

IW 7656177



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 16, 2018

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE
RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS
OF:

APPLICATION NUMBER: 09/608,266

FILING DATE: June 30, 2000

PATENT NUMBER: 6,771,646

ISSUE DATE: August 03, 2004

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office



P. SWAIN
Certifying Officer

PART (1) OF 2 PART(S)



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D. C. 20231
 www.uspto.gov



Bib Data Sheet

SERIAL NUMBER 09/608,266	FILING DATE 06/30/2000 RULE -	CLASS 370	GROUP ART UNIT 2731	ATTORNEY DOCKET NO. APPT-001-4
------------------------------------	---	---------------------	-------------------------------	--

APPLICANTS
 Haig A. Sarkissian, San Antonio, TX ;
 Russell S. Dietz, San Jose, CA ;

**** CONTINUING DATA ******* *Yes*
 THIS APPLN CLAIMS BENEFIT OF 60/141,903 06/30/1999

**** FOREIGN APPLICATIONS ******* *None*

IF REQUIRED, FOREIGN FILING LICENSE GRANTED ** 09/01/2000

Foreign Priority claimed <input type="checkbox"/> yes <input checked="" type="checkbox"/> no	STATE OR COUNTRY TX	SHEETS DRAWING 21	TOTAL CLAIMS 20	INDEPENDENT CLAIMS 3
35 USC 119 (a-d) conditions met <input type="checkbox"/> yes <input checked="" type="checkbox"/> no <input type="checkbox"/> Met after Allowance				
Verified and Acknowledged	Examiner's Signature	Initials		

ADDRESS
 Dov Rosenfeld
 5507 College Avenue
 Suite 2
 Oakland, CA 94618

TITLE
 Associative cache structure for lookups and updates of flow records in a network monitor

FILING FEE RECEIVED 840	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	<input type="checkbox"/> All Fees
		<input type="checkbox"/> 1.16 Fees (Filing)
		<input type="checkbox"/> 1.17 Fees (Processing Ext. of time)
		<input type="checkbox"/> 1.18 Fees (Issue)
		<input type="checkbox"/> Other
		<input type="checkbox"/> Credit

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

PTO-1556
(5/87)

*U.S. GPO: 1999-459-082/19144

07-03-00

A



IN THE U.S. PATENT AND TRADEMARK OFFICE
Application Transmittal Sheet

Our Ref./Docket No.: APPT-001-4

Box Patent Application
ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231



Dear Assistant Commissioner:

Transmitted herewith is the patent application of

INVENTOR(s)/APPLICANT(s)		
Last Name	First Name, MI	Residence (City and State or Country)
Sarkissian	Haig A.	San Antonio, Texas
Dietz	Russell S.	San Jose, CA

TITLE OF THE INVENTION

ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

CORRESPONDENCE ADDRESS AND AGENT FOR APPLICANT(S)

Dov Rosenfeld, Reg. No. 38,387
5507 College Avenue, Suite 2
Oakland, California, 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

ENCLOSED APPLICATION PARTS (check all that apply)

Included are:

- 65 sheet(s) of specification, claims, and abstract
- 21 sheet(s) of formal Drawing(s) with a submission letter to the Official Draftsperson
- Information Disclosure Statement.
- Form PTO-1449: INFORMATION DISCLOSURE CITATION IN AN APPLICATION, together with a copy of each references included in PTO-1449.
- Declaration and Power of Attorney
- An assignment of the invention to Apptitude, Inc.
- A letter requesting recordation of the assignment.
- An assignment Cover Sheet.
- Additional inventors are being named on separately numbered sheets attached hereto.
- Return postcard.

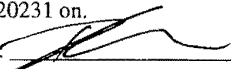
This application has:

- a small entity status. A verified statement:
 - is enclosed
 - was already filed.

The fee has been calculated as shown in the following page.

Certificate of Mailing under 37 CFR 1.10

I hereby certify that this application and all attachments are being deposited with the United States Postal Service as Express Mail (Express Mail Label: EI417961895US) in an envelope addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: June 30, 2000 Signed: 

Name: Dov Rosenfeld, Reg. No. 38687

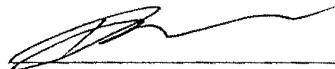
	TOTAL CLAIMS	NO. OF EXTRA CLAIMS	RATE	EXTRA CLAIM FEE
TOTAL CLAIMS	20	0	\$18	\$ 0.00
INDEP. CLAIMS	3	0	\$78	\$ 0.00
BASIC APPLICATION FEE:				\$ 690.00
TOTAL FEES PAYABLE:				\$ 690.00

METHOD OF PAYMENT

- _____ A check in the amount of _____ is attached for application fee and presentation of claims.
_____ A check in the amount of \$ 40.00 is attached for recordation of the Assignment.
_____ The Commissioner is hereby authorized to charge payment of the any missing filing or other fees required for this filing or credit any overpayment to Deposit Account No. 50-0292
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

June 30, 2000
Date


Dov Rosenfeld , Reg. No. 38687

Correspondence Address:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, California, 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, <i>et al.</i> Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Group Art Unit: unassigned Examiner: unassigned
--	--

**LETTER TO OFFICIAL DRAFTSPERSON
SUBMISSION OF FORMAL DRAWINGS**


The Assistant Commissioner for Patents
Washington, DC 20231
ATTN: Official Draftsperson

Dear Sir or Madam:

Attached please find 21 sheets of formal drawings to be made of record for the above identified patent application submitted herewith.

Respectfully Submitted,

June 30, 2000
Date



Dov Rosenfeld, Reg. No. 38687

Address for correspondence and attorney for applicant(s):
Dov Rosenfeld, Reg. No. 38,687
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

Certificate of Mailing under 37 CFR 1.10

I hereby certify that this application and all attachments are being deposited with the United States Postal Service as Express Mail (Express Mail Label: EI417961895US in an envelope addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231 on

Date: June 30, 2000

Signed: 
Name: Dov Rosenfeld, Reg. No. 38687

Our Ref./Docket No.: APPT-001-4

ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW
RECORDS IN A NETWORK MONITOR

Inventor(s):

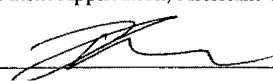
SARKISSIAN, Haig A.
San Antonio, Texas

DIETZ, Russell S.
San Jose, CA

Certificate of Mailing under 37 CFR 1.10

I hereby certify that this application and all attachments are being deposited with the United States Postal Service as Express Mail (Express Mail Label: E1417961895US in an envelope addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: June 30, 2000

Signed: 

ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of U.S. Provisional Patent Application Serial No.:
5 60/141,903 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK to inventors Dietz, et al., filed June 30, 1999, the contents of which are incorporated herein by reference.

Ani
4/14/09

U.S. patents cited

This application is related to the following U.S. patent applications, each filed concurrently with the present application, and each assigned to Appitude, Inc., the assignee of the present invention:

10

C
C

No. 6,651,594
U.S. Patent Application Serial No. ~~_____~~ for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al., ~~filed June 30, 2000, Attorney/Agent Reference Number APPT-001-1,~~ and incorporated herein by reference.

C
C

15

No. 6,665,725
U.S. Patent Application Serial No. ~~_____~~ for PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE, to inventors Koppenhaver, et al., ~~filed June 30, 2000, Attorney/Agent Reference Number APPT-001-2,~~ and incorporated herein by reference.

C
C

20

09/608,126
U.S. Patent Application Serial No. ~~_____~~ for RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING, to inventors Dietz, et al., ~~filed June 30, 2000, Attorney/Agent Reference Number APPT-001-3,~~ and incorporated herein by reference.

C
C

25

09/608,267
U.S. Patent Application Serial No. ~~_____~~ for STATE PROCESSOR FOR PATTERN MATCHING IN A NETWORK MONITOR DEVICE, to inventors Sarkissian, et al., ~~filed June 30, 2000, Attorney/Agent Reference Number APPT-001-5,~~ and incorporated herein by reference.

FIELD OF INVENTION

The present invention relates to computer networks, specifically to the real-time

elucidation of packets communicated within a data network, including classification according to protocol and application program.

BACKGROUND

There has long been a need for network activity monitors. This need has become especially acute, however, given the recent popularity of the Internet and other interconnected networks. In particular, there is a need for a real-time network monitor that can provide details as to the application programs being used. Such a monitor should enable non-intrusive, remote detection, characterization, analysis, and capture of all information passing through any point on the network (*i.e.*, of all packets and packet streams passing through any location in the network). Not only should all the packets be detected and analyzed, but for each of these packets the network monitor should determine the protocol (*e.g.*, http, ftp, H.323, VPN, etc.), the application/use within the protocol (*e.g.*, voice, video, data, real-time data, etc.), and an end user's pattern of use within each application or the application context (*e.g.*, options selected, service delivered, duration, time of day, data requested, etc.). Also, the network monitor should not be reliant upon server resident information such as log files. Rather, it should allow a user such as a network administrator or an Internet service provider (ISP) the means to measure and analyze network activity objectively; to customize the type of data that is collected and analyzed; to undertake real time analysis; and to receive timely notification of network problems.

Related and incorporated by reference U.S. Patent ^{No. 6,651,099} ~~application~~ ~~number~~ for *METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK*, to inventors Dietz, et al, Attorney/Agent Docket APPT-001-1, describes a network monitor that includes carrying out protocol specific operations on individual packets including extracting information from header fields in the packet to use for building a signature for identifying the conversational flow of the packet and for recognizing future packets as belonging to a previously encountered flow. A parser subsystem includes a parser for recognizing different patterns in the packet that identify the protocols used. For each protocol recognized, a slicer extracts important packet elements from the packet. These form a signature (*i.e.*, key) for the packet. The slicer also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

elucidation of packets communicated within a data network, including classification according to protocol and application program.

BACKGROUND

There has long been a need for network activity monitors. This need has become especially acute, however, given the recent popularity of the Internet and other interconnected networks. In particular, there is a need for a real-time network monitor that can provide details as to the application programs being used. Such a monitor should enable non-intrusive, remote detection, characterization, analysis, and capture of all information passing through any point on the network (*i.e.*, of all packets and packet streams passing through any location in the network). Not only should all the packets be detected and analyzed, but for each of these packets the network monitor should determine the protocol (*e.g.*, http, ftp, H.323, VPN, etc.), the application/use within the protocol (*e.g.*, voice, video, data, real-time data, etc.), and an end user's pattern of use within each application or the application context (*e.g.*, options selected, service delivered, duration, time of day, data requested, etc.). Also, the network monitor should not be reliant upon server resident information such as log files. Rather, it should allow a user such as a network administrator or an Internet service provider (ISP) the means to measure and analyze network activity objectively; to customize the type of data that is collected and analyzed; to undertake real time analysis; and to receive timely notification of network problems.

