, these digital cameras and other performance-oriented types of embedded system consumer electronic devices transfer a compressed image of their system code from a non-volatile ROM to a faster RAM at power up. The system code then executes from RAM."

Surine, 2:21-25.

> "Referring now to FIG. 4, a memory diagram depicting the contents of ROM 310 and RAM 315 is shown. As shown in FIG. 4, ROM 310 stores software including boot code 401, compressed high-use functions 402, and operating system code 403."

Surine, 5:15-19.

> "Patch manager 405 subsequently executes. Patch manager 405 includes decompression software which decompresses compressed high-use functions 402 and, as described in greater detail in the discussion of FIG.

> 5 below, loads the resulting decompressed high-use functions 416 into RAM 315."

Surine, 5:35-40.

> "Patch manager 405 then loads the decompressed high-use functions 416 into RAM 315 and updates function pointer table 510 with entries for high-use functions 416."

Surine: 6:32-34.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Surine, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

> "At least one high-use function is decompressed out of the non-volatile memory and instantiated in volatile memory. The function pointer table is updated using a patch manager to incorporate an entry for the high-use function(s). The operating system code is executed from the non-volatile memory while the high-use function is executed from the volatile memory."

Surine, Abstract.

> "At boot time, or power-up, boot code 202 executes, decompresses compressed code 201 into camera system code 203 and loads camera system code 203 into RAM 102."

Surine, 2:4-7, Fig. 2.

> "Consequently, these digital cameras and other performance-oriented types of embedded system consumer electronic devices transfer a compressed image of their system code from a non-volatile ROM to a faster RAM at power up. The system code then executes from RAM."

Surine, 2:21-25.

> "In accordance with the present invention, a set of high-use functions are decompressed out of ROM and instantiated in RAM using a patch manager."

Surine, 3:7-9.

> "In this embodiment, in addition to instantiating certain high-use functions, a memory configuration manager decompresses new software

> functions out of ROM, instantiates them in RAM, and updates the function pointer table to link them dynamically, as the capability of the new software functions are needed."

Surine, 3:27-32.

> "Patch manager 405 subsequently executes. Patch manager 405 includes decompression software which decompresses compressed high-use functions 402 and, as described in greater detail in the discussion of FIG. 5 below, loads the resulting decompressed high-use functions 416 into RAM 315."

Surine, 5:35-40.

> "Patch manager 405 then loads the decompressed high-use functions 416 into RAM 315 and updates function pointer table 510 with entries for high-use functions 416."

Surine: 6:32-34. *See also* Surine, 7:5-44, 8:14-17, 8:36-43.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Surine, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

> "At least one high-use function is decompressed out of the non-volatile memory and instantiated in volatile memory. The function pointer table is updated using a patch manager to incorporate an entry for the high-use function(s). The operating system code is executed from the non-volatile memory while the high-use function is executed from the volatile memory."

Surine, Abstract.

> "In accordance with the present invention, a set of high-use functions are decompressed out of ROM and instantiated in RAM using a patch manager."

Surine, 3:7-9. *See also* 8:14-17, 8:36-43.

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Surine, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Surine, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Similarly, in addition to extending or modifying functions of operating system code 403, patch manager 405 can extend the functionality of any software application which executes on embedded system 300."

Surine, 6:59-62.

> "After boot and system instantiation is complete, application code uses an available block of memory within RAM 315 to use as a display buffer 751, working memory 752, and a capture buffer 753."

Surine, 7:55-58.

> "In step 807, the initial software environment for embedded system 300 is set up by an application. For example, a first executed application (e.g., the default application code which executes after power-up) uses a block of available memory of RAM 315 for capture buffers, display buffers, working memory, and other software data structures which need to be instantiated in the writeable address space of RAM 314."

Surine, 8:63-9:3.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Surine, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "Similarly, in addition to extending or modifying functions of operating system code 403, patch manager 405 can extend the functionality of any software application which executes on embedded system 300."

Surine, 6:59-62.

> "After boot and system instantiation is complete, application code uses an available block of memory within RAM 315 to use as a display buffer 751, working memory 752, and a capture buffer 753."

Surine, 7:55-58.

> "In step 807, the initial software environment for embedded system 300 is set up by an application. For example, a first executed application (e.g., the default application code which executes after power-up) uses a block of available memory of RAM 315 for capture buffers, display buffers, working memory, and other software data structures which need to be instantiated in the writeable address space of RAM 314."

Surine, 8:63-9:3.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Surine, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Surine, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "At least one high-use function is decompressed out of the non-volatile memory and instantiated in volatile memory. The function pointer table is updated using a patch manager to incorporate an entry for the high-use function(s). The operating system code is executed from the non-volatile memory while the high-use function is executed from the volatile memory."

Surine, Abstract.

> "The patch manager subsequently updates the function pointer table to incorporate an entry for the high-use functions, thereby linking the high-use functions with the rest of the instantiated functions."

Surine, 3:15-18.

> "Patch manager 405 then loads the decompressed high-use functions 416 into RAM 315 and updates function pointer table 510 with entries for high-use functions 416."

Surine: 6:32-34.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Surine, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Surine, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

> "Patch manager 405 includes decompression software which decompresses compressed high-use functions 402 and, as described in greater detail in the discussion of FIG. 5 below, loads the resulting decompressed high-use functions 416 into RAM 315. This is shown by arrow 410."

Surine, 5:36-41.

11.1. a processor;	Surine, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Surine, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Surine, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Surine discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> "The embedded computer system includes a processor coupled to the volatile and non-volatile memories via a bus. The volatile and non volatile memories store computer readable software for execution by the embedded computer system."

Surine, Abstract.

> "At power-up, boot code stored in the non-volatile memory is executed and begins instantiating the initial operating environment of the embedded computer system. A function pointer table is instantiated in the volatile memory, wherein the function pointer table includes a plurality of entries for a corresponding plurality of instantiated functions, wherein at least one entry is for operating system code stored in the non-volatile memory. At least one high-use function is decompressed out of the non-volatile memory and instantiated 'in volatile memory."

Surine, Abstract.

> "The operating system code is executed from the non-volatile memory while the high-use function is executed from the volatile memory."

Surine, Abstract.

> "Typically, as shown in FIG. 2, the camera system code 203 (e.g., operating system software and its associated data structures, resources, etc.) is stored as compressed code 201 in non-volatile ROM 104."

Surine, 2:1-4.

> "Consequently, these digital cameras and other performance-oriented types of embedded system consumer electronic devices transfer a compressed image of their system code from a non-volatile ROM to a faster RAM at power up."

Surine, 2:21-25.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Surine, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Surine, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Surine, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Surine, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Surine, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Surine, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Surine discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B22
Invalidity of U.S. Patent 8,090,936 based on Teoman

U.S. Patent No. 6,370,614 Teoman ("Teoman") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Teoman, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

> "According to one embodiment, the user cache 25 includes boot firmware 66 for storing program code that is used to support operation of the user cache at system startup."

Teoman, 7:50-52.

> "In yet another embodiment for using the user cache to support system startup, before the OS boot sequence is begun, boot program code in the user cache firmware is executed to notify the system BIOS that the user cache is a bootable mass storage device. Also, when executed, the boot program code loads software into system memory for operating the user cache as a bootable mass storage device. In this embodiment, the user selects the user cache as the boot device in the system BIOS settings. Then, during the boot sequence, the operating system accesses boot up files in the user cache."

Teoman, 13:52-62.

1.2 initializing a central processing unit of said computer system;	Teoman, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

> "The program code is returned to the bus interface circuitry 61 which outputs it to the expansion bus where it is routed to its ultimate destination (e.g., the processor used to execute system boot code)."

Teoman, 7:61-64.

> "Still referring to FIG. 1, the processing unit 12 includes one or more processors that fetch program code from system memory 16 and execute the code to operate on data and to read and Write data to the system memory 16 and to the I/O devices on the expansion bus 18."

Teoman, 4:8-12.

> "In another embodiment, boot program code in the user cache firmware is executed to redirect boot time I/O requests before the operating system is loaded. In this way, boot time I/O requests are redirected to the user cache, making the user cache operable as a source of boot files during the entire boot sequence."

Teoman, 13:46-51.

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Teoman, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

> "When an I/O request 40 to access the mass storage 46 is issued (e.g., a file read or write request issued in the course of executing an application program), the I/O request 40 is first applied to the OS cache maintained in system memory 16. If the I/O request 40 hits the OS cache (i.e., the data sought to be accessed is cached in the OS cache), the access is performed in the OS cache. If the I/O request 40 is a read request, the data is returned to the requestor. If the I/O request 40 does not hit the OS cache, the I/O request 40 is redirected from the mass storage 46 to the user cache 25 by software mechanisms described below. If the I/O request 40 hits the user cache 25, the access is performed in the user cache 25 without having to access the mass storage 46, thereby substantially reducing the overall access time. Also, because the user cache 25 is significantly larger than the OS cache and supports data preloading (discussed below), much higher hit rates can be achieved in the user cache than in the OS cache."

Teoman, 4:57-5:7. *See also* Teoman, 5:16-20, 5:48-51.

> "According to one embodiment, the user cache 25 includes boot firmware 66 for storing program code that is used to support operation of the user cache at system startup."

Teoman, 7:50-52.

> "In general, there are two types of storage operations that take place in the user cache: preloading and responsive caching. Responsive caching refers to the storage of data in the user cache in response to I/O requests from application processes. In a preload operation, by contrast, data is retrieved from mass storage and stored in the user cache before being

requested for use in an application process."

Teoman, 9:24-31. *See also* Teoman, 9:31-44, 10:40-60.

> "The user interface 123 also permits the user to configure the computer system to use the user cache as a boot device, meaning that the user cache will be treated as a source of boot software at system startup. In one embodiment, if the user enables the "Use as a boot device" option, the user cache manager process prompts the user to specify the logical drive that is ordinarily the source of boot software. The user cache manager software responds by interacting with the BIOS configuration to treat the user-cache as the user-specified logical drive and thereby to boot out of the user-cache at system startup."

Teoman, 13:29-39.

> "In another embodiment, the user cache driver is loaded very early in the OS boot sequence and is operable for most of the sequence. Most operating systems support loading device drivers early in the boot sequence so that, in most cases, this mode of operation requires no special hardware or software support."

Teoman, 13:40-45.

> "In yet another embodiment for using the user cache to support system startup, before the OS boot sequence is begun, boot program code in the user cache firmware is executed to notify the system BIOS that the user cache is a bootable mass storage device. Also, when executed, the boot program code loads software into system memory for operating the user cache as a bootable mass storage device. In this embodiment, the user selects the user cache as the boot device in the system BIOS settings. Then, during the boot sequence, the operating system accesses boot up files in the user cache."

Teoman, 13:52-62.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Teoman, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

> "When an I/O request 40 to access the mass storage 46 is issued (e.g., a file read or write request issued in the course of executing an application program), the I/O request 40 is first applied to the OS cache maintained in system memory 16. If the I/O request 40 hits the OS cache (i.e., the data sought to be accessed is cached in the OS cache), the access is performed in the OS cache. If the I/O request 40 is a read request, the data is returned to the requestor. If the I/O request 40 does not hit the OS cache, the I/O request 40 is redirected from the mass storage 46 to the user cache 25 by software mechanisms described below. If the I/O request 40 hits the user cache 25, the access is performed in the user cache 25 without having to access the mass storage 46, thereby substantially reducing the overall access time. Also, because the user cache 25 is significantly larger than the OS cache and supports data preloading (discussed below), much higher hit rates can be achieved in the user cache than in the OS cache."

Teoman, 4:57-5:7. *See also* Teoman, 5:16-20, 5:48-51.

> "After a user has specified a set of commanded preload parameters, the user cache manager 90 responds by generating I/O requests to retrieve the data identified by the preload parameters from mass storage 46"

Teoman, 9:56-59.