Related and incorporated by reference U.S. Patent ^{No. 6,651,099} ~~application~~ _____ for *METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK*, to inventors Dietz, et al, Attorney/Agent Docket ~~APPT-001-1~~, describes a network monitor that includes carrying out protocol specific operations on individual packets including extracting information from header fields in the packet to use for building a signature for identifying the conversational flow of the packet and for recognizing future packets as belonging to a previously encountered flow. A parser subsystem includes a parser for recognizing different patterns in the packet that identify the protocols used. For each protocol recognized, a slicer extracts important packet elements from the packet. These form a signature (*i.e.*, key) for the packet. The slicer also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

elucidation of packets communicated within a data network, including classification according to protocol and application program.

BACKGROUND

There has long been a need for network activity monitors. This need has become especially acute, however, given the recent popularity of the Internet and other interconnected networks. In particular, there is a need for a real-time network monitor that can provide details as to the application programs being used. Such a monitor should enable non-intrusive, remote detection, characterization, analysis, and capture of all information passing through any point on the network (*i.e.*, of all packets and packet streams passing through any location in the network). Not only should all the packets be detected and analyzed, but for each of these packets the network monitor should determine the protocol (*e.g.*, http, ftp, H.323, VPN, etc.), the application/use within the protocol (*e.g.*, voice, video, data, real-time data, etc.), and an end user's pattern of use within each application or the application context (*e.g.*, options selected, service delivered, duration, time of day, data requested, etc.). Also, the network monitor should not be reliant upon server resident information such as log files. Rather, it should allow a user such as a network administrator or an Internet service provider (ISP) the means to measure and analyze network activity objectively; to customize the type of data that is collected and analyzed; to undertake real time analysis; and to receive timely notification of network problems.

Related and incorporated by reference U.S. Patent ^{No. 6,651,699} ~~application~~ ~~_____~~ for *METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK*, to inventors Dietz, et al, ~~Attorney/Agent Docket APPT-001-1~~, describes a network monitor that includes carrying out protocol specific operations on individual packets including extracting information from header fields in the packet to use for building a signature for identifying the conversational flow of the packet and for recognizing future packets as belonging to a previously encountered flow. A parser subsystem includes a parser for recognizing different patterns in the packet that identify the protocols used. For each protocol recognized, a slicer extracts important packet elements from the packet. These form a signature (*i.e.*, key) for the packet. The slicer also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

likely that a packet associated with the least recently used flow-entry will soon arrive.

A hash is often used to facilitate lookups. Such a hash may spread entries randomly in a database. In such a case, a associative cache is desirable.

There thus is a need for a associative cache subsystem that also includes a LRU
5 replacement policy.

SUMMARY

Described herein is an associative cache system for looking up one or more elements of an external memory. The cache system comprises a set of cache memory elements coupled to the external memory, a set of content addressable memory cells
10 (CAMs) containing an address and a pointer to one of the cache memory elements, and including a matching circuit having an input such that the CAM asserts a match output when the input is the same as the address in the CAM cell, ^{The} which cache memory
C element ^{which} a particular CAM points to changes over time. In the preferred implementation, the CAMs are connected in an order from top to bottom, and the bottom CAM points to
15 the least recently used cache memory element.

BRIEF DESCRIPTION OF THE DRAWINGS

Although the present invention is better understood by referring to the detailed preferred embodiments, these should not be taken to limit the present invention to any specific embodiment because such embodiments are provided only for the purposes of
20 explanation. The embodiments, in turn, are explained with the aid of the following figures.

FIG. 1 is a functional block diagram of a network embodiment of the present invention in which a monitor is connected to analyze packets passing at a connection point.

25 FIG. 2 is a diagram representing an example of some of the packets and their formats that might be exchanged in starting, as an illustrative example, a conversational flow between a client and server on a network being monitored and analyzed. A pair of flow signatures particular to this example and to embodiments of the present invention is also illustrated. This represents some of the possible flow signatures that can be

generated and used in the process of analyzing packets and of recognizing the particular server applications that produce the discrete application packet exchanges.

FIG. 3 is a functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. 1. This process may be implemented in software or hardware.

FIG. 4 is a flowchart of a high-level protocol language compiling and optimization process, which in one embodiment may be used to generate data for monitoring packets according to versions of the present invention.

FIG. 5 is a flowchart of a packet parsing process used as part of the parser in an embodiment of the inventive packet monitor.

FIG. 6 is a flowchart of a packet element extraction process that is used as part of the parser in an embodiment of the inventive packet monitor.

FIG. 7 is a flowchart of a flow-signature building process that is used as part of the parser in the inventive packet monitor.

FIG. 8 is a flowchart of a monitor lookup and update process that is used as part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 9 is a flowchart of an exemplary Sun Microsystems Remote Procedure Call application than may be recognized by the inventive packet monitor.

FIG. 10 is a functional block diagram of a hardware parser subsystem including the pattern recognizer and extractor that can form part of the parser module in an embodiment of the inventive packet monitor.

FIG. 11 is a functional block diagram of a hardware analyzer including a state processor that can form part of an embodiment of the inventive packet monitor.

FIG. 12 is a functional block diagram of a flow insertion and deletion engine process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 13 is a flowchart of a state processing process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 14 is a simple functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. 1. This process may be implemented in software.

FIG. 15 is a functional block diagram of how the packet monitor of FIG. 3 (and
5 FIGS. 10 and 11) may operate on a network with a processor such as a microprocessor.

FIG. 16 is an example of the top (MAC) layer of an Ethernet packet and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. 17A is an example of the header of an Ethertype type of Ethernet packet of
10 FIG. 16 and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. 17B is an example of an IP packet, for example, of the Ethertype packet shown in FIGS. 16 and 17A, and some of the elements that may be extracted to form a signature according to one aspect of the invention.

15 FIG. 18A is a three dimensional structure that can be used to store elements of the pattern, parse and extraction database used by the parser subsystem in accordance to one embodiment of the invention.

FIG. 18B is an alternate form of storing elements of the pattern, parse and extraction database used by the parser subsystem in accordance to another embodiment
20 of the invention.

FIG. 19 is a block diagram of the cache memory part of the cache subsystem
1115 of the analyzer subsystem of FIG. 11.

FIG. 20 is a block diagram of the cache memory controller and the cache CAM controller of the cache subsystem.

25 FIG. 21 is a block diagram of one implementation of the CAM array of the cache subsystem 1115.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Note that this document includes hardware diagrams and descriptions that may include signal names. In most cases, the names are sufficiently descriptive, in other cases however the signal names are not needed to understand the operation and practice of the invention.

Operation in a Network

FIG. 1 represents a system embodiment of the present invention that is referred to herein by the general reference numeral 100. The system 100 has a computer network 102 that communicates packets (*e.g.*, IP datagrams) between various computers, for example between the clients 104–107 and servers 110 and 112. The network is shown schematically as a cloud with several network nodes and links shown in the interior of the cloud. A monitor 108 examines the packets passing in either direction past its connection point 121 and, according to one aspect of the invention, can elucidate what application programs are associated with each packet. The monitor 108 is shown examining packets (*i.e.*, datagrams) between the network interface 116 of the server 110 and the network. The monitor can also be placed at other points in the network, such as connection point 123 between the network 102 and the interface 118 of the client 104, or some other location, as indicated schematically by connection point 125 somewhere in network 102. Not shown is a network packet acquisition device at the location 123 on the network for converting the physical information on the network into packets for input into monitor 108. Such packet acquisition devices are common.

Various protocols may be employed by the network to establish and maintain the required communication, *e.g.*, TCP/IP, etc. Any network activity—for example an application program run by the client 104 (CLIENT 1) communicating with another running on the server 110 (SERVER 2)—will produce an exchange of a sequence of packets over network 102 that is characteristic of the respective programs and of the network protocols. Such characteristics may not be completely revealing at the individual packet level. It may require the analyzing of many packets by the monitor 108 to have enough information needed to recognize particular application programs. The packets may need to be parsed then analyzed in the context of various protocols, for

example, the transport through the application session layer protocols for packets of a type conforming to the ISO layered network model.

Communication protocols are layered, which is also referred to as a protocol stack. The ISO (International Standardization Organization) has defined a general model that provides a framework for design of communication protocol layers. This model, shown in table form below, serves as a basic reference for understanding the functionality of existing communication protocols.

ISO MODEL

Layer	Functionality	Example
7	Application	Telnet, NFS, Novell NCP, HTTP, H.323
6	Presentation	XDR
5	Session	RPC, NETBIOS, SNMP, <i>etc.</i>
4	Transport	TCP, Novel SPX, UDP, <i>etc.</i>
3	Network	IP, Novell IPX, VIP, AppleTalk, <i>etc.</i>
2	Data Link	Network Interface Card (Hardware Interface). MAC layer
1	Physical	Ethernet, Token Ring, Frame Relay, ATM, T1 (Hardware Connection)

Different communication protocols employ different levels of the ISO model or may use a layered model that is similar to but which does not exactly conform to the ISO model. A protocol in a certain layer may not be visible to protocols employed at other layers. For example, an application (Level 7) may not be able to identify the source computer for a communication attempt (Levels 2–3).

In some communication arts, the term “frame” generally refers to encapsulated data at OSI layer 2, including a destination address, control bits for flow control, the data or payload, and CRC (cyclic redundancy check) data for error checking. The term

“packet” generally refers to encapsulated data at OSI layer 3. In the TCP/IP world, the term “datagram” is also used. In this specification, the term “packet” is intended to encompass packets, datagrams, frames, and cells. In general, a packet format or frame format refers to how data is encapsulated with various fields and headers for
5 transmission across a network. For example, a data packet typically includes an address destination field, a length field, an error correcting code (ECC) field, or cyclic redundancy check (CRC) field, as well as headers and footers to identify the beginning and end of the packet. The terms “packet format” and “frame format,” also referred to as “cell format,” are generally synonymous.

10 Monitor 108 looks at every packet passing the connection point 121 for analysis. However, not every packet carries the same information useful for recognizing all levels of the protocol. For example, in a conversational flow associated with a particular application, the application will cause the server to send a type-A packet, but so will another. If, though, the particular application program always follows a type-A packet
15 with the sending of a type-B packet, and the other application program does not, then in order to recognize packets of that application’s conversational flow, the monitor can be available to recognize packets that match the type-B packet to associate with the type-A packet. If such is recognized after a type-A packet, then the particular application program’s conversational flow has started to reveal itself to the monitor 108.

20 Further packets may need to be examined before the conversational flow can be identified as being associated with the application program. Typically, monitor 108 is simultaneously also in partial completion of identifying other packet exchanges that are parts of conversational flows associated with other applications. One aspect of monitor 108 is its ability to maintain the state of a flow. The state of a flow is an indication of all
25 previous events in the flow that lead to recognition of the content of all the protocol levels, *e.g.*, the ISO model protocol levels. Another aspect of the invention is forming a signature of extracted characteristic portions of the packet that can be used to rapidly identify packets belonging to the same flow.

In real-world uses of the monitor 108, the number of packets on the network 102
30 passing by the monitor 108’s connection point can exceed a million per second. Consequently, the monitor has very little time available to analyze and type each packet

and identify and maintain the state of the flows passing through the connection point. The monitor 108 therefore masks out all the unimportant parts of each packet that will not contribute to its classification. However, the parts to mask-out will change with each packet depending on which flow it belongs to and depending on the state of the flow.

5 The recognition of the packet type, and ultimately of the associated application programs according to the packets that their executions produce, is a multi-step process within the monitor 108. At a first level, for example, several application programs will all produce a first kind of packet. A first "signature" is produced from selected parts of a packet that will allow monitor 108 to identify efficiently any packets that belong to the
10 same flow. In some cases, that packet type may be sufficiently unique to enable the monitor to identify the application that generated such a packet in the conversational flow. The signature can then be used to efficiently identify all future packets generated in traffic related to that application.

 In other cases, that first packet only starts the process of analyzing the
15 conversational flow, and more packets are necessary to identify the associated application program. In such a case, a subsequent packet of a second type—but that
 potentially belongs to the same conversational flow—is recognized by using the signature. At such a second level, then, only a few of those application programs will have conversational flows that can produce such a second packet type. At this level in
20 the process of classification, all application programs that are not in the set of those that lead to such a sequence of packet types may be excluded in the process of classifying the conversational flow that includes these two packets. Based on the known patterns for the protocol and for the possible applications, a signature is produced that allows recognition of any future packets that may follow in the conversational flow.

25 It may be that the application is now recognized, or recognition may need to proceed to a third level of analysis using the second level signature. For each packet, therefore, the monitor parses the packet and generates a signature to determine if this signature identified a previously encountered flow, or shall be used to recognize future packets belonging to the same conversational flow. In real time, the packet is further
30 analyzed in the context of the sequence of previously encountered packets (the state), and of the possible future sequences such a past sequence may generate in conversational

flows associated with different applications. A new signature for recognizing future packets may also be generated. This process of analysis continues until the applications are identified. The last generated signature may then be used to efficiently recognize future packets associated with the same conversational flow. Such an arrangement makes it possible for the monitor 108 to cope with millions of packets per second that must be inspected.

Another aspect of the invention is adding Eavesdropping. In alternative embodiments of the present invention capable of eavesdropping, once the monitor 108 has recognized the executing application programs passing through some point in the network 102 (for example, because of execution of the applications by the client 105 or server 110), the monitor sends a message to some general purpose processor on the network that can input the same packets from the same location on the network, and the processor then loads its own executable copy of the application program and uses it to read the content being exchanged over the network. In other words, once the monitor 108 has accomplished recognition of the application program, eavesdropping can commence.

The Network Monitor

FIG. 3 shows a network packet monitor 300, in an embodiment of the present invention that can be implemented with computer hardware and/or software. The system 300 is similar to monitor 108 in FIG. 1. A packet 302 is examined, *e.g.*, from a packet acquisition device at the location 121 in network 102 (FIG. 1), and the packet evaluated, for example in an attempt to determine its characteristics, *e.g.*, all the protocol information in a multilevel model, including what server application produced the packet.

The packet acquisition device is a common interface that converts the physical signals and then decodes them into bits, and into packets, in accordance with the particular network (Ethernet, frame relay, ATM, *etc.*). The acquisition device indicates to the monitor 108 the type of network of the acquired packet or packets.

Aspects shown here include: (1) the initialization of the monitor to generate what operations need to occur on packets of different types—accomplished by compiler and optimizer 310, (2) the processing—parsing and extraction of selected portions—of packets to generate an identifying signature—accomplished by parser subsystem 301,

and (3) the analysis of the packets—accomplished by analyzer 303.

The purpose of compiler and optimizer 310 is to provide protocol specific information to parser subsystem 301 and to analyzer subsystem 303. The initialization occurs prior to operation of the monitor, and only needs to re-occur when new protocols
5 are to be added.

A flow is a stream of packets being exchanged between any two addresses in the network. For each protocol there are known to be several fields, such as the destination (recipient), the source (the sender), and so forth, and these and other fields are used in monitor 300 to identify the flow. There are other fields not important for identifying the
10 flow, such as checksums, and those parts are not used for identification.

Parser subsystem 301 examines the packets using pattern recognition process 304 that parses the packet and determines the protocol types and associated headers for each protocol layer that exists in the packet 302. An extraction process 306 in parser subsystem 301 extracts characteristic portions (signature information) from the packet
15 302. Both the pattern information for parsing and the related extraction operations, *e.g.*, extraction masks, are supplied from a parsing-pattern-structures and extraction-operations database (parsing/extractions database) 308 filled by the compiler and optimizer 310.

The protocol description language (PDL) files 336 describes both patterns and
20 states of all protocols that an occur at any layer, including how to interpret header information, how to determine from the packet header information the protocols at the next layer, and what information to extract for the purpose of identifying a flow, and ultimately, applications and services. The layer selections database 338 describes the particular layering handled by the monitor. That is, what protocols run on top of what
25 protocols at any layer level. Thus 336 and 338 combined describe how one would decode, analyze, and understand the information in packets, and, furthermore, how the information is layered. This information is input into compiler and optimizer 310.

When compiler and optimizer 310 executes, it generates two sets of internal data structures. The first is the set of parsing/extraction operations 308. The pattern structures
30 include parsing information and describe what will be recognized in the headers of packets; the extraction operations are what elements of a packet are to be extracted from

the packets based on the patterns that get matched. Thus, database 308 of parsing/extraction operations includes information describing how to determine a set of one or more protocol dependent extraction operations from data in the packet that indicate a protocol used in the packet.

5 The other internal data structure that is built by compiler 310 is the set of state patterns and processes 326. These are the different states and state transitions that occur in different conversational flows, and the state operations that need to be performed (*e.g.*, patterns that need to be examined and new signatures that need to be built) during any state of a conversational flow to further the task of analyzing the conversational flow.

10 Thus, compiling the PDL files and layer selections provides monitor 300 with the information it needs to begin processing packets. In an alternate embodiment, the contents of one or more of databases 308 and 326 may be manually or otherwise generated. Note that in some embodiments the layering selections information is inherent rather than explicitly described. For example, since a PDL file for a protocol includes the
15 child protocols, the parent protocols also may be determined.

In the preferred embodiment, the packet 302 from the acquisition device is input into a packet buffer. The pattern recognition process 304 is carried out by a pattern analysis and recognition (PAR) engine that analyzes and recognizes patterns in the packets. In particular, the PAR locates the next protocol field in the header and
20 determines the length of the header, and may perform certain other tasks for certain types of protocol headers. An example of this is type and length comparison to distinguish an IEEE 802.3 (Ethernet) packet from the older type 2 (or Version 2) Ethernet packet, also called a DIGITAL-Intel-Xerox (DIX) packet. The PAR also uses the pattern structures and extraction operations database 308 to identify the next protocol and parameters
25 associated with that protocol that enables analysis of the next protocol layer. Once a pattern or a set of patterns has been identified, it/they will be associated with a set of none or more extraction operations. These extraction operations (in the form of commands and associated parameters) are passed to the extraction process 306 implemented by an extracting and information identifying (EII) engine that extracts
30 selected parts of the packet, including identifying information from the packet as required for recognizing this packet as part of a flow. The extracted information is put in

sequence and then processed in block 312 to build a unique flow signature (also called a “key”) for this flow. A flow signature depends on the protocols used in the packet. For some protocols, the extracted components may include source and destination addresses. For example, Ethernet frames have end-point addresses that are useful in building a better flow signature. Thus, the signature typically includes the client and server address pairs. The signature is used to recognize further packets that are or may be part of this flow.

In the preferred embodiment, the building of the flow key includes generating a hash of the signature using a hash function. The purpose of using such a hash is conventional—to spread flow-entries identified by the signature across a database for efficient searching. The hash generated is preferably based on a hashing algorithm and such hash generation is known to those in the art.

In one embodiment, the parser passes data from the packet—a parser record—that includes the signature (i.e., selected portions of the packet), the hash, and the packet itself to allow for any state processing that requires further data from the packet. An improved embodiment of the parser subsystem might generate a parser record that has some predefined structure and that includes the signature, the hash, some flags related to some of the fields in the parser record, and parts of the packet’s payload that the parser subsystem has determined might be required for further processing, e.g., for state processing.

Note that alternate embodiments may use some function other than concatenation of the selected portions of the packet to make the identifying signature. For example, some “digest function” of the concatenated selected portions may be used.

The parser record is passed onto lookup process 314 which looks in an internal data store of records of known flows that the system has already encountered, and decides (in 316) whether or not this particular packet belongs to a known flow as indicated by the presence of a flow-entry matching this flow in a database of known flows 324. A record in database 324 is associated with each encountered flow.

The parser record enters a buffer called the unified flow key buffer (UFKB). The UFKB stores the data on flows in a data structure that is similar to the parser record, but that includes a field that can be modified. In particular, one or the UFKB record fields

stores the packet sequence number, and another is filled with state information in the form of a program counter for a state processor that implements state processing 328.

The determination (316) of whether a record with the same signature already exists is carried out by a lookup engine (LUE) that obtains new UFKB records and uses the hash in the UFKB record to lookup if there is a matching known flow. In the particular embodiment, the database of known flows 324 is in an external memory. A cache is associated with the database 324. A lookup by the LUE for a known record is carried out by accessing the cache using the hash, and if the entry is not already present in the cache, the entry is looked up (again using the hash) in the external memory.

10 The flow-entry database 324 stores flow-entries that include the unique flow-signature, state information, and extracted information from the packet for updating flows, and one or more statistical about the flow. Each entry completely describes a flow. Database 324 is organized into bins that contain a number, denoted N, of flow-entries (also called flow-entries, each a bucket), with N being 4 in the preferred embodiment. Buckets (i.e., flow-entries) are accessed via the hash of the packet from the parser subsystem 301 (i.e., the hash in the UFKB record). The hash spreads the flows across the database to allow for fast lookups of entries, allowing shallower buckets. The designer selects the bucket depth N based on the amount of memory attached to the monitor, and the number of bits of the hash data value used. For example, in one embodiment, each flow-entry is 128 bytes long, so for 128K flow-entries, 16 Mbytes are required. Using a 16-bit hash gives two flow-entries per bucket. Empirically, this has been shown to be more than adequate for the vast majority of cases. Note that another embodiment uses flow-entries that are 256 bytes long.