> "In yet another embodiment for using the user cache to support system startup, before the OS boot sequence is begun, boot program code in the user cache firmware is executed to notify the system BIOS that the user cache is a bootable mass storage device. Also, when executed, the boot program code loads software into system memory for operating the user cache as a bootable mass storage device. In this embodiment, the user

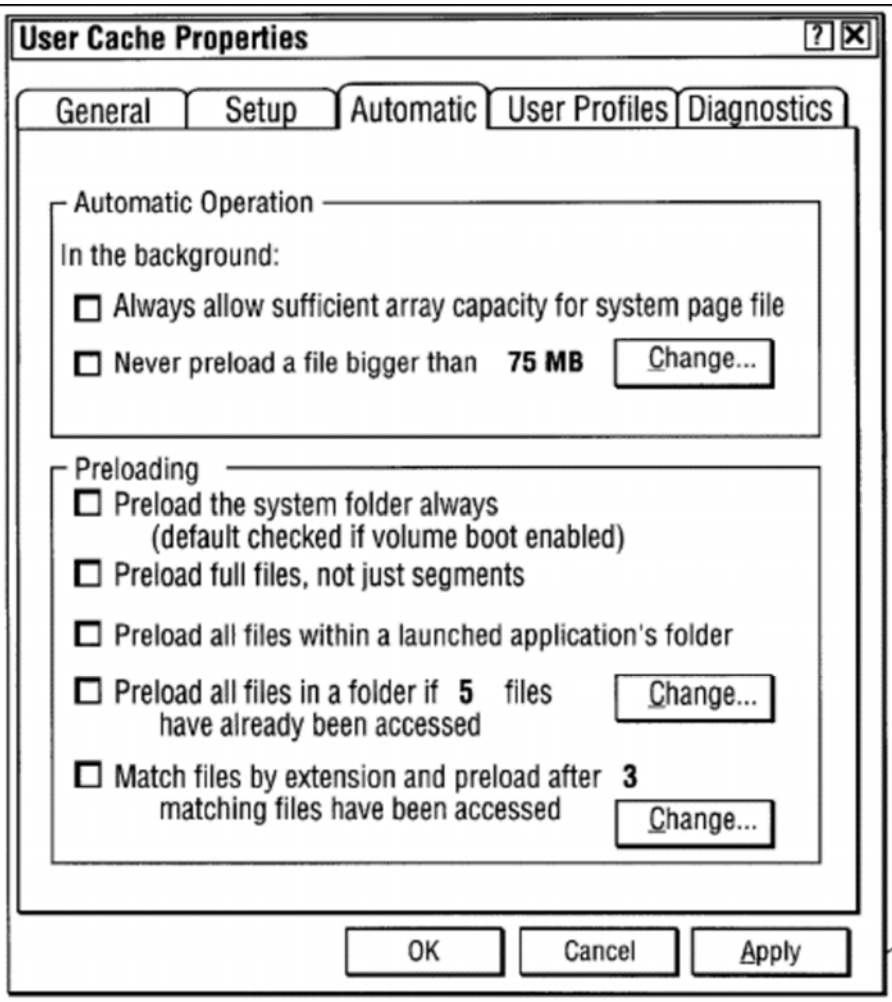
selects the user cache as the boot device in the system BIOS settings. Then, during the boot sequence, the operating system accesses boot up files in the user cache."

Teoman, 13:52-62.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Teoman, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Teoman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Teoman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also* ***Look for additional references to application programs***	



Teoman, Fig. 10.

> "In another embodiment, an application program called a user cache manager is executed to receive user preferences as to what data to store and not to store in the user cache."

Teoman, 3:18-20.

> "The program code in the system memory 16 includes operating system (OS) program code 30, application program code 34 and device driver program code 32. The application program code 34 is executed by the processing unit 12 to implement application processes which, in turn, invoke operating system services to display user-interfaces, operate on user-input and access user-specified data."

Teoman, 4:16-22.

Teoman Claim 3

"The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

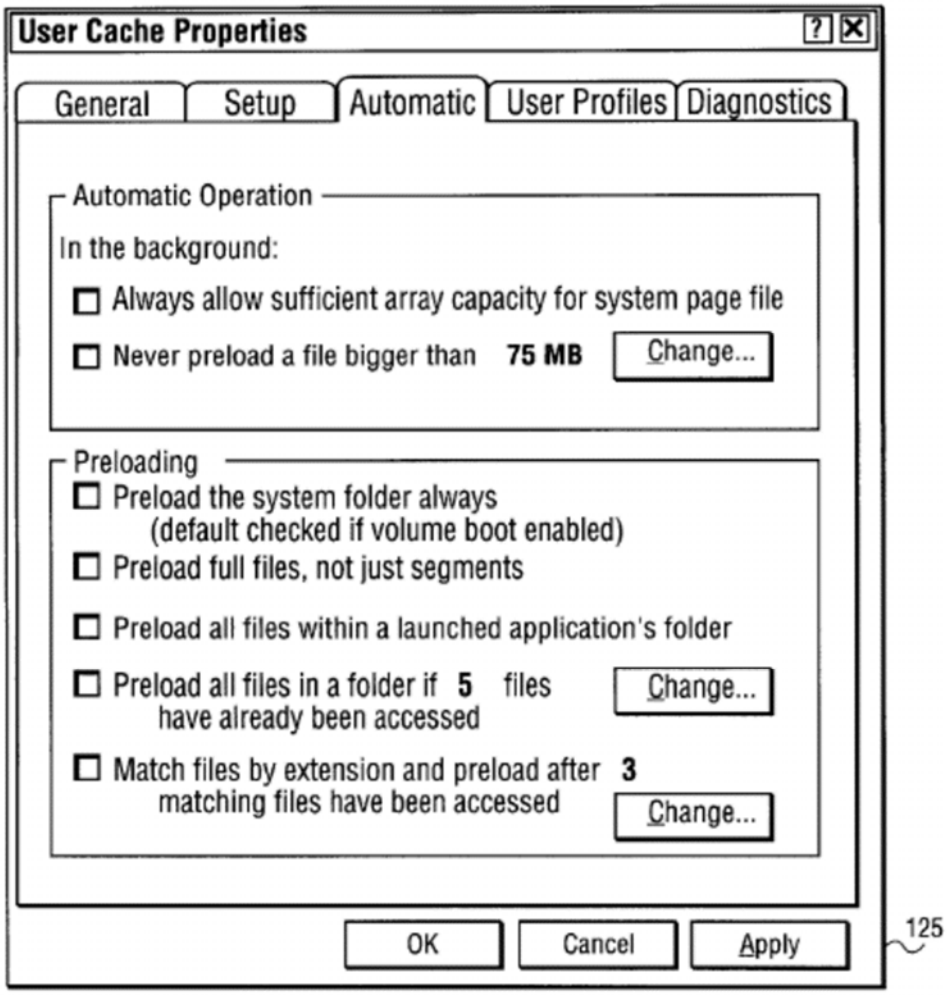
> "For example, when an application process invokes an operating system service to perform I/O to a device attached to the expansion bus 18, the operating system 30 invokes a standard routine Within the appropriate device driver 32 to carry out the requested I/O."

Teoman, 4:35-40.

> "When an I/O request 40 to access the mass storage 46 is issued (e.g., a ?le read or write request issued in the course of executing an application program), the I/O request 40 is first applied to the OS cache maintained in system memory 16."

Teoman, 4:57-61.

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Teoman, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above. *See also*	



Teoman, Fig. 10.

> "In another embodiment, an application program called a user cache manager is executed to receive user preferences as to what data to store and not to store in the user cache."

Teoman, 3:18-20.

> "The program code in the system memory 16 includes operating system (OS) program code 30, application program code 34 and device driver program code 32. The application program code 34 is executed by the processing unit 12 to implement application processes which, in turn, invoke operating system services to display user-interfaces, operate on user-input and access user-specified data."

Teoman, 4:16-22.

Teoman Claim 4

"The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

> "For example, when an application process invokes an operating system service to perform I/O to a device attached to the expansion bus 18, the operating system 30 invokes a standard routine Within the appropriate device driver 32 to carry out the requested I/O."

Teoman, 4:35-40.

> "When an I/O request 40 to access the mass storage 46 is issued (e.g., a ?le read or write request issued in the course of executing an application program), the I/O request 40 is first applied to the OS cache maintained in system memory 16."

Teoman, 4:57-61.

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Teoman, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The expansion bus 18 supports connection of a number of I/O devices including a self-buffered disk drive controller 20 and its associated disk drive 26, a non-buffered disk drive controller 22 and its associated disk drive 28, a network access device 24 such as a modem or local/Wide area network communications card and a user cache 25."

Teoman, 3:51-57.

> "The user-cache 25 includes bus inter face circuitry 61, a DRAM controller 63 and a plurality of rows (1 through N) of DRAM components 68A—68C, 69A—69C, 70A—70C."

Teoman, 6:16-19.

> "The address and control inputs of the DRAM components within a given row are coupled to a common address path and a common control path from the DRAM controller 63."

Teoman, 6:24-27.

> "In one embodiment, the power source selector 67 distinguishes between full system power and trickle power and asserts a sleep signal 81 to the DRAM controller 63 Whenever the user cache is being powered by trickle power or battery power. The DRAM controller 63 responds to the sleep signal 81 by issuing control signals to place the DRAM components of the user cache 25 in reduced power state. In the

> reduced power state, DRAM refresh operations are continued either under control of the DRAM controller 63 or by logic on board the DRAM components themselves. Other logic elements Within the user-cache 25, including the bus interface circuitry 61 and portions of the DRAM controller that operate on access requests from the bus interface circuitry are shut down to save power."
>
> Teoman, 7:18-32.

6. The method of claim 1, further comprising updating the list of boot data.	Teoman, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> "The requested blocks are then passed back to the user cache driver 45 which writes them to the user cache 25 and updates the user cache directory."

Teoman, 10:20-23.

> "After loading data into the user cache 25, the user cache driver 45 updates the user cache directory to indicate the newly cached blocks."

Teoman, 12:50-52.

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Teoman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Teoman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Teoman, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Teoman, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Teoman, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Teoman discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> "According to one embodiment, the user cache 25 is a non-volatile storage array that is used to cache data from mass storage devices, such as the local disk drives 26, 28 or disk drives on network servers 29A, 29B."

Teoman, 3:63-67.

> "The boot firmware 66 may be implemented using a number of different types of non-volatile memory including, but not limited to, programmable read only memory (PROM), erasable PROM (EPROM), electrically erasable PROM (EEPROM), flash EEPROM, and so forth."

Teoman, 7:64-8:3.

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Teoman, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Teoman, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Teoman, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Teoman, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Teoman, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B22
Invalidity of U.S. Patent 8,090,936 based on Teoman

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Teoman, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Teoman discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B23
Invalidity of U.S. Patent 8,090,936 based on Vers

German Patent No. DE19721786 to Michael Vers ("Vers") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Vers, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> The invention is based on the problem to provide a method for operating a data processing device in such a way that non-volatile at least consistent performance storage elements with a minimum memory capacity can be used. the problem is inventively solved in that the system software stored in compressed form in the non-volatile memory and to reset in response to a flag element either is decompressed and stored in the volatile memory or copied without decompression in the volatile memory and started there and / or that the application software associated, are compressed changes of application software files containing or by instructing from the programming device before power failure and stored in the nonvolatile memory and decompressed when power is restored to regenerate the files

in the volatile memories.

Vers, 2

Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

A particularly preferred compressing algorithm is the LZW algorithm used.

Vers, 2

In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

By a further storage area 36, the destination address is specified in the volatile memory to which the operating system copies or store decompressed. Finally, the tray of the operating system begins in compressed or transparent form.

Vers, 3

To inventively to guarantee the use of non-volatile memories of a minimum memory size is provided that the stored in compressed form in the non-volatile memory operating system during initialization in the volatile memory is loaded. For this purpose it is provided that the non-volatile memory compresses stored operating system, ie, when the flag element the value "ONE" has a function of the marker element, decompressed by a stored in nonvolatile memory uncompressed routine and is stored in the nonvolatile memory. Before the decompression of the operating system a CRC checksum is calculated and compared with the likewise in the header 28 stored in the storage area 30 by means of a checksum stored in the memory area 32 of the first header 28 length information. Is the calculated checksum matches the checksum stored, the process continues with the decompression. Here, the operating system is decompressed without the header 28 and stored decompressed to the stored in the memory area 36 of the header 28 destination address in the volatile memory. Does the marker element the value "zero" on, is the operating system directly, ie without decompression and without header in the volatile memory copied. After decompress or copying of the operating system as previously described a CRC checksum is determined and compared with a in a second header 40 in volatile memory which is present in decompressed form. If the two checksums match, a startup routine of the operating system is started, whose address is also stored in the header 40 in the storage area 46th

Vers, 3-4

1.2 initializing a central processing unit of said computer system;	Vers, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

> In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the

operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be. Vers, 3

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Vers, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> The invention is based on the problem to provide a method for operating a data processing device in such a way that non-volatile at least consistent performance storage elements with a minimum memory capacity can be used. the problem is inventively solved in that the system software stored in compressed form in the non-volatile memory and to reset in response to a flag element either is decompressed and stored in the volatile memory or copied without decompression in the volatile memory and started there and / or that the application software associated, are compressed changes of application software files containing or by instructing from the programming device before power failure and stored in the nonvolatile memory and decompressed when power is restored to regenerate the files in the volatile memories.