25 Herein, whenever an access to database 324 is described, it is to be understood that the access is via the cache, unless otherwise stated or clear from the context.

If there is no flow-entry found matching the signature, i.e., the signature is for a new flow, then a protocol and state identification process 318 further determines the state and protocol. That is, process 318 determines the protocols and where in the state sequence for a flow for this protocol's this packet belongs. Identification process 318 uses the extracted information and makes reference to the database 326 of state patterns and processes. Process 318 is then followed by any state operations that need to be

executed on this packet by a state processor 328.

If the packet is found to have a matching flow-entry in the database 324 (e.g., in the cache), then a process 320 determines, from the looked-up flow-entry, if more classification by state processing of the flow signature is necessary. If not, a process 322
5 updates the flow-entry in the flow-entry database 324 (e.g., via the cache). Updating includes updating one or more statistical measures stored in the flow-entry. In our embodiment, the statistical measures are stored in counters in the flow-entry.

If state processing is required, state process 328 is commenced. State processor 328 carries out any state operations specified for the state of the flow and updates the
10 state to the next state according to a set of state instructions obtained from the state pattern and processes database 326.

The state processor 328 analyzes both new and existing flows in order to analyze all levels of the protocol stack, ultimately classifying the flows by application (level 7 in the ISO model). It does this by proceeding from state-to-state based on predefined state
15 transition rules and state operations as specified in state processor instruction database 326. A state transition rule is a rule typically containing a test followed by the next-state to proceed to if the test result is true. An operation is an operation to be performed while the state processor is in a particular state—for example, in order to evaluate a quantity needed to apply the state transition rule. The state processor goes through each rule and
20 each state process until the test is true, or there are no more tests to perform.

In general, the set of state operations may be none or more operations on a packet, and carrying out the operation or operations may leave one in a state that causes exiting the system prior to completing the identification, but possibly knowing more about what state and state processes are needed to execute next, *i.e.*, when a next packet
25 of this flow is encountered. As an example, a state process (set of state operations) at a particular state may build a new signature for future recognition packets of the next state.

By maintaining the state of the flows and knowing that new flows may be set up using the information from previously encountered flows, the network traffic monitor 300 provides for (a) single-packet protocol recognition of flows, and (b) multiple-packet
30 protocol recognition of flows. Monitor 300 can even recognize the application program from one or more disjointed sub-flows that occur in server announcement type flows.

What may seem to prior art monitors to be some unassociated flow, may be recognized by the inventive monitor using the flow signature to be a sub-flow associated with a previously encountered sub-flow.

Thus, state processor 328 applies the first state operation to the packet for this particular flow-entry. A process 330 decides if more operations need to be performed for this state. If so, the analyzer continues looping between block 330 and 328 applying additional state operations to this particular packet until all those operations are completed—that is, there are no more operations for this packet in this state. A process 332 decides if there are further states to be analyzed for this type of flow according to the state of the flow and the protocol, in order to fully characterize the flow. If not, the conversational flow has now been fully characterized and a process 334 finalizes the classification of the conversational flow for the flow.

In the particular embodiment, the state processor 328 starts the state processing by using the last protocol recognized by the parser as an offset into a jump table (jump vector). The jump table finds the state processor instructions to use for that protocol in the state patterns and processes database 326. Most instructions test something in the unified flow key buffer, or the flow-entry in the database of known flows 324, if the entry exists. The state processor may have to test bits, do comparisons, add, or subtract to perform the test. For example, a common operation carried out by the state processor is searching for one or more patterns in the payload part of the UFKB.

Thus, in 332 in the classification, the analyzer decides whether the flow is at an end state. If not at an end state, the flow-entry is updated (or created if a new flow) for this flow-entry in process 322.

Furthermore, if the flow is known and if in 332 it is determined that there are further states to be processed using later packets, the flow-entry is updated in process 322.

The flow-entry also is updated after classification finalization so that any further packets belonging to this flow will be readily identified from their signature as belonging to this fully analyzed conversational flow.

After updating, database 324 therefore includes the set of all the conversational flows that have occurred.

Thus, the embodiment of present invention shown in FIG. 3 automatically maintains flow-entries, which in one aspect includes storing states. The monitor of FIG. 3 also generates characteristic parts of packets—the signatures—that can be used to recognize flows. The flow-entries may be identified and accessed by their signatures. Once a packet is identified to be from a known flow, the state of the flow is known and this knowledge enables state transition analysis to be performed in real time for each different protocol and application. In a complex analysis, state transitions are traversed as more and more packets are examined. Future packets that are part of the same conversational flow have their state analysis continued from a previously achieved state. When enough packets related to an application of interest have been processed, a final recognition state is ultimately reached, *i.e.*, a set of states has been traversed by state analysis to completely characterize the conversational flow. The signature for that final state enables each new incoming packet of the same conversational flow to be individually recognized in real time.

In this manner, one of the great advantages of the present invention is realized. Once a particular set of state transitions has been traversed for the first time and ends in a final state, a short-cut recognition pattern—a signature—can be generated that will key on every new incoming packet that relates to the conversational flow. Checking a signature involves a simple operation, allowing high packet rates to be successfully monitored on the network.

In improved embodiments, several state analyzers are run in parallel so that a large number of protocols and applications may be checked for. Every known protocol and application will have at least one unique set of state transitions, and can therefore be uniquely identified by watching such transitions.

When each new conversational flow starts, signatures that recognize the flow are automatically generated on-the-fly, and as further packets in the conversational flow are encountered, signatures are updated and the states of the set of state transitions for any potential application are further traversed according to the state transition rules for the flow. The new states for the flow—those associated with a set of state transitions for one

or more potential applications—are added to the records of previously encountered states for easy recognition and retrieval when a new packet in the flow is encountered.

Detailed operation

FIG. 4 diagrams an initialization system 400 that includes the compilation
 5 process. That is, part of the initialization generates the pattern structures and extraction operations database 308 and the state instruction database 328. Such initialization can occur off-line or from a central location.

The different protocols that can exist in different layers may be thought of as
 nodes of one or more trees of linked nodes. The packet type is the root of a tree (called
 10 level 0). Each protocol is either a parent node or a terminal node. A parent node links a protocol to other protocols (child protocols) that can be at higher layer levels. Thus a protocol may have zero or more children. Ethernet packets, for example, have several variants, each having a basic format that remains substantially the same. An Ethernet packet (the root or level 0 node) may be an Ethertype packet—also called an Ethernet
 15 Type/Version 2 and a DIX (DIGITAL-Intel-Xerox packet)—or an IEEE 803.2 packet. Continuing with the IEEE 802.3 packet, one of the children nodes may be the IP protocol, and one of the children of the IP protocol may be the TCP protocol.

FIG. 16 shows the header 1600 (base level 1) of a complete Ethernet frame (*i.e.*,
 packet) of information and includes information on the destination media access control
 20 address (Dst MAC 1602) and the source media access control address (Src MAC 1604). Also shown in FIG. 16 is some (but not all) of the information specified in the PDL files for extraction the signature.

FIG. 17A now shows the header information for the next level (level-2) for an
 Ethertype packet 1700. For an Ethertype packet 1700, the relevant information from the
 25 packet that indicates the next layer level is a two-byte type field 1702 containing the child recognition pattern for the next level. The remaining information 1704 is shown hatched because it not relevant for this level. The list 1712 shows the possible children for an Ethertype packet as indicated by what child recognition pattern is found offset 12. FIG. 17B shows the structure of the header of one of the possible next levels, that of the
 30 IP protocol. The possible children of the IP protocol are shown in table 1752.

The pattern, parse, and extraction database (pattern recognition database, or PRD) 308 generated by compilation process 310, in one embodiment, is in the form of a three dimensional structure that provides for rapidly searching packet headers for the next protocol. FIG. 18A shows such a 3-D representation 1800 (which may be
5 considered as an indexed set of 2-D representations). A compressed form of the 3-D structure is preferred.

An alternate embodiment of the data structure used in database 308 is illustrated in FIG. 18B. Thus, like the 3-D structure of FIG. 18A, the data structure permits rapid searches to be performed by the pattern recognition process 304 by indexing locations in
10 a memory rather than performing address link computations. In this alternate embodiment, the PRD 308 includes two parts, a single protocol table 1850 (PT) which has an entry for each protocol known for the monitor, and a series of Look Up Tables 1870 (LUT's) that are used to identify known protocols and their children. The protocol table includes the parameters needed by the pattern analysis and recognition process 304
15 (implemented by PRE 1006) to evaluate the header information in the packet that is associated with that protocol, and parameters needed by extraction process 306 (implemented by slicer 1007) to process the packet header. When there are children, the PT describes which bytes in the header to evaluate to determine the child protocol. In particular, each PT entry contains the header length, an offset to the child, a slicer
20 command, and some flags.

The pattern matching is carried out by finding particular "child recognition codes" in the header fields, and using these codes to index one or more of the LUT's. Each LUT entry has a node code that can have one of four values, indicating the protocol that has been recognized, a code to indicate that the protocol has been partially
25 recognized (more LUT lookups are needed), a code to indicate that this is a terminal node, and a null node to indicate a null entry. The next LUT to lookup is also returned from a LUT lookup.

Compilation process is described in FIG. 4. The source-code information in the form of protocol description files is shown as 402. In the particular embodiment, the
30 high level decoding descriptions includes a set of protocol description files 336, one for each protocol, and a set of packet layer selections 338, which describes the particular

layering (sets of trees of protocols) that the monitor is to be able to handle.

A compiler 403 compiles the descriptions. The set of packet parse-and-extract operations 406 is generated (404), and a set of packet state instructions and operations 407 is generated (405) in the form of instructions for the state processor that implements state processing process 328. Data files for each type of application and protocol to be recognized by the analyzer are downloaded from the pattern, parse, and extraction database 406 into the memory systems of the parser and extraction engines. (See the parsing process 500 description and FIG. 5; the extraction process 600 description and FIG. 6; and the parsing subsystem hardware description and FIG. 10). Data files for each type of application and protocol to be recognized by the analyzer are also downloaded from the state-processor instruction database 407 into the state processor. (see the state processor 1108 description and FIG. 11.).

Note that generating the packet parse and extraction operations builds and links the three dimensional structure (one embodiment) or the or all the lookup tables for the PRD.

Because of the large number of possible protocol trees and subtrees, the compiler process 400 includes optimization that compares the trees and subtrees to see which children share common parents. When implemented in the form of the LUT's, this process can generate a single LUT from a plurality of LUT's. The optimization process further includes a compaction process that reduces the space needed to store the data of the PRD.

As an example of compaction, consider the 3-D structure of FIG. 18A that can be thought of as a set of 2-D structures each representing a protocol. To enable saving space by using only one array per protocol which may have several parents, in one embodiment, the pattern analysis subprocess keeps a "current header" pointer. Each location (offset) index for each protocol 2-D array in the 3-D structure is a relative location starting with the start of header for the particular protocol. Furthermore, each of the two-dimensional arrays is sparse. The next step of the optimization, is checking all the 2-D arrays against all the other 2-D arrays to find out which ones can share memory. Many of these 2-D arrays are often sparsely populated in that they each have only a small number of valid entries. So, a process of "folding" is next used to combine two or more

2-D arrays together into one physical 2-D array without losing the identity of any of the original 2-D arrays (i.e., all the 2-D arrays continue to exist logically). Folding can occur between any 2-D arrays irrespective of their location in the tree as long as certain conditions are met. Multiple arrays may be combined into a single array as long as the individual entries do not conflict with each other. A fold number is then used to associate each element with its original array. A similar folding process is used for the set of LUTs 1850 in the alternate embodiment of FIG. 18B.

In 410, the analyzer has been initialized and is ready to perform recognition.

FIG. 5 shows a flowchart of how actual parser subsystem 301 functions. Starting at 501, the packet 302 is input to the packet buffer in step 502. Step 503 loads the next (initially the first) packet component from the packet 302. The packet components are extracted from each packet 302 one element at a time. A check is made (504) to determine if the load-packet-component operation 503 succeeded, indicating that there was more in the packet to process. If not, indicating all components have been loaded, the parser subsystem 301 builds the packet signature (512)—the next stage (FIG 6).

If a component is successfully loaded in 503, the node and processes are fetched (505) from the pattern, parse and extraction database 308 to provide a set of patterns and processes for that node to apply to the loaded packet component. The parser subsystem 301 checks (506) to determine if the fetch pattern node operation 505 completed successfully, indicating there was a pattern node that loaded in 505. If not, step 511 moves to the next packet component. If yes, then the node and pattern matching process are applied in 507 to the component extracted in 503. A pattern match obtained in 507 (as indicated by test 508) means the parser subsystem 301 has found a node in the parsing elements; the parser subsystem 301 proceeds to step 509 to extract the elements.

If applying the node process to the component does not produce a match (test 508), the parser subsystem 301 moves (510) to the next pattern node from the pattern database 308 and to step 505 to fetch the next node and process. Thus, there is an “applying patterns” loop between 508 and 505. Once the parser subsystem 301 completes all the patterns and has either matched or not, the parser subsystem 301 moves to the next packet component (511).

Once all the packet components have been the loaded and processed from the

input packet 302, then the load packet will fail (indicated by test 504), and the parser subsystem 301 moves to build a packet signature which is described in FIG. 6

FIG. 6 is a flow chart for extracting the information from which to build the packet signature. The flow starts at 601, which is the exit point 513 of FIG. 5. At this point parser subsystem 301 has a completed packet component and a pattern node available in a buffer (602). Step 603 loads the packet component available from the pattern analysis process of FIG. 5. If the load completed (test 604), indicating that there was indeed another packet component, the parser subsystem 301 fetches in 605 the extraction and process elements received from the pattern node component in 602. If the fetch was successful (test 606), indicating that there are extraction elements to apply, the parser subsystem 301 in step 607 applies that extraction process to the packet component based on an extraction instruction received from that pattern node. This removes and saves an element from the packet component.

In step 608, the parser subsystem 301 checks if there is more to extract from this component, and if not, the parser subsystem 301 moves back to 603 to load the next packet component at hand and repeats the process. If the answer is yes, then the parser subsystem 301 moves to the next packet component ratchet. That new packet component is then loaded in step 603. As the parser subsystem 301 moved through the loop between 608 and 603, extra extraction processes are applied either to the same packet component if there is more to extract, or to a different packet component if there is no more to extract.

The extraction process thus builds the signature, extracting more and more components according to the information in the patterns and extraction database 308 for the particular packet. Once loading the next packet component operation 603 fails (test 604), all the components have been extracted. The built signature is loaded into the signature buffer (610) and the parser subsystem 301 proceeds to FIG. 7 to complete the signature generation process.

Referring now to FIG. 7, the process continues at 701. The signature buffer and the pattern node elements are available (702). The parser subsystem 301 loads the next pattern node element. If the load was successful (test 704) indicating there are more nodes, the parser subsystem 301 in 705 hashes the signature buffer element based on the

hash elements that are found in the pattern node that is in the element database. In 706 the resulting signature and the hash are packed. In 707 the parser subsystem 301 moves on to the next packet component which is loaded in 703.

The 703 to 707 loop continues until there are no more patterns of elements left
5 (test 704). Once all the patterns of elements have been hashed, processes 304, 306 and 312 of parser subsystem 301 are complete. Parser subsystem 301 has generated the signature used by the analyzer subsystem 303.

A parser record is loaded into the analyzer, in particular, into the UFKB in the form of a UFKB record which is similar to a parser record, but with one or more
10 different fields.

FIG. 8 is a flow diagram describing the operation of the lookup/update engine (LUE) that implements lookup operation 314. The process starts at 801 from FIG. 7 with the parser record that includes a signature, the hash and at least parts of the payload. In 802 those elements are shown in the form of a UFKB-entry in the buffer. The LUE, the
15 lookup engine 314 computes a "record bin number" from the hash for a flow-entry. A bin herein may have one or more "buckets" each containing a flow-entry. The preferred embodiment has four buckets per bin.

Since preferred hardware embodiment includes the cache, all data accesses to records in the flowchart of FIG. 8 are stated as being to or from the cache.

20 Thus, in 804, the system looks up the cache for a bucket from that bin using the hash. If the cache successfully returns with a bucket from the bin number, indicating there are more buckets in the bin, the lookup/update engine compares (807) the current signature (the UFKB-entry's signature) from that in the bucket (i.e., the flow-entry signature). If the signatures match (test 808), that record (in the cache) is marked in step
25 810 as "in process" and a timestamp added. Step 811 indicates to the UFKB that the UFKB-entry in 802 has a status of "found." The "found" indication allows the state processing 328 to begin processing this UFKB element. The preferred hardware embodiment includes one or more state processors, and these can operate in parallel with the lookup/update engine.

30 In the preferred embodiment, a set of statistical operations is performed by a

calculator for every packet analyzed. The statistical operations may include one or more of counting the packets associated with the flow; determining statistics related to the size of packets of the flow; compiling statistics on differences between packets in each direction, for example using timestamps; and determining statistical relationships of
5 timestamps of packets in the same direction. The statistical measures are kept in the flow-entries. Other statistical measures also may be compiled. These statistics may be used singly or in combination by a statistical processor component to analyze many different aspects of the flow. This may include determining network usage metrics from the statistical measures, for example to ascertain the network's ability to transfer
10 information for this application. Such analysis provides for measuring the quality of service of a conversation, measuring how well an application is performing in the network, measuring network resources consumed by an application, and so forth.

To provide for such analyses, the lookup/update engine updates one or more counters that are part of the flow-entry (in the cache) in step 812. The process exits at
15 813. In our embodiment, the counters include the total packets of the flow, the time, and a differential time from the last timestamp to the present timestamp.

It may be that the bucket of the bin did not lead to a signature match (test 808). In such a case, the analyzer in 809 moves to the next bucket for this bin. Step 804 again looks up the cache for another bucket from that bin. The lookup/update engine thus
20 continues lookup up buckets of the bin until there is either a match in 808 or operation 804 is not successful (test 805), indicating that there are no more buckets in the bin and no match was found.

If no match was found, the packet belongs to a new (not previously encountered) flow. In 806 the system indicates that the record in the unified flow key buffer for this
25 packet is new, and in 812, any statistical updating operations are performed for this packet by updating the flow-entry in the cache. The update operation exits at 813. A flow insertion/deletion engine (FIDE) creates a new record for this flow (again via the cache).

Thus, the update/lookup engine ends with a UFKB-entry for the packet with a "new" status or a "found" status.

30 Note that the above system uses a hash to which more than one flow-entry can match. A longer hash may be used that corresponds to a single flow-entry. In such an

embodiment, the flow chart of FIG. 8 is simplified as would be clear to those in the art.

The hardware system

Each of the individual hardware elements through which the data flows in the system are now described with reference to FIGS. 10 and 11. Note that while we are
5 describing a particular hardware implementation of the invention embodiment of FIG. 3, it would be clear to one skilled in the art that the flow of FIG. 3 may alternatively be implemented in software running on one or more general-purpose processors, or only partly implemented in hardware. An implementation of the invention that can operate in software is shown in FIG. 14. The hardware embodiment (FIGS. 10 and 11) can operate
10 at over a million packets per second, while the software system of FIG. 14 may be suitable for slower networks. To one skilled in the art it would be clear that more and more of the system may be implemented in software as processors become faster.

FIG. 10 is a description of the parsing subsystem (301, shown here as subsystem
15 1000) as implemented in hardware. Memory 1001 is the pattern recognition database memory, in which the patterns that are going to be analyzed are stored. Memory 1002 is the extraction-operation database memory, in which the extraction instructions are stored. Both 1001 and 1002 correspond to internal data structure 308 of FIG. 3. Typically, the system is initialized from a microprocessor (not shown) at which time these memories are loaded through a host interface multiplexor and control register 1005
20 via the internal buses 1003 and 1004. Note that the contents of 1001 and 1002 are preferably obtained by compiling process 310 of FIG. 3.

A packet enters the parsing system via 1012 into a parser input buffer memory
1008 using control signals 1021 and 1023, which control an input buffer interface
25 controller 1022. The buffer 1008 and interface control 1022 connect to a packet acquisition device (not shown). The buffer acquisition device generates a packet start signal 1021 and the interface control 1022 generates a next packet (i.e., ready to receive data) signal 1023 to control the data flow into parser input buffer memory 1008. Once a packet starts loading into the buffer memory 1008, pattern recognition engine (PRE)
1006 carries out the operations on the input buffer memory described in block 304 of
30 FIG. 3. That is, protocol types and associated headers for each protocol layer that exist in the packet are determined.

The PRE searches database 1001 and the packet in buffer 1008 in order to recognize the protocols the packet contains. In one implementation, the database 1001 includes a series of linked lookup tables. Each lookup table uses eight bits of addressing. The first lookup table is always at address zero. The Pattern Recognition Engine uses a
5 base packet offset from a control register to start the comparison. It loads this value into a current offset pointer (COP). It then reads the byte at base packet offset from the parser input buffer and uses it as an address into the first lookup table.

Each lookup table returns a word that links to another lookup table or it returns a terminal flag. If the lookup produces a recognition event the database also returns a
10 command for the slicer. Finally it returns the value to add to the COP.