Vers, 2

> Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after

decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

A particularly preferred compressing algorithm is the LZW algorithm used.

Vers, 2

In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

By a further storage area 36, the destination address is specified in the volatile memory to which the operating system copies or store decompressed. Finally, the tray of the operating system begins in compressed or transparent form.

Vers, 3

To inventively to guarantee the use of non-volatile memories of a minimum memory size is provided that the stored in compressed form in the non-volatile memory operating system during initialization in the volatile memory is loaded. For this purpose it is provided that the non-volatile memory compresses stored operating system, ie, when the flag element the value "ONE" has a function of the marker element, decompressed by a stored in nonvolatile memory uncompressed routine

and is stored in the nonvolatile memory. Before the decompression of the operating system a CRC checksum is calculated and compared with the likewise in the header 28 stored in the storage area 30 by means of a checksum stored in the memory area 32 of the first header 28 length information. Is the calculated checksum matches the checksum stored, the process continues with the decompression. Here, the operating system is decompressed without the header 28 and stored decompressed to the stored in the memory area 36 of the header 28 destination address in the volatile memory. Does the marker element the value "zero" on, is the operating system directly, ie without decompression and without header in the volatile memory copied. After decompress or copying of the operating system as previously described a CRC checksum is determined and compared with a in a second header 40 in volatile memory which is present in decompressed form. If the two checksums match, a startup routine of the operating system is started, whose address is also stored in the header 40 in the storage area 46th

Vers, 3-4

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Vers, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> The invention is based on the problem to provide a method for operating a data processing device in such a way that non-volatile at least consistent performance storage elements with a minimum memory capacity can be used. the problem is inventively solved in that the system software stored in compressed form in the non-volatile memory and to reset in response to a flag element either is decompressed and stored in the volatile memory or copied without decompression in the volatile memory and started there and / or that the application software associated, are compressed changes of application software files containing or by instructing from the programming device before power failure and stored in the nonvolatile memory and decompressed when power is restored to regenerate the files in the volatile memories.

Vers, 2

> Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC

checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

A particularly preferred compressing algorithm is the LZW algorithm used.

Vers, 2

In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

By a further storage area 36, the destination address is specified in the volatile memory to which the operating system copies or store decompressed. Finally, the tray of the operating system begins in compressed or transparent form.

Vers, 3

To inventively to guarantee the use of non-volatile memories of a minimum memory size is provided that the stored in compressed form in the non-volatile memory operating system during initialization in the volatile memory is loaded. For this purpose it is provided that the non-volatile memory compresses stored operating system, ie, when the flag element the value "ONE" has a function of the marker element, decompressed by a stored in nonvolatile memory uncompressed routine and is stored in the nonvolatile memory. Before the decompression of the

operating system a CRC checksum is calculated and compared with the likewise in the header 28 stored in the storage area 30 by means of a checksum stored in the memory area 32 of the first header 28 length information. Is the calculated checksum matches the checksum stored, the process continues with the decompression. Here, the operating system is decompressed without the header 28 and stored decompressed to the stored in the memory area 36 of the header 28 destination address in the volatile memory. Does the marker element the value "zero" on, is the operating system directly, ie without decompression and without header in the volatile memory copied. After decompress or copying of the operating system as previously described a CRC checksum is determined and compared with a in a second header 40 in volatile memory which is present in decompressed form. If the two checksums match, a startup routine of the operating system is started, whose address is also stored in the header 40 in the storage area 46th

Vers, 3-4

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Vers, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> The invention is based on the problem to provide a method for operating a data processing device in such a way that non-volatile at least consistent performance storage elements with a minimum memory capacity can be used. the problem is inventively solved in that the system software stored in compressed form in the non-volatile memory and to reset in response to a flag element either is decompressed and stored in the volatile memory or copied without decompression in the volatile memory and started there and / or that the application software associated, are compressed changes of application software files containing or by instructing from the programming device before power failure and stored in the nonvolatile memory and decompressed when power is restored to regenerate the files in the volatile memories.

Vers, 2

> Preferably, a destination address in the volatile memory is in the header

in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

A particularly preferred compressing algorithm is the LZW algorithm used.

Vers, 2

In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

By a further storage area 36, the destination address is specified in the volatile memory to which the operating system copies or store decompressed. Finally, the tray of the operating system begins in compressed or transparent form.

Vers, 3

To inventively to guarantee the use of non-volatile memories of a minimum memory size is provided that the stored in compressed form in the non-volatile memory operating system during initialization in the volatile memory is loaded. For this purpose it is provided that the non-volatile memory compresses stored operating system, ie, when the flag

Vers Claim 1.5

"utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

element the value "ONE" has a function of the marker element, decompressed by a stored in nonvolatile memory uncompressed routine and is stored in the nonvolatile memory. Before the decompression of the operating system a CRC checksum is calculated and compared with the likewise in the header 28 stored in the storage area 30 by means of a checksum stored in the memory area 32 of the first header 28 length information. Is the calculated checksum matches the checksum stored, the process continues with the decompression. Here, the operating system is decompressed without the header 28 and stored decompressed to the stored in the memory area 36 of the header 28 destination address in the volatile memory. Does the marker element the value "zero" on, is the operating system directly, ie without decompression and without header in the volatile memory copied. After decompress or copying of the operating system as previously described a CRC checksum is determined and compared with a in a second header 40 in volatile memory which is present in decompressed form. If the two checksums match, a startup routine of the operating system is started, whose address is also stored in the header 40 in the storage area 46th

Vers, 3-4

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Vers, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Vers, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The invention relates to a method for operating a data processing device, in particular automation device, with volatile and non-volatile memories, said system and / or application software is copied from non-volatile to a volatile memory.

Vers, 1

> From the prior art it is known that in a data processing device, both the system and the application software is stored in a nonvolatile memory. The system software can run directly on the one hand from the non-volatile memory or on the other hand copies of the non-volatile memory to the volatile memory and run from this. The start of the system software from non-volatile memory has the disadvantage that either a quicker and thereby more expensive non-volatile memory must be used, or that the system power is discontinued by inserting wait states ago. At the start of the system software from the volatile memory non-volatile memory must be in full size of the system software, however, is no longer needed after copying the system software.

Vers, 1

> The problem is inventively achieved in that the system software stored compression form in the in the non-volatile memory after reset depending on a flag element either is decompressed and stored in the volatile memory or copied without decompression in the volatile memory and

started there and / or that the application software associated, are compressed files containing changes the user software or by appointment by the programmer before power failure and stored in non-volatile memory and decompressed when power is restored to regenerate the files in the volatile memories.

Vers 2

In particular, programmable logic controllers or PLCs can be stored in non-volatile memory in compressed form and application software.

Vers, 3

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Vers, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The invention relates to a method for operating a data processing device, in particular automation device, with volatile and non-volatile memories, said system and / or application software is copied from non-volatile to a volatile memory.

Vers, 1

> From the prior art it is known that in a data processing device, both the system and the application software is stored in a nonvolatile memory. The system software can run directly on the one hand from the non-volatile memory or on the other hand copies of the non-volatile memory to the volatile memory and run from this. The start of the system software from non-volatile memory has the disadvantage that either a quicker and thereby more expensive non-volatile memory must be used, or that the system power is discontinued by inserting wait states ago. At the start of the system software from the volatile memory non-volatile memory must be in full size of the system software, however, is no longer needed after copying the system software.

Vers, 1

> The problem is inventively achieved in that the system software stored compression form in the in the non-volatile memory after reset depending on a flag element either is decompressed and stored in the volatile

memory or copied without decompression in the volatile memory and started there and / or that the application software associated, are compressed files containing changes the user software or by appointment by the programmer before power failure and stored in non-volatile memory and decompressed when power is restored to regenerate the files in the volatile memories.

Vers 2

Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

In particular, programmable logic controllers or PLCs can be stored in non-volatile memory in compressed form and application software.

Vers, 3

In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

Vers Claim 4

"The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Vers, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> In particular, programmable logic controllers or PLCs can be stored in non-volatile memory in compressed form and application software.

Vers, 3

> To not use volatile memory with a small storage capacity possible is, the invention proposes that an art compression routine at Beauf transfer compressed by a programmer the latch and the HEAP a memory management and stored in the nonvolatile memory 16 and when the PLC again is turned on, a Dekomprimierroutine is activated, which regenerates the signal memory, and the entire heap of memory management of the nonvolatile memory to the volatile memory. Stay All online changes obtained so that a further change of user program when Spannungsaus case can be avoided. In addition, while minimizing the hardware expense, the functionality of the programmable controller expanded.

Vers 4

6. The method of claim 1, further comprising updating the list of boot data.	Vers, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> Preferably, a destination address in the volatile memory is in the header in the non-volatile memory, where the system software such as operating system copied or stored decompressed. It is also provided that after decompression or copying of the system software calculated CRC checksum compared with a in a second header in the region of the beginning of the file system software in the volatile memory, and if they match a start routine is started.

Vers, 2

> In Fig. 1 the basic construction of a programmable controller is shown with a bus 12 to which a microprocessor 14, non-volatile and volatile memories 16, 18 and one or more peripheral devices 20 are connected. as non-volatile memory 16 preferably flash memory modules are used, which can be written and read, and in contrast to the volatile memory 18 may be retained their content when they are not supplied with operating voltage. Other non-volatile memory such as EPROM and EEPROM can also be used. Preferably as a volatile memory SRAM and / or DRAM's

are used. In Fig. 2, the memory map of the nonvolatile memory 16 is shown. According to the invention, that the system software as the operating system stored in an initial region 22nd Following the operating system is followed by a free space 24, which is a storage area 26 connects with a reset initialization and a decompression algorithm as executable code. Especially with programmable logic controllers or PLCs can also use the software in the nonvolatile memory is stored in compressed form be.

Vers, 3

By a further storage area 36, the destination address is specified in the volatile memory to which the operating system copies or store decompressed. Finally, the tray of the operating system begins in compressed or transparent form.

Vers, 3

If a user software from the volatile memory operates 18 may at any time provide the required memory locations for storing online modifications available memory management. For an online modified user program is again even with reclosing of the automation device in the previously amended version is available, it must be stored before switching off the automation device in the non-volatile memory, so that all previous changes and variable initialization are stored.

Vers, 4

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Vers, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

> A particularly preferred compressing algorithm is the LZW algorithm used.

Vers, 2

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Vers, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Vers, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Vers, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Vers, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Vers discloses this limitation:

See Claims 1.1, 1.3, 1.4 above.

See also

> The method involves placing system software into a non-volatile memory (16) in compressed form. After a reset, the software is copied into the volatile memory (18) and is started there. This is done either after decompression or without decompression, depending on the characteristic element. Data corresponding to changes in the user software prior to voltage drop-out are compressed and stored in the non-volatile memory and after reset are decompressed for regeneration of the data in volatile memory.

Vers, Abstract

> From the prior art it is known that in a data processing device, both the system and the application software is stored in a nonvolatile memory. The system software can run directly on the one hand from the non-volatile memory or on the other hand copies of the non-volatile memory to the volatile memory and run from this. The start of the system software from non-volatile memory has the disadvantage that either a quicker and

thereby more expensive non-volatile memory must be used, or that the system power is discontinued by inserting wait states ago. At the start of the system software from the volatile memory non-volatile memory must be in full size of the system software, however, is no longer needed after copying the system software.

It is also known that application software can run in non-volatile memory or can be loaded from a programming unit via a controlled by the system software programming in the volatile memory directly. While executing the application software from non-volatile memory often occurs a problem that the application software on-line changes are difficult to achieve or only with a considerable increase in the cycle time. In a sequence of user software from the volatile memory is also known that a memory management can always provide the necessary resources for online changes available. If the modified data upon reconnection of the data processing apparatus provided in its amended form available to use must be assured that they were previously stored in non-volatile memory so that the previous online changes are saved. However, this would require an unnecessarily large non-volatile memory which would not be used during the lifetime of the data processing device.