The PRE 1006 includes of a comparison engine. The comparison engine has a first stage that checks the protocol type field to determine if it is an 802.3 packet and the field should be treated as a length. If it is not a length, the protocol is checked in a second stage. The first stage is the only protocol level that is not programmable. The
15 second stage has two full sixteen bit content addressable memories (CAMs) defined for future protocol additions.

Thus, whenever the PRE recognizes a pattern, it also generates a command for the extraction engine (also called a "slicer") 1007. The recognized patterns and the commands are sent to the extraction engine 1007 that extracts information from the
20 packet to build the parser record. Thus, the operations of the extraction engine are those carried out in blocks 306 and 312 of FIG. 3. The commands are sent from PRE 1006 to slicer 1007 in the form of extraction instruction pointers which tell the extraction engine 1007 where to find the instructions in the extraction operations database memory (i.e., slicer instruction database) 1002.

25 Thus, when the PRE 1006 recognizes a protocol it outputs both the protocol identifier and a process code to the extractor. The protocol identifier is added to the flow signature and the process code is used to fetch the first instruction from the instruction database 1002. Instructions include an operation code and usually source and destination offsets as well as a length. The offsets and length are in bytes. A typical operation is the
30 MOVE instruction. This instruction tells the slicer 1007 to copy n bytes of data unmodified from the input buffer 1008 to the output buffer 1010. The extractor contains

a byte-wise barrel shifter so that the bytes moved can be packed into the flow signature. The extractor contains another instruction called HASH. This instruction tells the extractor to copy from the input buffer 1008 to the HASH generator.

Thus these instructions are for extracting selected element(s) of the packet in the input buffer memory and transferring the data to a parser output buffer memory 1010. Some instructions also generate a hash.

The extraction engine 1007 and the PRE operate as a pipeline. That is, extraction engine 1007 performs extraction operations on data in input buffer 1008 already processed by PRE 1006 while more (i.e., later arriving) packet information is being simultaneously parsed by PRE 1006. This provides high processing speed sufficient to accommodate the high arrival rate speed of packets.

Once all the selected parts of the packet used to form the signature are extracted, the hash is loaded into parser output buffer memory 1010. Any additional payload from the packet that is required for further analysis is also included. The parser output memory 1010 is interfaced with the analyzer subsystem by analyzer interface control 1011. Once all the information of a packet is in the parser output buffer memory 1010, a data ready signal 1025 is asserted by analyzer interface control. The data from the parser subsystem 1000 is moved to the analyzer subsystem via 1013 when an analyzer ready signal 1027 is asserted.

FIG. 11 shows the hardware components and dataflow for the analyzer subsystem that performs the functions of the analyzer subsystem 303 of FIG. 3. The analyzer is initialized prior to operation, and initialization includes loading the state processing information generated by the compilation process 310 into a database memory for the state processing, called state processor instruction database (SPID) memory 1109.

The analyzer subsystem 1100 includes a host bus interface 1122 using an analyzer host interface controller 1118, which in turn has access to a cache system 1115. The cache system has bi-directional access to and from the state processor of the system 1108. State processor 1108 is responsible for initializing the state processor instruction database memory 1109 from information given over the host bus interface 1122.

With the SPID 1109 loaded, the analyzer subsystem 1100 receives parser records

comprising packet signatures and payloads that come from the parser into the unified flow key buffer (UFKB) 1103. UFKB is comprised of memory set up to maintain UFKB records. A UFKB record is essentially a parser record; the UFKB holds records of packets that are to be processed or that are in process. Furthermore, the UFKB provides
5 for one or more fields to act as modifiable status flags to allow different processes to run concurrently.

Three processing engines run concurrently and access records in the UFKB 1103: the lookup/update engine (LUE) 1107, the state processor (SP) 1108, and the flow insertion and deletion engine (FIDE) 1110. Each of these is implemented by one or more
10 finite state machines (FSM's). There is bi-directional access between each of the finite state machines and the unified flow key buffer 1103. The UFKB record includes a field that stores the packet sequence number, and another that is filled with state information in the form of a program counter for the state processor 1108 that implements state
15 processing 328. The status flags of the UFKB for any entry includes that the LUE is done and that the LUE is transferring processing of the entry to the state processor. The LUE done indicator is also used to indicate what the next entry is for the LUE. There also is provided a flag to indicate that the state processor is done with the current flow and to indicate what the next entry is for the state processor. There also is provided a flag to indicate the state processor is transferring processing of the UFKB-entry to the flow
20 insertion and deletion engine.

A new UFKB record is first processed by the LUE 1107. A record that has been processed by the LUE 1107 may be processed by the state processor 1108, and a UFKB record data may be processed by the flow insertion/deletion engine 1110 after being
25 processed by the state processor 1108 or only by the LUE. Whether or not a particular engine has been applied to any unified flow key buffer entry is determined by status fields set by the engines upon completion. In one embodiment, a status flag in the UFKB-entry indicates whether an entry is new or found. In other embodiments, the LUE issues a flag to pass the entry to the state processor for processing, and the required operations for a new record are included in the SP instructions.

30 Note that each UFKB-entry may not need to be processed by all three engines. Furthermore, some UFKB entries may need to be processed more than once by a

particular engine.

Each of these three engines also has bi-directional access to a cache subsystem 1115 that includes a caching engine. Cache 1115 is designed to have information flowing in and out of it from five different points within the system: the three engines, external
5 memory via a unified memory controller (UMC) 1119 and a memory interface 1123, and a microprocessor via analyzer host interface and control unit (ACIC) 1118 and host interface bus (HIB) 1122. The analyzer microprocessor (or dedicated logic processor) can thus directly insert or modify data in the cache.

The cache subsystem 1115 is an associative cache that includes a set of content
10 addressable memory cells (CAMs) each including an address portion and a pointer portion pointing to the cache memory (e.g., RAM) containing the cached flow-entries. The CAMs are arranged as a stack ordered from a top CAM to a bottom CAM. The bottom CAM's pointer points to the least recently used (LRU) cache memory entry. Whenever there is a cache miss, the contents of cache memory pointed to by the bottom
15 CAM are replaced by the flow-entry from the flow-entry database 324. This now becomes the most recently used entry, so the contents of the bottom CAM are moved to the top CAM and all CAM contents are shifted down. Thus, the cache is an associative cache with a true LRU replacement policy.

The LUE 1107 first processes a UFKB-entry, and basically performs the
20 operation of blocks 314 and 316 in FIG. 3. A signal is provided to the LUE to indicate that a "new" UFKB-entry is available. The LUE uses the hash in the UFKB-entry to read a matching bin of up to four buckets from the cache. The cache system attempts to obtain the matching bin. If a matching bin is not in the cache, the cache 1115 makes the request to the UMC 1119 to bring in a matching bin from the external memory.

25 When a flow-entry is found using the hash, the LUE 1107 looks at each bucket and compares it using the signature to the signature of the UFKB-entry until there is a match or there are no more buckets.

If there is no match, or if the cache failed to provide a bin of flow-entries from the cache, a time stamp is set in the flow key of the UFKB record, a protocol
30 identification and state determination is made using a table that was loaded by compilation process 310 during initialization, the status for the record is set to indicate

the LUE has processed the record, and an indication is made that the UFKB-entry is ready to start state processing. The identification and state determination generates a protocol identifier which in the preferred embodiment is a "jump vector" for the state processor which is kept by the UFKB for this UFKB-entry and used by the state processor to start state processing for the particular protocol. For example, the jump vector jumps to the subroutine for processing the state.

If there was a match, indicating that the packet of the UFKB-entry is for a previously encountered flow, then a calculator component enters one or more statistical measures stored in the flow-entry, including the timestamp. In addition, a time difference from the last stored timestamp may be stored, and a packet count may be updated. The state of the flow is obtained from the flow-entry is examined by looking at the protocol identifier stored in the flow-entry of database 324. If that value indicates that no more classification is required, then the status for the record is set to indicate the LUE has processed the record. In the preferred embodiment, the protocol identifier is a jump vector for the state processor to a subroutine to state processing the protocol, and no more classification is indicated in the preferred embodiment by the jump vector being zero. If the protocol identifier indicates more processing, then an indication is made that the UFKB-entry is ready to start state processing and the status for the record is set to indicate the LUE has processed the record.

The state processor 1108 processes information in the cache system according to a UFKB-entry after the LUE has completed. State processor 1108 includes a state processor program counter SPPC that generates the address in the state processor instruction database 1109 loaded by compiler process 310 during initialization. It contains an Instruction Pointer (SPIP) which generates the SPID address. The instruction pointer can be incremented or loaded from a Jump Vector Multiplexor which facilitates conditional branching. The SPIP can be loaded from one of three sources: (1) A protocol identifier from the UFKB, (2) an immediate jump vector from the currently decoded instruction, or (3) a value provided by the arithmetic logic unit (SPALU) included in the state processor.

Thus, after a Flow Key is placed in the UFKB by the LUE with a known protocol identifier, the Program Counter is initialized with the last protocol recognized by the

Parser. This first instruction is a jump to the subroutine which analyzes the protocol that was decoded.

The State Processor ALU (SPALU) contains all the Arithmetic, Logical and String Compare functions necessary to implement the State Processor instructions. The main blocks of the SPALU are: The A and B Registers, the Instruction Decode & State
5 Machines, the String Reference Memory the Search Engine, an Output Data Register and an Output Control Register

The Search Engine in turn contains the Target Search Register set, the Reference Search Register set, and a Compare block which compares two operands by exclusive-
10 or-ing them together.

Thus, after the UFKB sets the program counter, a sequence of one or more state operations are executed in state processor 1108 to further analyze the packet that is in the flow key buffer entry for this particular packet.

FIG. 13 describes the operation of the state processor 1108. The state processor is entered at 1301 with a unified flow key buffer entry to be processed. The UFKB-entry is
15 new or corresponding to a found flow-entry. This UFKB-entry is retrieved from unified flow key buffer 1103 in 1301. In 1303, the protocol identifier for the UFKB-entry is used to set the state processor's instruction counter. The state processor 1108 starts the process by using the last protocol recognized by the parser subsystem 301 as an offset
20 into a jump table. The jump table takes us to the instructions to use for that protocol. Most instructions test something in the unified flow key buffer or the flow-entry if it exists. The state processor 1108 may have to test bits, do comparisons, add or subtract to perform the test.

The first state processor instruction is fetched in 1304 from the state processor
25 instruction database memory 1109. The state processor performs the one or more fetched operations (1304). In our implementation, each single state processor instruction is very primitive (e.g., a move, a compare, etc.), so that many such instructions need to be performed on each unified flow key buffer entry. One aspect of the state processor is its ability to search for one or more (up to four) reference strings in the payload part of the
30 UFKB entry. This is implemented by a search engine component of the state processor responsive to special searching instructions.

In 1307, a check is made to determine if there are any more instructions to be performed for the packet. If yes, then in 1308 the system sets the state processor instruction pointer (SPIP) to obtain the next instruction. The SPIP may be set by an immediate jump vector in the currently decoded instruction, or by a value provided by the SPALU during processing.

The next instruction to be performed is now fetched (1304) for execution. This state processing loop between 1304 and 1307 continues until there are no more instructions to be performed.

At this stage, a check is made in 1309 if the processing on this particular packet has resulted in a final state. That is, is the analyzer is done processing not only for this particular packet, but for the whole flow to which the packet belongs, and the flow is fully determined. If indeed there are no more states to process for this flow, then in 1311 the processor finalizes the processing. Some final states may need to put a state in place that tells the system to remove a flow—for example, if a connection disappears from a lower level connection identifier. In that case, in 1311, a flow removal state is set and saved in the flow-entry. The flow removal state may be a NOP (no-op) instruction which means there are no removal instructions.

Once the appropriate flow removal instruction as specified for this flow (a NOP or otherwise) is set and saved, the process is exited at 1313. The state processor now obtains another unified flow key buffer entry to process.

If at 1309 it is determined that processing for this flow is not completed, then in 1310 the system saves the state processor instruction pointer in the current flow-entry in the current flow-entry. That will be the next operation that will be performed the next time the LRE 1107 finds packet in the UFKB that matches this flow. The processor now exits processing this particular unified flow key buffer entry at 1313.

Note that state processing updates information in the unified flow key buffer 1103 and the flow-entry in the cache. Once the state processor is done, a flag is set in the UFKB for the entry that the state processor is done. Furthermore, if the flow needs to be inserted or deleted from the database of flows, control is then passed on to the flow insertion/deletion engine 1110 for that flow signature and packet entry. This is done by the state processor setting another flag in the UFKB for this UFKB-entry indicating that

the state processor is passing processing of this entry to the flow insertion and deletion engine.

The flow insertion and deletion engine 1110 is responsible for maintaining the flow-entry database. In particular, for creating new flows in the flow database, and
5 deleting flows from the database so that they can be reused.

The process of flow insertion is now described with the aid of FIG. 12. Flows are grouped into bins of buckets by the hash value. The engine processes a UFKB-entry that may be new or that the state processor otherwise has indicated needs to be created. FIG. 12 shows the case of a new entry being created. A conversation record bin
10 (preferably containing 4 buckets for four records) is obtained in 1203. This is a bin that matches the hash of the UFKB, so this bin may already have been sought for the UFKB-entry by the LUE. In 1204 the FIDE 1110 requests that the record bin/bucket be maintained in the cache system 1115. If in 1205 the cache system 1115 indicates that the bin/bucket is empty, step 1207 inserts the flow signature (with the hash) into the bucket
15 and the bucket is marked "used" in the cache engine of cache 1115 using a timestamp that is maintained throughout the process. In 1209, the FIDE 1110 compares the bin and bucket record flow signature to the packet to verify that all the elements are in place to complete the record. In 1211 the system marks the record bin and bucket as "in process" and as "new" in the cache system (and hence in the external memory). In 1212, the initial
20 statistical measures for the flow-record are set in the cache system. This in the preferred embodiment clears the set of counters used to maintain statistics, and may perform other procedures for statistical operations requires by the analyzer for the first packet seen for a particular flow.

Back in step 1205, if the bucket is not empty, the FIDE 1110 requests the next
25 bucket for this particular bin in the cache system. If this succeeds, the processes of 1207, 1209, 1211 and 1212 are repeated for this next bucket. If at 1208, there is no valid bucket, the unified flow key buffer entry for the packet is set as "drop," indicating that the system cannot process the particular packet because there are no buckets left in the system. The process exits at 1213. The FIDE 1110 indicates to the UFKB that the flow
30 insertion and deletion operations are completed for this UFKB-entry. This also lets the UFKB provide the FIDE with the next UFKB record.

Once a set of operations is performed on a unified flow key buffer entry by all of the engines required to access and manage a particular packet and its flow signature, the unified flow key buffer entry is marked as "completed." That element will then be used by the parser interface for the next packet and flow signature coming in from the parsing and extracting system.

All flow-entries are maintained in the external memory and some are maintained in the cache 1115. The cache system 1115 is intelligent enough to access the flow database and to understand the data structures that exists on the other side of memory interface 1123. The lookup/update engine 1107 is able to request that the cache system pull a particular flow or "buckets" of flows from the unified memory controller 1119 into the cache system for further processing. The state processor 1108 can operate on information found in the cache system once it is looked up by means of the lookup/update engine request, and the flow insertion/deletion engine 1110 can create new entries in the cache system if required based on information in the unified flow key buffer 1103. The cache retrieves information as required from the memory through the memory interface 1123 and the unified memory controller 1119, and updates information as required in the memory through the memory controller 1119.

There are several interfaces to components of the system external to the module of FIG. 11 for the particular hardware implementation. These include host bus interface 1122, which is designed as a generic interface that can operate with any kind of external processing system such as a microprocessor or a multiplexor (MUX) system. Consequently, one can connect the overall traffic classification system of FIGS. 11 and 12 into some other processing system to manage the classification system and to extract data gathered by the system.

The memory interface 1123 is designed to interface to any of a variety of memory systems that one may want to use to store the flow-entries. One can use different types of memory systems like regular dynamic random access memory (DRAM), synchronous DRAM, synchronous graphic memory (SGRAM), static random access memory (SRAM), and so forth.

FIG. 10 also includes some "generic" interfaces. There is a packet input interface 1012—a general interface that works in tandem with the signals of the input buffer

interface control 1022. These are designed so that they can be used with any kind of generic systems that can then feed packet information into the parser. Another generic interface is the interface of pipes 1031 and 1033 respectively out of and into host interface multiplexor and control registers 1005. This enables the parsing system to be managed by an external system, for example a microprocessor or another kind of external logic, and enables the external system to program and otherwise control the parser.

The preferred embodiment of this aspect of the invention is described in a hardware description language (HDL) such as VHDL or Verilog. It is designed and created in an HDL so that it may be used as a single chip system or, for instance, integrated into another general-purpose system that is being designed for purposes related to creating and analyzing traffic within a network. Verilog or other HDL implementation is only one method of describing the hardware.

In accordance with one hardware implementation, the elements shown in FIGS. 10 and 11 are implemented in a set of six field programmable logic arrays (FPGA's). The boundaries of these FPGA's are as follows. The parsing subsystem of FIG. 10 is implemented as two FPGAS; one FPGA, and includes blocks 1006, 1008 and 1012, parts of 1005, and memory 1001. The second FPGA includes 1002, 1007, 1013, 1011 parts of 1005. Referring to FIG. 11, the unified look-up buffer 1103 is implemented as a single FPGA. State processor 1108 and part of state processor instruction database memory 1109 is another FPGA. Portions of the state processor instruction database memory 1109 are maintained in external SRAM's. The lookup/update engine 1107 and the flow insertion/deletion engine 1110 are in another FPGA. The sixth FPGA includes the cache system 1115, the unified memory control 1119, and the analyzer host interface and control 1118.

Note that one can implement the system as one or more VSLI devices, rather than as a set of application specific integrated circuits (ASIC's) such as FPGA's. It is anticipated that in the future device densities will continue to increase, so that the complete system may eventually form a sub-unit (a "core") of a larger single chip unit.

Operation of the Invention

Fig. 15 shows how an embodiment of the network monitor 300 might be used to analyze traffic in a network 102. Packet acquisition device 1502 acquires all the packets from a connection point 121 on network 102 so that all packets passing point 121 in either direction are supplied to monitor 300. Monitor 300 comprises the parser sub-system 301, which determines flow signatures, and analyzer sub-system 303 that analyzes the flow signature of each packet. A memory 324 is used to store the database of flows that are determined and updated by monitor 300. A host computer 1504, which might be any processor, for example, a general-purpose computer, is used to analyze the flows in memory 324. As is conventional, host computer 1504 includes a memory, say RAM, shown as host memory 1506. In addition, the host might contain a disk. In one application, the system can operate as an RMON probe, in which case the host computer is coupled to a network interface card 1510 that is connected to the network 102.

The preferred embodiment of the invention is supported by an optional Simple Network Management Protocol (SNMP) implementation. Fig. 15 describes how one would, for example, implement an RMON probe, where a network interface card is used to send RMON information to the network. Commercial SNMP implementations also are available, and using such an implementation can simplify the process of porting the preferred embodiment of the invention to any platform.

In addition, MIB Compilers are available. An MIB Compiler is a tool that greatly simplifies the creation and maintenance of proprietary MIB extensions.

Examples of Packet Elucidation

Monitor 300, and in particular, analyzer 303 is capable of carrying out state analysis for packet exchanges that are commonly referred to as "server announcement" type exchanges. Server announcement is a process used to ease communications between a server with multiple applications that can all be simultaneously accessed from multiple clients. Many applications use a server announcement process as a means of multiplexing a single port or socket into many applications and services. With this type of exchange, messages are sent on the network, in either a broadcast or multicast approach, to announce a server and application, and all stations in the network may receive and decode these messages. The messages enable the stations to derive the

appropriate connection point for communicating that particular application with the particular server. Using the server announcement method, a particular application communicates using a service channel, in the form of a TCP or UDP socket or port as in the IP protocol suite, or using a SAP as in the Novell IPX protocol suite.

5 The analyzer 303 is also capable of carrying out "in-stream analysis" of packet exchanges. The "in-stream analysis" method is used either as a primary or secondary recognition process. As a primary process, in-stream analysis assists in extracting detailed information which will be used to further recognize both the specific application and application component. A good example of in-stream analysis is any Web-based
10 application. For example, the commonly used PointCast Web information application can be recognized using this process; during the initial connection between a PointCast server and client, specific key tokens exist in the data exchange that will result in a signature being generated to recognize PointCast.

 The in-stream analysis process may also be combined with the server
15 announcement process. In many cases in-stream analysis will augment other recognition processes. An example of combining in-stream analysis with server announcement can be found in business applications such as SAP and BAAN.

 "Session tracking" also is known as one of the primary processes for tracking applications in client/server packet exchanges. The process of tracking sessions requires
20 an initial connection to a predefined socket or port number. This method of communication is used in a variety of transport layer protocols. It is most commonly seen in the TCP and UDP transport protocols of the IP protocol.

 During the session tracking, a client makes a request to a server using a specific port or socket number. This initial request will cause the server to create a TCP or UDP
25 port to exchange the remainder of the data between the client and the server. The server then replies to the request of the client using this newly created port. The original port used by the client to connect to the server will never be used again during this data exchange.