Vers, 1-2

To inventively to guarantee the use of non-volatile memories of a minimum memory size is provided that the stored in compressed form in the non-volatile memory operating system during initialization in the volatile memory is loaded. For this purpose it is provided that the non-volatile memory compresses stored operating system, ie, when the flag element the value "ONE" has a function of the marker element, decompressed by a stored in nonvolatile memory uncompressed routine and is stored in the nonvolatile memory. Before the decompression of the operating system a CRC checksum is calculated and compared with the likewise in the header 28 stored in the storage area 30 by means of a checksum stored in the memory area 32 of the first header 28 length information. Is the calculated checksum matches the checksum stored, the process continues with the decompression. Here, the operating system is decompressed without the header 28 and stored decompressed to the stored in the memory area 36 of the header 28 destination address in the volatile memory. Does the marker element the value "zero" on, is the operating system directly, ie without decompression and without header in the volatile memory copied. After decompress or copying of the operating system as previously described a CRC checksum is determined and compared with a in a second header 40 in volatile memory which is present in decompressed form. If the two checksums match, a startup

routine of the operating system is started, whose address is also stored in the header 40 in the storage area 46th

Vers, 3-4

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Vers, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Vers, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Vers, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Vers, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Vers, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Vers, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Vers discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B24
Invalidity of U.S. Patent 8,090,936 based on Zwiegincew

U.S. Patent No. 6,317,818 to Zwiegincew ("Zwiegincew") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

In addition, Apple incorporates by reference, as if set forth fully herein, all arguments related to Zwiegincew in pending inter partes review petitions IPR2016-01737, IPR2016-01738, and IPR2016-01739.

Appendix B24
Invalidity of U.S. Patent 8,090,936 based on Zwiegincew

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Zwiegincew, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Zwiegincew discloses this limitation:

> Hard page fault patterns of an application program module are analyzed in order to determine the pages that will be retrieved from disk storage during a common hard page fault scenario. Copies of, or references to, the determined pages are stored in a scenario file, along with an index referred to as a page sequence. The scenario file may also include a prologue indicating events that lead to a hard page fault scenario and an epilogue that may indicate subsequent hard page fault scenarios. Execution of the application program module is monitored to detect the occurrence of a hard page fault scenario. When a hard page fault scenario is detected, a corresponding scenario file is fetched from disk storage and the determined pages, or copies thereof, are transferred into RAM. The determined pages, or copies thereof, may be placed on a stand-by list in RAM and later soft-faulted into the working set of the application program upon request by the application program module, thereby avoiding a sequence of hard page faults.

Zwiegincew, Abstract

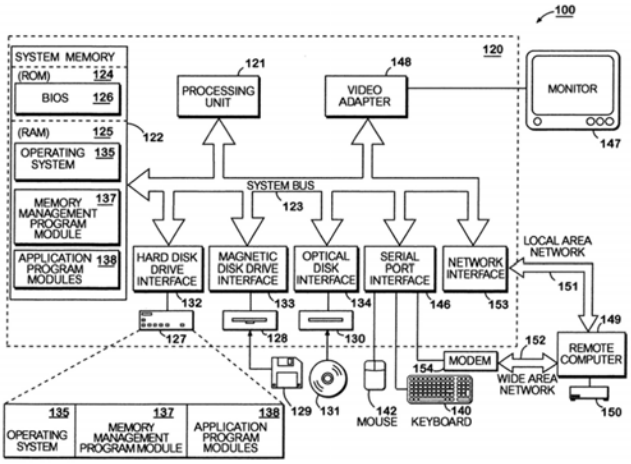


FIG.1

Zwiegincew, Fig. 1

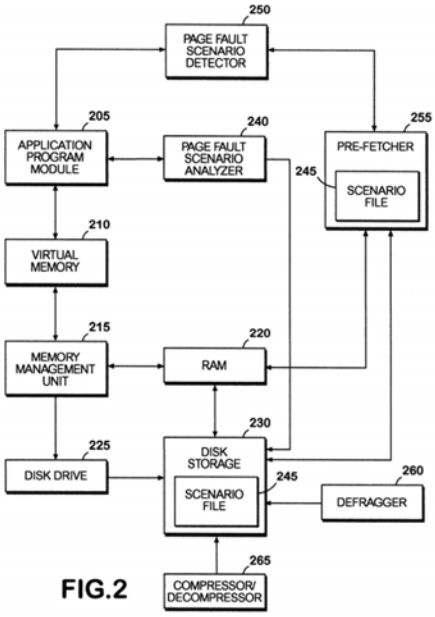


FIG.2

Zwiegincew. Fig 2

There are various potential solutions to the performance bottleneck caused by disk access time during hard page fault scenarios. An obvious potential solution is to reduce disk access time. The reduction of disk access time is primarily a hardware consideration and is not easily

> accomplished. However, other potential solutions involve the manipulation of memory storage through software program modules.

Zwiegincew, 1:45-51

> In an exemplary embodiment, the present invention may comprise one or more memory management program modules 137 stored on the drives or RAM 125 of the computer 100. Specifically, program modules 137 of the present invention may comprise computer implemented instructions for determining which pages will have to be retrieved from disk during a potential hard page fault scenario and pre-fetching the determined pages into RAM prior to the occurrence of the potential hard page fault sequence.

Zwiegincew, 5:50-51.

> The present invention meets the needs described above by providing a system and method for improving the performance of an application program module by reducing the occurrence of hard page faults during the operation of an application program module. The present invention may be embodied in an add-on software program module that operates in conjunction with the application program module. In this manner, no effort is required on the part of the application programmer to manipulate or modify the application program module in order to improve performance. Furthermore, the add-on software program module does not detract from the intended operation of the application program module.
>
> According to one aspect of the present invention, a scenario file is created which comprises ordered copies of pages that are likely to be retrieved from disk storage by an application program module during a hard page fault. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the scenario file is fetched from disk storage and the ordered copies of the pages are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. In another aspect of the invention, a hard page fault scenario analyzer is provided for analyzing a hard page fault scenario of an application program module in order to

> determine the pages that will be retrieved from disk storage upon the occurrence of a hard page fault scenario. The hard page fault scenario analyzer creates a scenario file comprising copies of the pages in the determined order. A hard page fault scenario detector is provided for monitoring the execution of the application module, detecting a hard page fault scenario and sending a message to a pre-fetcher. The hard page fault scenario detector may be manual or automatic. A pre-fetcher retrieves a scenario file from disk storage and transfers the copies of the determined pages into RAM. The copies of the determined pages are placed on a standby list in RAM. Accordingly, the determined pages will be available in RAM during a hard page fault scenario and will be soft-faulted into the working set of the application program module when they are requested by the application program module, thereby avoiding a hard page fault.
>
> According to another aspect of the invention, a scenario file is created which comprises ordered references to pages that are likely to be retrieved from disk storage by an application program module during a hard page fault scenario, rather than the actual pages themselves. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the pages referenced by the scenario file are fetched from disk storage in the optimal manner and are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. This aspect of the invention will result in more seek operations on disk, but will still allow reading of the required pages in an optimal manner, rather than the 'as needed' ordering if the pages are hard faulted into RAM. This aspect of the invention also reduces the disk space requirements over the previously mentioned aspect.

Zwiegincew, 2:43-3:49.

> A hard page fault scenario analyzer 240 anticipates and analyzes hard page fault scenarios. As mentioned, a hard page fault scenario is a situation in which a hard page fault sequence is highly likely to occur. The hard page fault scenario analyzer logs various hard page fault scenarios that occur during operation of the application program module 205. The logged hard page fault scenarios are then analyzed to

> determine if they re-occur frequently, and if they do, they are put in a scenario file. This analysis can occur programmatically on the end-user's computer system, or in advance by the developer of a particular software product. As an example, suppose the application program module 205 is a word processor and that an anticipated hard page fault scenario is the situation in which the user selects a well known "file open" command. In response to the "file open" command, the application program will display a graphical representation of a file directory. However, in order to display the graphical representation of the file directory, a sequence of hard page faults will occur because the word processor must retrieve a particular set of pages from disk. In accordance with an exemplary embodiment of the present invention, the hard page fault scenario analyzer 240 anticipates the "open file" hard page fault scenario of the example and determines the set of pages that will need to be retrieved from disk upon the occurrence of the hard page fault. The determination of pages that will need to be retrieved from disk will be described in greater detail below. The detection of particular classes of hard page fault scenarios may be built into the system. For example, application launch is virtually always a hard page fault scenario, so an exemplary embodiment of the present invention may be configured such that any launch operation of an application program will be considered to be a hard page fault scenario.

Zwiegincew, 6:29-61.

> The hard page fault scenario analyzer 240 may comprise functionality for automatically analyzing hard page fault scenarios and generating corresponding scenario files. By way of illustration, the hard page fault analyzer 240 may log hard page faults that occur upon execution of a process during operation of an application program module 205. During idle time of the application program module 205, the hard page fault scenario analyzer 240 may write the log of hard page faults to a log file. Then, a pattern matching algorithm may be used to find a pattern of hard page faults based on all log files generated for the process same. If a pattern of hard page faults is found, a new scenario file may be generated based on the pages that are retrieved from disk during the pattern. Automatically generated scenario files may be subject to subsequent refinement, i.e., they may be input into the pattern-matching algorithm.

Zwiegincew, 7:24-40

See also Zwiegincew. 1:5-2:40, Figs 1-2

1.2 initializing a central processing unit of said computer system;	Zwiegincew, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Zwiegincew, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Zwiegincew discloses this limitation:

> Hard page fault patterns of an application program module are analyzed in order to determine the pages that will be retrieved from disk storage during a common hard page fault scenario. Copies of, or references to, the determined pages are stored in a scenario file, along with an index referred to as a page sequence. The scenario file may also include a prologue indicating events that lead to a hard page fault scenario and an epilogue that may indicate subsequent hard page fault scenarios. Execution of the application program module is monitored to detect the occurrence of a hard page fault scenario. When a hard page fault scenario is detected, a corresponding scenario file is fetched from disk storage and the determined pages, or copies thereof, are transferred into RAM. The determined pages, or copies thereof, may be placed on a stand-by list in RAM and later soft-faulted into the working set of the application program upon request by the application program module, thereby avoiding a sequence of hard page faults.

Zwiegincew, Abstract

> There are various potential solutions to the performance bottleneck caused by disk access time during hard page fault scenarios. An obvious potential solution is to reduce disk access time. The reduction of disk access time is primarily a hardware consideration and is not easily accomplished. However, other potential solutions involve the manipulation of memory storage through software program modules.

Zwiegincew, 1:45-51

> In an exemplary embodiment, the present invention may comprise one or more memory management program modules 137 stored on the

> drives or RAM 125 of the computer 100. Specifically, program modules 137 of the present invention may comprise computer implemented instructions for determining which pages will have to be retrieved from disk during a potential hard page fault scenario and pre-fetching the determined pages into RAM prior to the occurrence of the potential hard page fault sequence.

Zwiegincew, 5:50-51.

> The present invention meets the needs described above by providing a system and method for improving the performance of an application program module by reducing the occurrence of hard page faults during the operation of an application program module. The present invention may be embodied in an add-on software program module that operates in conjunction with the application program module. In this manner, no effort is required on the part of the application programmer to manipulate or modify the application program module in order to improve performance. Furthermore, the add-on software program module does not detract from the intended operation of the application program module.
>
> According to one aspect of the present invention, a scenario file is created which comprises ordered copies of pages that are likely to be retrieved from disk storage by an application program module during a hard page fault. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the scenario file is fetched from disk storage and the ordered copies of the pages are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. In another aspect of the invention, a hard page fault scenario analyzer is provided for analyzing a hard page fault scenario of an application program module in order to determine the pages that will be retrieved from disk storage upon the occurrence of a hard page fault scenario. The hard page fault scenario analyzer creates a scenario file comprising copies of the pages in the determined order. A hard page fault scenario detector is provided for monitoring the execution of the application module, detecting a hard page fault scenario and sending a message to a pre-fetcher. The hard page fault scenario detector may be manual or automatic. A pre-fetcher

> retrieves a scenario file from disk storage and transfers the copies of the determined pages into RAM. The copies of the determined pages are placed on a standby list in RAM. Accordingly, the determined pages will be available in RAM during a hard page fault scenario and will be soft-faulted into the working set of the application program module when they are requested by the application program module, thereby avoiding a hard page fault.
>
> According to another aspect of the invention, a scenario file is created which comprises ordered references to pages that are likely to be retrieved from disk storage by an application program module during a hard page fault scenario, rather than the actual pages themselves. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the pages referenced by the scenario file are fetched from disk storage in the optimal manner and are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. This aspect of the invention will result in more seek operations on disk, but will still allow reading of the required pages in an optimal manner, rather than the 'as needed' ordering if the pages are hard faulted into RAM. This aspect of the invention also reduces the disk space requirements over the previously mentioned aspect.

Zwiegincew, 2:43-3:49.

See also Zwiegincew. 1:5-2:40, Figs 1-2

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Zwiegincew, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Zwiegincew discloses this limitation:

> Hard page fault patterns of an application program module are analyzed in order to determine the pages that will be retrieved from disk storage during a common hard page fault scenario. Copies of, or references to, the determined pages are stored in a scenario file, along with an index referred to as a page sequence. The scenario file may also include a prologue indicating events that lead to a hard page fault scenario and an epilogue that may indicate subsequent hard page fault scenarios. Execution of the application program module is monitored to detect the occurrence of a hard page fault scenario. When a hard page fault scenario is detected, a corresponding scenario file is fetched from disk storage and the determined pages, or copies thereof, are transferred into RAM. The determined pages, or copies thereof, may be placed on a stand-by list in RAM and later soft-faulted into the working set of the application program upon request by the application program module, thereby avoiding a sequence of hard page faults.

Zwiegincew, Abstract

> There are various potential solutions to the performance bottleneck caused by disk access time during hard page fault scenarios. An obvious potential solution is to reduce disk access time. The reduction of disk access time is primarily a hardware consideration and is not easily accomplished. However, other potential solutions involve the manipulation of memory storage through software program modules.

Zwiegincew, 1:45-51

> In an exemplary embodiment, the present invention may comprise one or more memory management program modules 137 stored on the drives or RAM 125 of the computer 100. Specifically, program modules

> 137 of the present invention may comprise computer implemented instructions for determining which pages will have to be retrieved from disk during a potential hard page fault scenario and pre-fetching the determined pages into RAM prior to the occurrence of the potential hard page fault sequence.

Zwiegincew, 5:50-51.

> The present invention meets the needs described above by providing a system and method for improving the performance of an application program module by reducing the occurrence of hard page faults during the operation of an application program module. The present invention may be embodied in an add-on software program module that operates in conjunction with the application program module. In this manner, no effort is required on the part of the application programmer to manipulate or modify the application program module in order to improve performance. Furthermore, the add-on software program module does not detract from the intended operation of the application program module.
>
> According to one aspect of the present invention, a scenario file is created which comprises ordered copies of pages that are likely to be retrieved from disk storage by an application program module during a hard page fault. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the scenario file is fetched from disk storage and the ordered copies of the pages are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. In another aspect of the invention, a hard page fault scenario analyzer is provided for analyzing a hard page fault scenario of an application program module in order to determine the pages that will be retrieved from disk storage upon the occurrence of a hard page fault scenario. The hard page fault scenario analyzer creates a scenario file comprising copies of the pages in the determined order. A hard page fault scenario detector is provided for monitoring the execution of the application module, detecting a hard page fault scenario and sending a message to a pre-fetcher. The hard page fault scenario detector may be manual or automatic. A pre-fetcher retrieves a scenario file from disk storage and transfers the copies of the

> determined pages into RAM. The copies of the determined pages are placed on a standby list in RAM. Accordingly, the determined pages will be available in RAM during a hard page fault scenario and will be soft-faulted into the working set of the application program module when they are requested by the application program module, thereby avoiding a hard page fault.
>
> According to another aspect of the invention, a scenario file is created which comprises ordered references to pages that are likely to be retrieved from disk storage by an application program module during a hard page fault scenario, rather than the actual pages themselves. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the pages referenced by the scenario file are fetched from disk storage in the optimal manner and are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. This aspect of the invention will result in more seek operations on disk, but will still allow reading of the required pages in an optimal manner, rather than the 'as needed' ordering if the pages are hard faulted into RAM. This aspect of the invention also reduces the disk space requirements over the previously mentioned aspect.

Zwiegincew, 2:43-3:49.

> Further, an exemplary embodiment includes a disk compressor/decompressor 265. Well known compression algorithms may be employed to achieve approximately 50% compression with 25 MB/s decompression throughput. These results may be achieved with as little as 64 KB extra memory. Average disk transfer rates are about 8 MB/s. So, for an illustrative 3 MB pre-fetch scenario, comparative pre-fetch times are as follows:
>
> No compression: 0.012 s (seek)+3 MB/8 MB/s (read)=0.3870 s.
>
> 50% compression: 0.012 s (seek)+1.5 MB/8 MB/s (read)+3 MB/25 MB/s (decompress)=0.3195 s.

> Thus, there is a 17.5% improvement in pre-fetch time using 50% compression.

Zwiegincew, 8:66-9:13

See also Zwiegincew. 1:5-2:40, Figs 1-2

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Zwiegincew, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Zwiegincew discloses this limitation:

> Hard page fault patterns of an application program module are analyzed in order to determine the pages that will be retrieved from disk storage during a common hard page fault scenario. Copies of, or references to, the determined pages are stored in a scenario file, along with an index referred to as a page sequence. The scenario file may also include a prologue indicating events that lead to a hard page fault scenario and an epilogue that may indicate subsequent hard page fault scenarios. Execution of the application program module is monitored to detect the occurrence of a hard page fault scenario. When a hard page fault scenario is detected, a corresponding scenario file is fetched from disk storage and the determined pages, or copies thereof, are transferred into RAM. The determined pages, or copies thereof, may be placed on a stand-by list in RAM and later soft-faulted into the working set of the application program upon request by the application program module, thereby avoiding a sequence of hard page faults.

Zwiegincew, Abstract

> There are various potential solutions to the performance bottleneck caused by disk access time during hard page fault scenarios. An obvious potential solution is to reduce disk access time. The reduction of disk access time is primarily a hardware consideration and is not easily

> accomplished. However, other potential solutions involve the manipulation of memory storage through software program modules.

Zwiegincew, 1:45-51

> In an exemplary embodiment, the present invention may comprise one or more memory management program modules 137 stored on the drives or RAM 125 of the computer 100. Specifically, program modules 137 of the present invention may comprise computer implemented instructions for determining which pages will have to be retrieved from disk during a potential hard page fault scenario and pre-fetching the determined pages into RAM prior to the occurrence of the potential hard page fault sequence.

Zwiegincew, 5:50-51.

> The present invention meets the needs described above by providing a system and method for improving the performance of an application program module by reducing the occurrence of hard page faults during the operation of an application program module. The present invention may be embodied in an add-on software program module that operates in conjunction with the application program module. In this manner, no effort is required on the part of the application programmer to manipulate or modify the application program module in order to improve performance. Furthermore, the add-on software program module does not detract from the intended operation of the application program module.
>
> According to one aspect of the present invention, a scenario file is created which comprises ordered copies of pages that are likely to be retrieved from disk storage by an application program module during a hard page fault. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the scenario file is fetched from disk storage and the ordered copies of the pages are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. In another aspect of the invention, a hard page fault scenario analyzer is provided for analyzing a hard page fault scenario of an application program module in order to

> determine the pages that will be retrieved from disk storage upon the occurrence of a hard page fault scenario. The hard page fault scenario analyzer creates a scenario file comprising copies of the pages in the determined order. A hard page fault scenario detector is provided for monitoring the execution of the application module, detecting a hard page fault scenario and sending a message to a pre-fetcher. The hard page fault scenario detector may be manual or automatic. A pre-fetcher retrieves a scenario file from disk storage and transfers the copies of the determined pages into RAM. The copies of the determined pages are placed on a standby list in RAM. Accordingly, the determined pages will be available in RAM during a hard page fault scenario and will be soft-faulted into the working set of the application program module when they are requested by the application program module, thereby avoiding a hard page fault.
>
> According to another aspect of the invention, a scenario file is created which comprises ordered references to pages that are likely to be retrieved from disk storage by an application program module during a hard page fault scenario, rather than the actual pages themselves. The scenario file is stored in the disk storage. Then, the execution of the application program module is monitored until either an explicit begin-of-scenario instruction is detected, or a hard page fault scenario is detected. A hard page fault scenario may comprise any situation or event that is likely to trigger a hard page fault, i.e., one or more requested pages will not be available in RAM and will be retrieved from disk storage. In response to the detection of a begin-of-scenario instruction or a hard page fault scenario, the pages referenced by the scenario file are fetched from disk storage in the optimal manner and are transferred into a standby list in RAM. In this manner, the requested pages will be soft faulted into a working set of the application program module, and no hard page fault will occur. This aspect of the invention will result in more seek operations on disk, but will still allow reading of the required pages in an optimal manner, rather than the 'as needed' ordering if the pages are hard faulted into RAM. This aspect of the invention also reduces the disk space requirements over the previously mentioned aspect.

Zwiegincew, 2:43-3:49.

> Further, an exemplary embodiment includes a disk compressor/decompressor 265. Well known compression algorithms may be employed to achieve approximately 50% compression with 25 MB/s decompression throughput. These results may be achieved with as little as 64 KB extra memory. Average disk transfer rates are about 8

MB/s. So, for an illustrative 3 MB pre-fetch scenario, comparative pre-fetch times are as follows:

No compression: 0.012 s (seek)+3 MB/8 MB/s (read)=0.3870 s.

50% compression: 0.012 s (seek)+1.5 MB/8 MB/s (read)+3 MB/25 MB/s (decompress)=0.3195 s.

Thus, there is a 17.5% improvement in pre-fetch time using 50% compression.

Zwiegincew, 8:66-9:13

See also Zwiegincew. 1:5-2:40, Figs 1-2

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Zwiegincew, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Zwiegincew, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Zwiegincew, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Zwiegincew, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Zwiegincew, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Zwiegincew, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Zwiegincew, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Zwiegincew, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

11.2. a memory; and	Zwiegincew, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Zwiegincew, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Zwiegincew, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Zwiegincew, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Zwiegincew, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Zwiegincew, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Zwiegincew, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1, 8, and 11 above.	

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Zwiegincew, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Zwiegincew discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B25
Invalidity of U.S. Patent 8,090,936 based on Anyimi

The publication Anyimi, "Implementing a Plug and Play BIOS Using Intel's Boot Block Flash Memory," Feb. 1995 ("Anyimi") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B25
Invalidity of U.S. Patent 8,090,936 based on Anyimi

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Anyimi, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this claim limitation:

Plug and Play can make the usual experience of adding new functionality to an existing system as easy as:

1. Turn the system off.
2. Insert the new device.
3. Turn the system on.

PnP flash BIOS can also extend the life of the PC. By enabling simple updates to the BIOS (simply insert the upgrade disk and the software programs the new BIOS), the user can get more out of their PC investment.

Anyimi, 1.0

3.1 PnP Components

For a system to be fully PnP-compliant, four basic elements are required:

1. System/PnP BIOS
2. PnP operating system
3. PnP hardware devices
4. PnP application software

Anyimi, 3.1

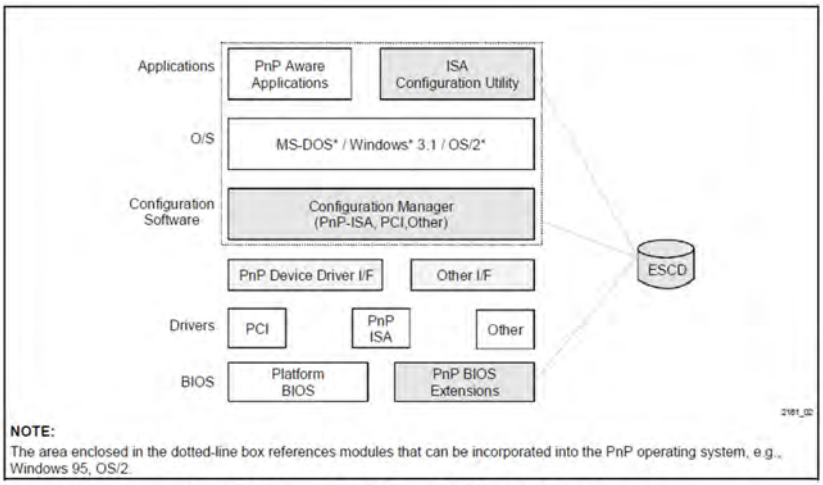


Figure 2. Plug and Play Software Architecture Components.
Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

Anyimi, Fig. 2

Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Although most of this code may be compressed, certain code segments cannot; besides even compression has its limits. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

Anyimi, 2.2

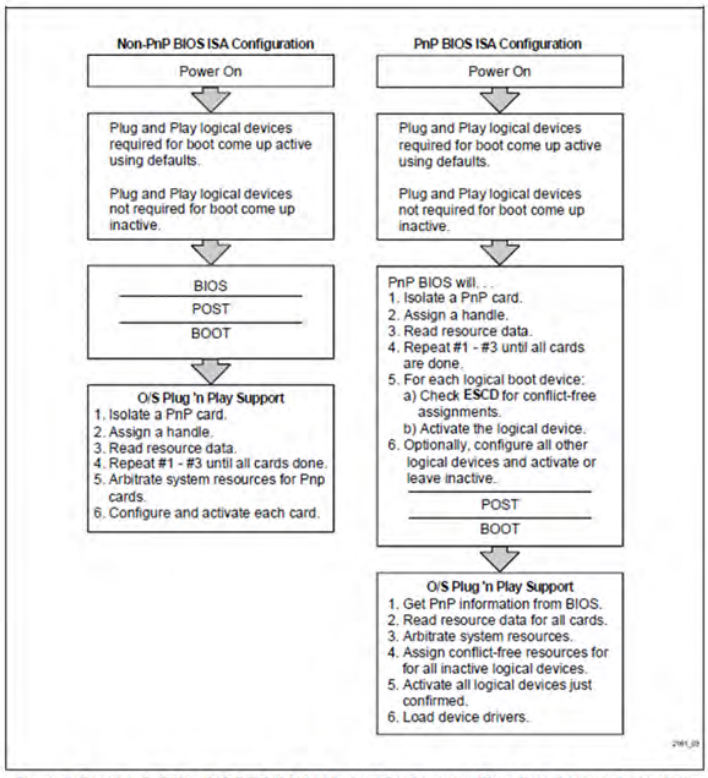


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

Anyimi **Claim 1.1**

"maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

Anyimi, 3.2

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

Anyimi, 3.2

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Anyimi, 5.1

1.2 initializing a central processing unit of said computer system;	Anyimi, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this claim limitation:

3.1 PnP Components

For a system to be fully PnP-compliant, four basic elements are required:

1. System/PnP BIOS
2. PnP operating system
3. PnP hardware devices
4. PnP application software

Anyimi, 3.1

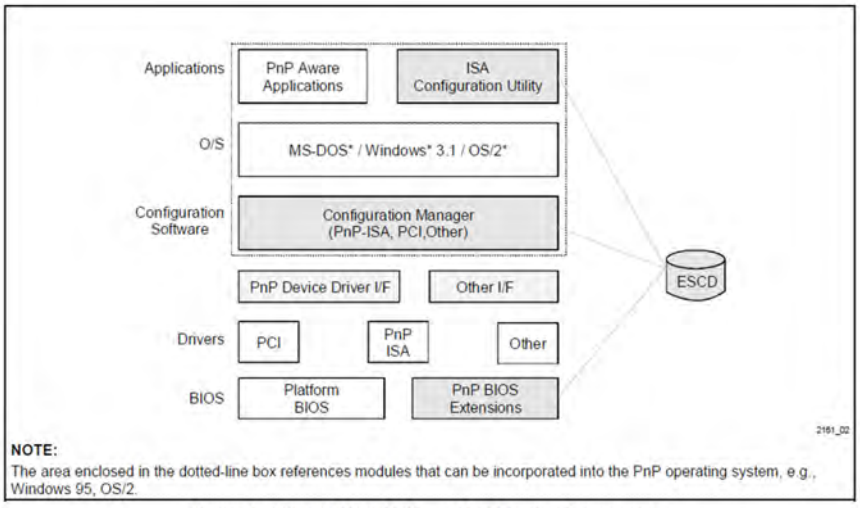


Figure 2. Plug and Play Software Architecture Components.
Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

Anyimi, Fig. 2

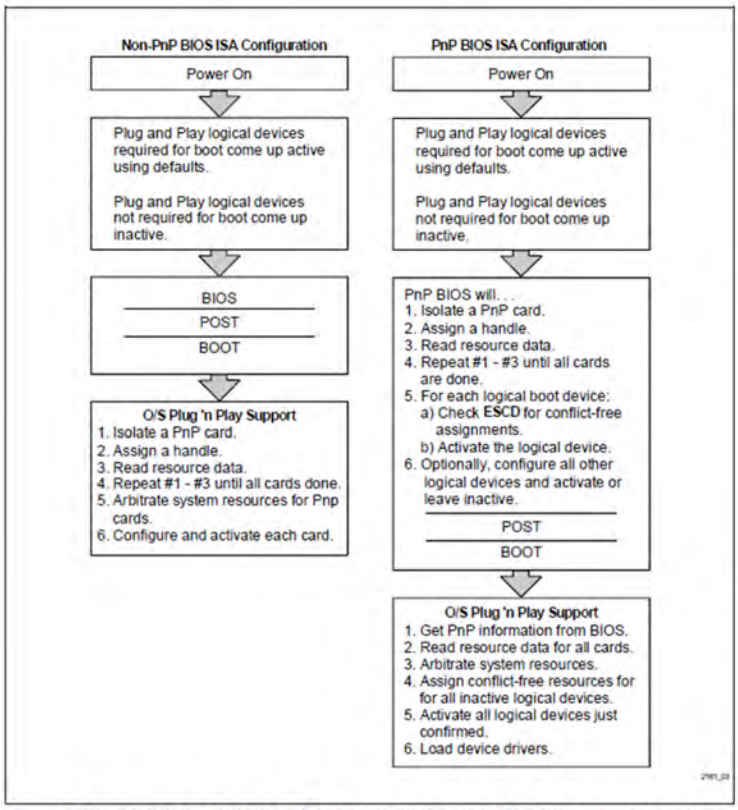


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Anyimi, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions..

Anyimi discloses this claim limitation:

Anyimi discloses this claim limitation:

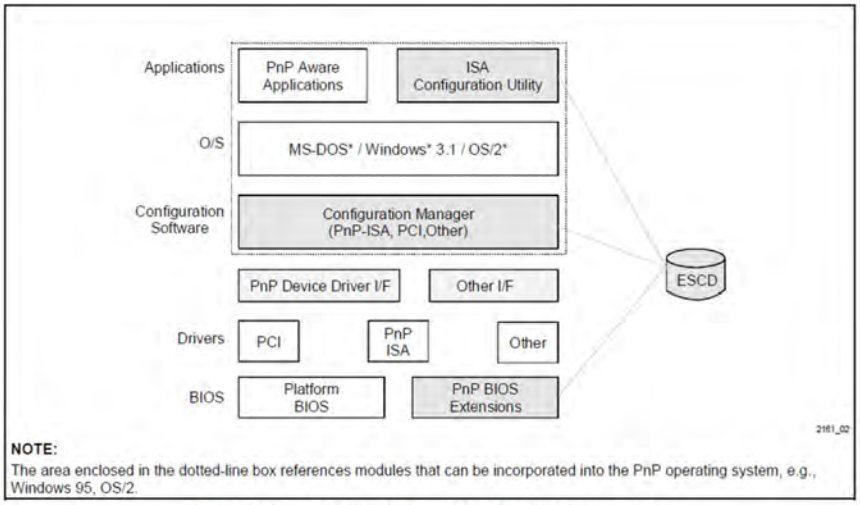


Figure 2. Plug and Play Software Architecture Components.
Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

Anyimi, Fig. 2

Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Although most of this code may be compressed, certain code segments cannot; besides even compression has its limits. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

Anyimi, 2.2

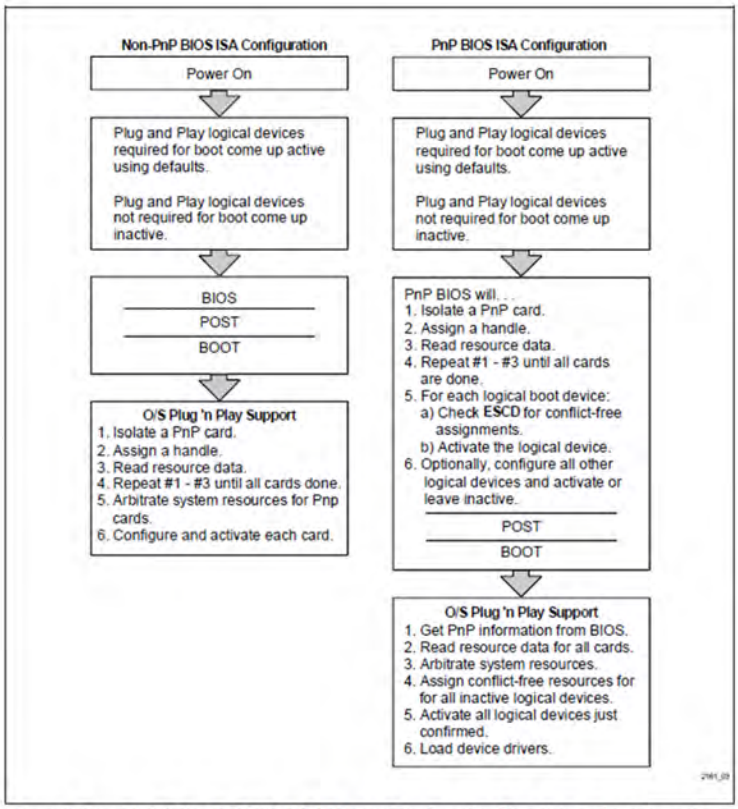


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

Anyimi, 3.2

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

Anyimi, 3.2

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Anyimi, 5.1

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Anyimi, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this claim limitation:

> Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Although most of this code may be compressed, certain code segments cannot; besides even compression has its limits. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

Anyimi, 2.2

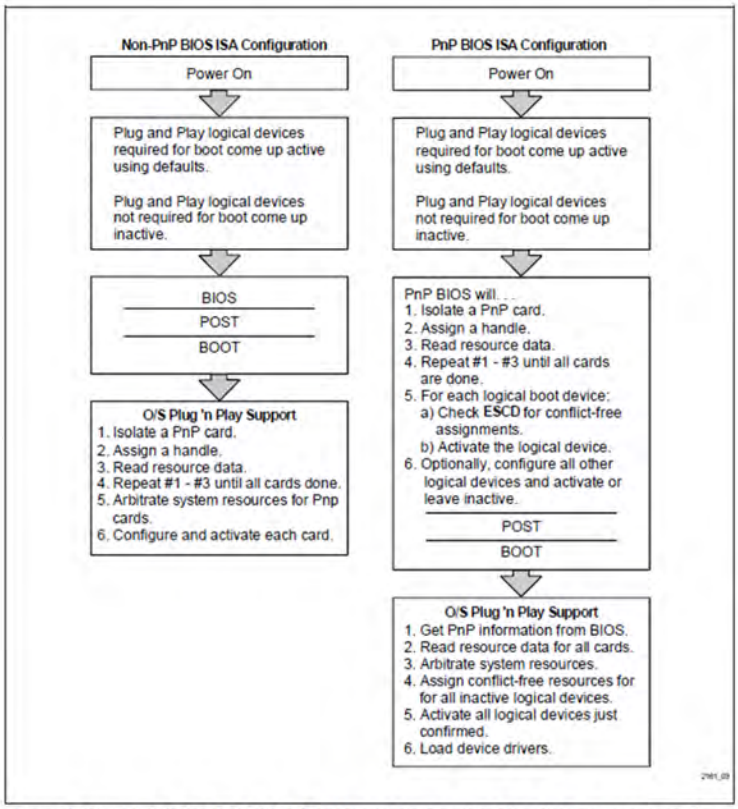


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

Anyimi, 3.2

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

Anyimi, 3.2

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Anyimi, 5.1

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Anyimi, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this claim limitation:

Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Although most of this code may be compressed, certain code segments cannot; besides even compression has its limits. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

Anyimi, 2.2

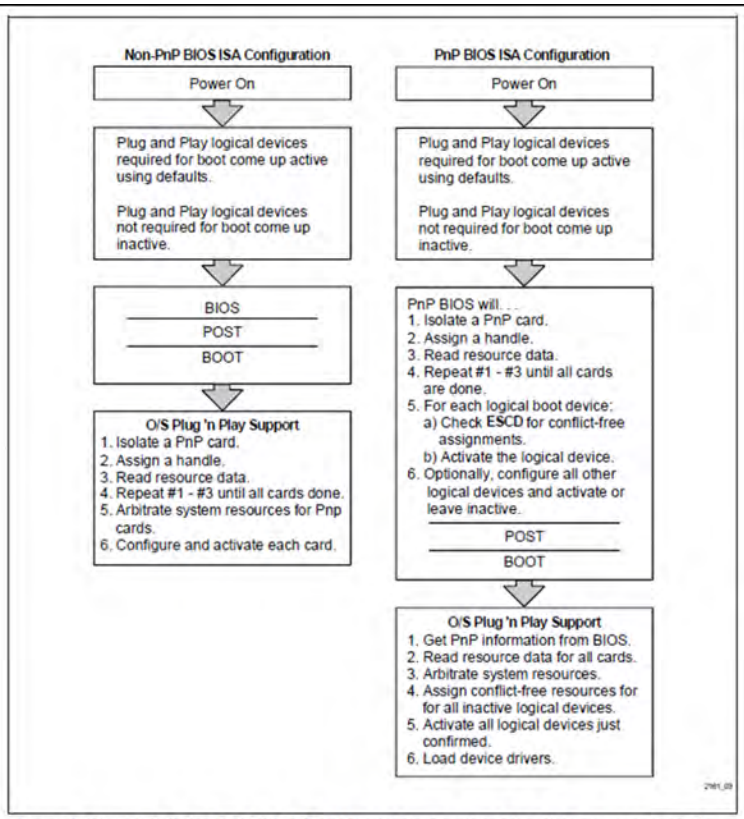


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

Anyimi, 3.2

Anyimi Claim 1.5

"utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

Anyimi, 3.2

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Anyimi, 5.1

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Anyimi, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Anyimi, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

Perhaps you are wondering why flash is the medium of choice advocated in this application note as the means for storing system BIOS, particularly for Plug and Play? From a PC user's perspective, a PnP flash BIOS enables users to install new hardware without having to call the support number. This translates to ease-of-use:

- Easily updatable code assuring optimal BIOS performance
- No need to edit the CONFIG.SYS file.
- No need to determine system type and match, somehow, to jumper settings.
- No need to investigate available system resources and muddle through reassigning them.
- No need to fiddle with system memory reallocation.
- No need to buy a new system every time something changes (increases system's useful lifespan).

Anyimi, 2.0

For a system to be fully PnP-compliant, four basic elements are required:

1. System/PnP BIOS
2. PnP operating system
3. PnP hardware devices
4. PnP application software

Anyimi, 3.1

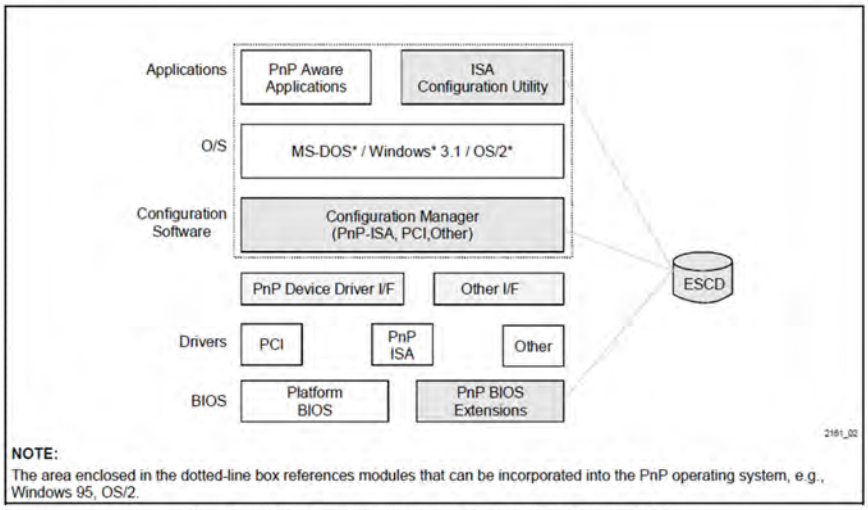


Figure 2. Plug and Play Software Architecture Components. Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

Anyimi, Fig. 2

The Intel boot block (BB) flash memory products are particularly well suited for BIOS applications. Boot block flash is segmented into a lockable boot block, two parameter blocks and one or more main blocks. All boot

block devices are manufactured on Intel's ETOX™ flash memory process technology. An on-chip Write State Machine (WSM) provides automated program and erase algorithms with an SRAM-compatible write interface. The key feature of the boot block architecture that differentiates it from other flash memories is its hardware-lockable boot block, which allows system recovery from fatal crashes. Additional features of boot block flash include:

- Hardware write protection via pin
- Hardware locking of boot block
- Hardware pin for system reset during write
- High performance reads (for speedy access to data)
- Deep power-down mode (a key feature for "green" PCs)
- Extended cycling capability (100,000 block/erase cycles)

Anyimi, 4.0

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Anyimi, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

Perhaps you are wondering why flash is the medium of choice advocated in this application note as the means for storing system BIOS, particularly for Plug and Play? From a PC user's perspective, a PnP flash BIOS enables users to install new hardware without having to call the support number. This translates to ease-of-use:

- Easily updatable code assuring optimal BIOS performance
- No need to edit the CONFIG.SYS file.
- No need to determine system type and match, somehow, to jumper settings.
- No need to investigate available system resources and muddle through reassigning them.
- No need to fiddle with system memory reallocation.
- No need to buy a new system every time something changes (increases system's useful lifespan).

Anyimi, 2.0

For a system to be fully PnP-compliant, four basic elements are required:

1. System/PnP BIOS
2. PnP operating system
3. PnP hardware devices
4. PnP application software

Anyimi, 3.1

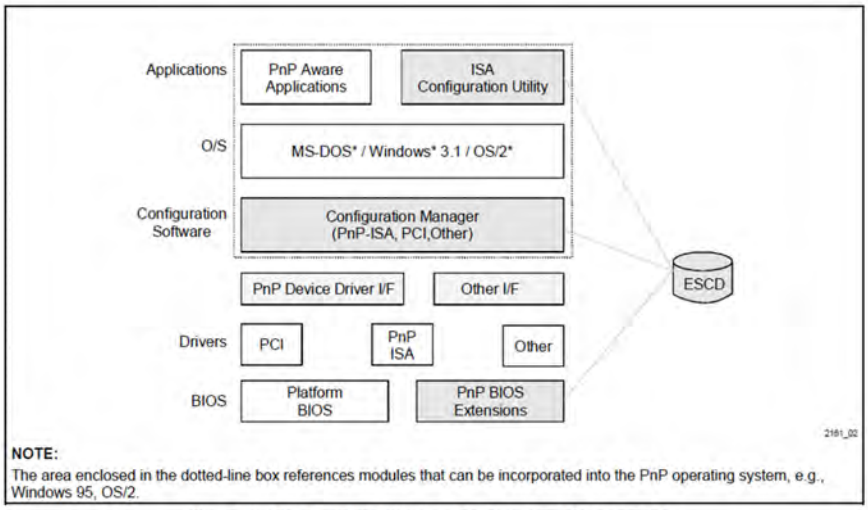


Figure 2. Plug and Play Software Architecture Components. Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

Anyimi, Fig. 2

The Intel boot block (BB) flash memory products are particularly well suited for BIOS applications. Boot block flash is segmented into a lockable boot block, two parameter blocks and one or more main blocks. All boot

block devices are manufactured on Intel's ETOX™ flash memory process technology. An on-chip Write State Machine (WSM) provides automated program and erase algorithms with an SRAM-compatible write interface. The key feature of the boot block architecture that differentiates it from other flash memories is its hardware-lockable boot block, which allows system recovery from fatal crashes. Additional features of boot block flash include:

- Hardware write protection via pin
- Hardware locking of boot block
- Hardware pin for system reset during write
- High performance reads (for speedy access to data)
- Deep power-down mode (a key feature for "green" PCs)
- Extended cycling capability (100,000 block/erase cycles)

Anyimi, 4.0

Anyimi Claim 4

"The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Anyimi, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

The following items are required to have an ISW capable system for updating the PnP BIOS:

- Microprocessor or controller (to control the reprogramming process)
- PnP BIOS boot code, communications software, and PnP BIOS update algorithm
- Data import capability (floppy disk, serial, network, etc.)
- V_{PP} generator or regulator (12V products only)

Anyimi, 5.3.2.1

6. The method of claim 1, further comprising updating the list of boot data.	Anyimi, as evidenced by the example citations below, discloses "updating the list of boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Anyimi discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

Plug and Play can make the usual experience of adding new functionality to an existing system as easy as:

1. Turn the system off.
2. Insert the new device.
3. Turn the system on.

PnP flash BIOS can also extend the life of the PC. By enabling simple updates to the BIOS (simply insert the upgrade disk and the software programs the new BIOS), the user can get more out of their PC investment.

Anyimi, 1.0

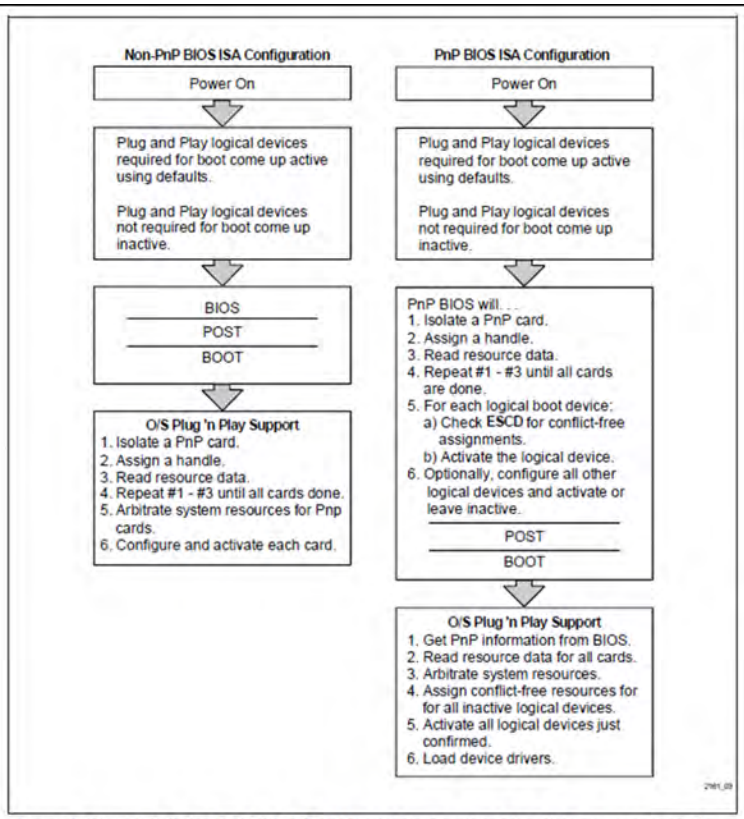


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Anyimi, Fig. 3

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

Anyimi, 3.2

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

Anyimi, 3.2

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Anyimi, 5.1

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Anyimi, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Anyimi, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

11.1. a processor;	Anyimi, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Anyimi, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Anyimi, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Anyimi, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Anyimi, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Anyimi, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Anyimi, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Anyimi, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B25
Invalidity of U.S. Patent 8,090,936 based on Anyimi

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Anyimi, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Anyimi discloses this limitation: *See* Claims 1, 9, and 11 above.	

Appendix B26
Invalidity of U.S. Patent 8,090,936 based on Bennett

The publication D. Bennett, "Booting Linux from EPROM," Linux Journal, January 1997 ("Bennett") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B26
Invalidity of U.S. Patent 8,090,936 based on Bennett

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Bennett, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this claim limitation:

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

Bennett, 1

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

Bennett, 1

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

The first phase of disk image development was identifying the required and the desired items. The first step was to come up with a minimal system and then add the items required for the operator interface. Not being a Unix expert, coming up with the minimal system ended up being something of a trial and error process. I started with what I thought was needed, then tried running it. When an error occurred because of a missing program or library, that file was added. This process went on until the system ran happily.

The bulk of this was done by copying files from the "full" Linux partition to the 6MB partition, booting DOS and using the loadlin line:

```
loadlin zimage root=/dev/hda2 ro
```

Bennett, 3

Once the system was fairly stable, the 6MB partition was loaded into the RAM disk. This is very similar to how the RAM disk is loaded from EPROM, but development went faster since EPROMs weren't being programmed. To test the system without programming EPROMs, the system booted DOS and ran loadlin with the line:

```
loadlin zimage root=/dev/hda2 ramdisk=6144 ro
```

Because of the modification to ramdisk.c, the /dev/hda2 disk image is loaded into the RAM disk, then the root file system is switched to the RAM disk. The process of refining the disk image continues until everything is "perfect".

Bennett, 4

Bennett **Claim 1.1**

"maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

1.2 initializing a central processing unit of said computer system;	Bennett, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions

Bennett discloses this claim limitation:

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

Bennett, 1

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Bennett, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this claim limitation:

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

Bennett, 1

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

Bennett, 1

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

The first phase of disk image development was identifying the required and the desired items. The first step was to come up with a minimal system and then add the items required for the operator interface. Not being a Unix expert, coming up with the minimal system ended up being something of a trial and error process. I started with what I thought was needed, then tried running it. When an error occurred because of a missing program or library, that file was added. This process went on until the system ran happily.

The bulk of this was done by copying files from the "full" Linux partition to the 6MB partition, booting DOS and using the loadlin line:

```
loadlin zimage root=/dev/hda2 ro
```

Bennett, 3

Once the system was fairly stable, the 6MB partition was loaded into the RAM disk. This is very similar to how the RAM disk is loaded from EPROM, but development went faster since EPROMs weren't being programmed. To test the system without programming EPROMs, the system booted DOS and ran loadlin with the line:

```
loadlin zimage root=/dev/hda2 ramdisk=6144 ro
```

Because of the modification to ramdisk.c, the /dev/hda2 disk image is loaded into the RAM disk, then the root file system is switched to the RAM disk. The process of refining the disk image continues until everything is "perfect".

Bennett, 4

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Bennett, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this claim limitation:

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

Bennett, 1

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

Bennett, 1

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

The first phase of disk image development was identifying the required and the desired items. The first step was to come up with a minimal system and then add the items required for the operator interface. Not being a Unix expert, coming up with the minimal system ended up being something of a trial and error process. I started with what I thought was needed, then tried running it. When an error occurred because of a missing program or library, that file was added. This process went on until the system ran happily.

The bulk of this was done by copying files from the "full" Linux partition to the 6MB partition, booting DOS and using the loadlin line:

```
loadlin zimage root=/dev/hda2 ro
```

Bennett, 3

Once the system was fairly stable, the 6MB partition was loaded into the RAM disk. This is very similar to how the RAM disk is loaded from EPROM, but development went faster since EPROMs weren't being programmed. To test the system without programming EPROMs, the system booted DOS and ran loadlin with the line:

```
loadlin zimage root=/dev/hda2 ramdisk=6144 ro
```

Because of the modification to ramdisk.c, the /dev/hda2 disk image is loaded into the RAM disk, then the root file system is switched to the RAM disk. The process of refining the disk image continues until everything is "perfect".

Bennett, 4

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Bennett, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this claim limitation:

System Operation

For booting, two options were considered:

- booting DOS, then running the loadlin program (to load Linux) from autoexec.bat
- installing LILO and booting Linux directly

The advantage of the second option would be a slightly shorter boot time. However, we implemented the first option, because we wanted to use a programmable keyboard—the software for programming the keyboard runs under DOS.

Bennett, 1

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

Bennett, 1

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

The first phase of disk image development was identifying the required and the desired items. The first step was to come up with a minimal system and then add the items required for the operator interface. Not being a Unix expert, coming up with the minimal system ended up being something of a trial and error process. I started with what I thought was needed, then tried running it. When an error occurred because of a missing program or library, that file was added. This process went on until the system ran happily.

The bulk of this was done by copying files from the "full" Linux partition to the 6MB partition, booting DOS and using the loadlin line:

```
loadlin zimage root=/dev/hda2 ro
```

Bennett, 3

Once the system was fairly stable, the 6MB partition was loaded into the RAM disk. This is very similar to how the RAM disk is loaded from EPROM, but development went faster since EPROMs weren't being programmed. To test the system without programming EPROMs, the system booted DOS and ran loadlin with the line:

```
loadlin zimage root=/dev/hda2 ramdisk=6144 ro
```

Because of the modification to ramdisk.c, the /dev/hda2 disk image is loaded into the RAM disk, then the root file system is switched to the RAM disk. The process of refining the disk image continues until everything is "perfect".

Bennett, 4

Bennett Claim 1.5

"utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

2. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.	Bennett, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1, 1.4, and 1.5 above.	

3. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.	Bennett, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

This is a quick look at making Linux bootable from EPROM on a 486 single board computer.

This article describes one way to run Linux in an embedded system with no hard disk. The application described is an Operator Interface in a monitor and display system developed by Boeing Flight Test. The airborne environment requires something fairly rugged which can withstand common power interruptions. To meet these requirements we decided to build the operator interface without a hard disk.

Bennett, 1

In order to automatically run the operator interface application, a program was written to replace getty. This program (dboot.c) will run login for a given user, and set the stdin, stdout and stderr to the specified virtual console.

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

4. The method of claim 1, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.	Bennett, as evidenced by the example citations below, discloses "said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

This is a quick look at making Linux bootable from EPROM on a 486 single board computer.

This article describes one way to run Linux in an embedded system with no hard disk. The application described is an Operator Interface in a monitor and display system developed by Boeing Flight Test. The airborne environment requires something fairly rugged which can withstand common power interruptions. To meet these requirements we decided to build the operator interface without a hard disk.

Bennett, 1

In order to automatically run the operator interface application, a program was written to replace getty. This program (dboot.c) will run login for a given user, and set the stdin, stdout and stderr to the specified virtual console.

The boot sequence is:

- Power up and run memory tests
- load DOS which executes AUTOEXEC.BAT
  - run the keyboard programming application
  - run loadlin—this reads a linux kernel from the DOS disk & executes it
- the linux kernel takes over
- load the RAM disk from the EPROM disk
- switch the root file system to the RAM disk
- init will read inittab which executes dboot instead of getty
- the operator interface application is started

Bennett, 2

5. The method of claim 1, wherein said preloading is performed by a data storage controller connected to said boot device.	Bennett, as evidenced by the example citations below, discloses "said preloading is performed by a data storage controller connected to said boot device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said preloading is performed by a data storage controller connected to said boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1, 1.3, and 1.4 above.	

6. The method of claim 1, further comprising updating the list of boot data.	Bennett, as evidenced by the example citations below, discloses "updating the list of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the list of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

8. The method of claim 1, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.	Bennett, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Bennett discloses this limitation:

See Claims 1.1, 1.3, and 1.4 above.

See also

When deciding how to implement the EPROM device driver, the first idea was to create an image of a disk in the EPROM. This would provide a RAM disk of the same size as the EPROM, 3.5MB in this case (the DOS portion of the SSD takes 1/2 MB). Instead, to allow a larger RAM disk, a compressed disk image is used. The compression used is simple—any sectors which are identical are only stored once. The primary advantage this gives is blank areas of the disk image don't need to take up EPROM space. Listing 1 shows the SSD disk compression used.

EPROM Disk Compression
(/files/linuxjournal.com/linuxjournal/articles/002/0243/0243f1.jpg)

Bennett, 1

9. The method of claim 1, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.	Bennett, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

11.1. a processor;	Bennett, as evidenced by the example citations below, discloses "a processor."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claim 1.2 above.	

11.2. a memory; and	Bennett, as evidenced by the example citations below, discloses "a memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.3, and 1.4 above.	

11.3.1 a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device,	Bennett, as evidenced by the example citations below, discloses "a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1, 1.3, 1.4 above.	

11.3.2 said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and,	Bennett, as evidenced by the example citations below, discloses "said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.3, 1.4, and 1.5 above.	

11.4 a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.	Bennett, as evidenced by the example citations below, discloses "a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1 and 1.5 above.	

12. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.	Bennett, as evidenced by the example citations below, discloses "said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1, 3, and 11.3.1 above.	

13. The system of claim 11, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.	Bennett, as evidenced by the example citations below, discloses "wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1.1, 1.3, 3, 5 and 11.3.1 and 11.3.2 above.	

15. The system of claim 11, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.	Bennett, as evidenced by the example citations below, discloses "Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Bennett discloses this limitation: *See* Claims 1, 8, and 11 above.	

Appendix B26
Invalidity of U.S. Patent 8,090,936 based on Bennett

16. The system of claim 11, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.	Bennett, as evidenced by the example citations below, discloses "a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

Appendix B27
Invalidity of U.S. Patent 8,090,936 based on Burrows

The publication Michael Burrows, Charles Jerian, Butler Lampson and Timothy Mann, On-line data compression in a log-structured file system. ("Burrows") invalidates claims 1-6, 8-9, 11-13, and 15-16 of United States Patent No. 8,090,936 ("the '936 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '936 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix B27
Invalidity of U.S. Patent 8,090,936 based on Burrows

1.1 maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;	Burrows, as evidenced by the exemplary citations below, discloses "maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Burrows discloses this limitation:

> "We have incorporated on-line data compression into the low levels of a log-structured file system (Rosenblum's Sprite LFS). Each block of data or meta-data is compressed as it is written to the disk and decompressed as it is read."

Burrows, Abstract.

> "The module that reads a disk block given its logical address needs a way to find the physical address of the compressed bytes. We keep a logical block map for each segment, which is simply an array indexed by compression block number, whose entries are the physical byte addresses of the blocks relative to the start of the segment. The block map is constructed in memory as the segment is being compressed, and written to the end of the segment when the segment is full. The maps are needed for all file reads, so they are cached in memory whenever possible."

Burrows, 5.

1.2 initializing a central processing unit of said computer system;	Burrows, as evidenced by the example citations below, discloses "initializing a central processing unit of said computer system;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, initializing a central processing unit of said computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.3 preloading said at least a portion of said boot data in compressed form from said boot device to a memory;	Burrows, as evidenced by the example citations below, discloses "preloading said at least a portion of said boot data in compressed form from said boot device to a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, preloading said at least a portion of said boot data in compressed form from said boot device to a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Burrows discloses this limitation:

> "We have incorporated on-line data compression into the low levels of a log-structured file system (Rosenblum's Sprite LFS). Each block of data or meta-data is compressed as it is written to the disk and decompressed as it is read."

Burrows, Abstract.

> "The module that reads a disk block given its logical address needs a way to find the physical address of the compressed bytes. We keep a logical block map for each segment, which is simply an array indexed by compression block number, whose entries are the physical byte addresses of the blocks relative to the start of the segment. The block map is constructed in memory as the segment is being compressed, and written to the end of the segment when the segment is full. The maps are needed for all file reads, so they are cached in memory whenever possible."

Burrows, 5.

> "Unfortunately, this procedure reads and decompresses a full compression block even if the caller wanted only some smaller unit, such as a file system block or a physical sector. We alleviate this problem by caching the entire decompressed block in memory, rather than just caching the requested sectors. The data could be placed in the file system buffer cache, but for simplicity in our prototype, we cached the last decompressed block within the read routine. Sprite LFS reads files sequentially in 4 KByte units, so this simple caching strategy typically achieves three hits for each 16 KByte compression block when reading large files.

> When the file system is reading non-sequentially, the additional time to read a full compression block cannot be hidden by caching. Fortunately, this time is small compared to the rotational latency. The time needed to decompress the full block in software is several milliseconds; it would be much smaller if decompression were implemented in hardware."
>
> Burrows, 6.

1.4 accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and	Burrows, as evidenced by the example citations below, discloses "accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing and decompressing said at least a portion of said boot data in said compressed form from said memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Burrows discloses this limitation:

> "We have incorporated on-line data compression into the low levels of a log-structured file system (Rosenblum's Sprite LFS). Each block of data or meta-data is compressed as it is written to the disk and decompressed as it is read."

Burrows, Abstract.

> "The module that reads a disk block given its logical address needs a way to find the physical address of the compressed bytes. We keep a logical block map for each segment, which is simply an array indexed by compression block number, whose entries are the physical byte addresses of the blocks relative to the start of the segment. The block map is constructed in memory as the segment is being compressed, and written to the end of the segment when the segment is full. The maps are needed for all file reads, so they are cached in memory whenever possible."

Burrows, 5.

> "The procedure for finding the compressed data associated with a logical address is as follows:
>
> > 1. Extract the segment number from the logical address. Use it to find the logical block map for the segment.
> >
> > 2. Extract the compression block number from the address. Use it to index the logical block map. This yields the physical byte offset of the compressed data within the segment.
> >
> > 3. Examine the next entry in the map, to find the start of the next block. This determines how much data should be read from the

disk.

4. Read the compressed data from the disk and decompress it.

5. Extract the sector number from the logical address. Use it to identify the desired sector within the decompressed block."

Burrows, 5-6.

> "Unfortunately, this procedure reads and decompresses a full compression block even if the caller wanted only some smaller unit, such as a file system block or a physical sector. We alleviate this problem by caching the entire decompressed block in memory, rather than just caching the requested sectors. The data could be placed in the file system buffer cache, but for simplicity in our prototype, we cached the last decompressed block within the read routine. Sprite LFS reads files sequentially in 4 KByte units, so this simple caching strategy typically achieves three hits for each 16 KByte compression block when reading large files.
>
> When the file system is reading non-sequentially, the additional time to read a full compression block cannot be hidden by caching. Fortunately, this time is small compared to the rotational latency. The time needed to decompress the full block in software is several milliseconds; it would be much smaller if decompression were implemented in hardware."

Burrows, 6.

1.5 utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.	Burrows, as evidenced by the example citations below, discloses "utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Burrows discloses this limitation:

> "We have incorporated on-line data compression into the low levels of a log-structured file system (Rosenblum's Sprite LFS). Each block of data or meta-data is compressed as it is written to the disk and decompressed as it is read."

Burrows, Abstract.

> "The procedure for finding the compressed data associated with a logical address is as follows:
>
> 1. Extract the segment number from the logical address. Use it to find the logical block map for the segment.
>
> 2. Extract the compression block number from the address. Use it to index the logical block map. This yields the physical byte offset of the compressed data within the segment.
>
> 3. Examine the next entry in the map, to find the start of the next block. This determines how much data should be read from the disk.
>
> 4. Read the compressed data from the disk and decompress it.
>
> 5. Extract the sector number from the logical address. Use it to identify the desired sector within the decompressed block."