 One example of session tracking is TFTP (Trivial File Transfer Protocol), a
30 version of the TCP/IP FTP protocol that has no directory or password capability. During the client/server exchange process of TFTP, a specific port (port number 69) is always

used to initiate the packet exchange. Thus, when the client begins the process of communicating, a request is made to UDP port 69. Once the server receives this request, a new port number is created on the server. The server then replies to the client using the new port. In this example, it is clear that in order to recognize TFTP; network monitor
5 300 analyzes the initial request from the client and generates a signature for it. Monitor 300 uses that signature to recognize the reply. Monitor 300 also analyzes the reply from the server with the key port information, and uses this to create a signature for monitoring the remaining packets of this data exchange.

Network monitor 300 can also understand the current state of particular
10 connections in the network. Connection-oriented exchanges often benefit from state tracking to correctly identify the application. An example is the common TCP transport protocol that provides a reliable means of sending information between a client and a server. When a data exchange is initiated, a TCP request for synchronization message is sent. This message contains a specific sequence number that is used to track an
15 acknowledgement from the server. Once the server has acknowledged the synchronization request, data may be exchanged between the client and the server. When communication is no longer required, the client sends a finish or complete message to the server, and the server acknowledges this finish request with a reply containing the sequence numbers from the request. The states of such a connection-oriented exchange
20 relate to the various types of connection and maintenance messages.

Server Announcement Example

The individual methods of server announcement protocols vary. However, the basic underlying process remains similar. A typical server announcement message is sent to one or more clients in a network. This type of announcement message has specific
25 content, which, in another aspect of the invention, is salvaged and maintained in the database of flow-entries in the system. Because the announcement is sent to one or more stations, the client involved in a future packet exchange with the server will make an assumption that the information announced is known, and an aspect of the inventive monitor is that it too can make the same assumption.

30 Sun-RPC is the implementation by Sun Microsystems, Inc. (Palo Alto, California) of the Remote Procedure Call (RPC), a programming interface that allows

one program to use the services of another on a remote machine. A Sun-RPC example is now used to explain how monitor 300 can capture server announcements.

A remote program or client that wishes to use a server or procedure must establish a connection, for which the RPC protocol can be used.

5 Each server running the Sun-RPC protocol must maintain a process and database called the port Mapper. The port Mapper creates a direct association between a Sun-RPC program or application and a TCP or UDP socket or port (for TCP or UDP implementations). An application or program number is a 32-bit unique identifier assigned by ICANN (the Internet Corporation for Assigned Names and Numbers,
10 www.icann.org), which manages the huge number of parameters associated with Internet protocols (port numbers, router protocols, multicast addresses, *etc.*) Each port Mapper on a Sun-RPC server can present the mappings between a unique program number and a specific transport socket through the use of specific request or a directed announcement. According to ICANN, port number 111 is associated with Sun RPC.

15 As an example, consider a client (*e.g.*, CLIENT 3 shown as 106 in FIG. 1) making a specific request to the server (*e.g.*, SERVER 2 of FIG. 1, shown as 110) on a predefined UDP or TCP socket. Once the port Mapper process on the sun RPC server receives the request, the specific mapping is returned in a directed reply to the client.

- 20 1. A client (CLIENT 3, 106 in FIG. 1) sends a TCP packet to SERVER 2 (110 in FIG. 1) on port 111, with an RPC Bind Lookup Request (`rpcBindLookup`). TCP or UDP port 111 is always associated Sun RPC. This request specifies the program (as a program identifier), version, and might specify the protocol (UDP or TCP).
- 25 2. The server SERVER 2 (110 in FIG. 1) extracts the program identifier and version identifier from the request. The server also uses the fact that this packet came in using the TCP transport and that no protocol was specified, and thus will use the TCP protocol for its reply.

3. The server 110 sends a TCP packet to port number 111, with an RPC Bind Lookup Reply. The reply contains the specific port number (*e.g.*, port number 'port') on which future transactions will be accepted for the specific RPC program identifier (*e.g.*, Program 'program') and the protocol (UDP or TCP) for use.

5
10 It is desired that from now on every time that port number 'port' is used, the packet is associated with the application program 'program' until the number 'port' no longer is to be associated with the program 'program'. Network monitor 300 by creating a flow-entry and a signature includes a mechanism for remembering the exchange so that future packets that use the port number 'port' will be associated by the network monitor with the application program 'program'.

15 In addition to the Sun RPC Bind Lookup request and reply, there are other ways that a particular program—say 'program'—might be associated with a particular port number, for example number 'port'. One is by a broadcast announcement of a particular association between an application service and a port number, called a Sun RPC portMapper Announcement. Another, is when some server—say the same SERVER 2—replies to some client—say CLIENT 1—requesting some portMapper assignment with a RPC portMapper Reply. Some other client—say CLIENT 2—might inadvertently see this request, and thus know that for this particular server, SERVER 2, port number 'port' is associated with the application service 'program'. It is desirable for the network monitor 300 to be able to associate any packets to SERVER 2 using port number 'port' with the application program 'program'.

25 FIG. 9 represents a dataflow 900 of some operations in the monitor 300 of FIG. 3 for Sun Remote Procedure Call. Suppose a client 106 (*e.g.*, CLIENT 3 in FIG. 1) is communicating via its interface to the network 118 to a server 110 (*e.g.*, SERVER 2 in FIG. 1) via the server's interface to the network 116. Further assume that Remote Procedure Call is used to communicate with the server 110. One path in the data flow 900 starts with a step 910 that a Remote Procedure Call bind lookup request is issued by client 106 and ends with the server state creation step 904. Such RPC bind lookup request includes values for the 'program,' 'version,' and 'protocol' to use, *e.g.*, TCP or
30

UDP. The process for Sun RPC analysis in the network monitor 300 includes the following aspects. :

- Process 909: Extract the 'program,' 'version,' and 'protocol' (UDP or TCP). Extract the TCP or UDP port (process 909) which is 111 indicating Sun RPC.
- 5 • Process 908: Decode the Sun RPC packet. Check RPC type field for ID. If value is portMapper, save paired socket (*i.e.*, dest for destination address, src for source address). Decode ports and mapping, save ports with socket/addr key. There may be more than one pairing per mapper packet. Form a signature (*e.g.*, a key). A flow-entry is created in database 324. The saving of the request is now complete.

10 At some later time, the server (process 907) issues a RPC bind lookup reply. The packet monitor 300 will extract a signature from the packet and recognize it from the previously stored flow. The monitor will get the protocol port number (906) and lookup the request (905). A new signature (*i.e.*, a key) will be created and the creation of the server state (904) will be stored as an entry identified by the new signature in the flow-
 15 entry database. That signature now may be used to identify packets associated with the server.

 The server state creation step 904 can be reached not only from a Bind Lookup Request/Reply pair, but also from a RPC Reply portMapper packet shown as 901 or an RPC Announcement portMapper shown as 902. The Remote Procedure Call protocol
 20 can announce that it is able to provide a particular application service. Embodiments of the present invention preferably can analyze when an exchange occurs between a client and a server, and also can track those stations that have received the announcement of a service in the network.

 The RPC Announcement portMapper announcement 902 is a broadcast. Such
 25 causes various clients to execute a similar set of operations, for example, saving the information obtained from the announcement. The RPC Reply portMapper step 901 could be in reply to a portMapper request, and is also broadcast. It includes all the service parameters.

 Thus monitor 300 creates and saves all such states for later classification of flows
 30 that relate to the particular service 'program'.

FIG. 2 shows how the monitor 300 in the example of Sun RPC builds a signature and flow states. A plurality of packets 206-209 are exchanged, *e.g.*, in an exemplary Sun Microsystems Remote Procedure Call protocol. A method embodiment of the present invention might generate a pair of flow signatures, "signature-1" 210 and "signature-2" 212, from information found in the packets 206 and 207 which, in the example, correspond to a Sun RPC Bind Lookup request and reply, respectively.

Consider first the Sun RPC Bind Lookup request. Suppose packet 206 corresponds to such a request sent from CLIENT 3 to SERVER 2. This packet contains important information that is used in building a signature according to an aspect of the invention. A source and destination network address occupy the first two fields of each packet, and according to the patterns in pattern database 308, the flow signature (shown as KEY1 230 in FIG. 2) will also contain these two fields, so the parser subsystem 301 will include these two fields in signature KEY 1 (230). Note that in FIG. 2, if an address identifies the client 106 (shown also as 202), the label used in the drawing is "C₁". If such address identifies the server 110 (shown also as server 204), the label used in the drawing is "S₁". The first two fields 214 and 215 in packet 206 are "S₁" and C₁" because packet 206 is provided from the server 110 and is destined for the client 106. Suppose for this example, "S₁" is an address numerically less than address "C₁". A third field "p¹" 216 identifies the particular protocol being used, *e.g.*, TCP, UDP, etc.

In packet 206, a fourth field 217 and a fifth field 218 are used to communicate port numbers that are used. The conversation direction determines where the port number field is. The diagonal pattern in field 217 is used to identify a source-port pattern, and the hash pattern in field 218 is used to identify the destination-port pattern. The order indicates the client-server message direction. A sixth field denoted "i¹" 219 is an element that is being requested by the client from the server. A seventh field denoted "s₁a" 220 is the service requested by the client from server 110. The following eighth field "QA" 221 (for question mark) indicates that the client 106 wants to know what to use to access application "s₁a". A tenth field "QP" 223 is used to indicate that the client wants the server to indicate what protocol to use for the particular application.

Packet 206 initiates the sequence of packet exchanges, *e.g.*, a RPC Bind Lookup Request to SERVER 2. It follows a well-defined format, as do all the

packets, and is transmitted to the server 110 on a well-known service connection identifier (port 111 indicating Sun RPC).

Packet 207 is the first sent in reply to the client 106 from the server. It is the RPC Bind Lookup Reply as a result of the request packet 206.

5 Packet 207 includes ten fields 224–233. The destination and source addresses are carried in fields 224 and 225, *e.g.*, indicated “C₁” and “S₁”, respectively. Notice the order is now reversed, since the client-server message direction is from the server 110 to the client 106. The protocol “p¹” is used as indicated in field 226. The request “i¹” is in field 229. Values have been filled in for the application port number, *e.g.*, in field 233 and protocol ““p²”” in field 233.

The flow signature and flow states built up as a result of this exchange are now described. When the packet monitor 300 sees the request packet 206 from the client, a first flow signature 210 is built in the parser subsystem 301 according to the pattern and extraction operations database 308. This signature 210 includes a destination and a
 15 source address 240 and 241. One aspect of the invention is that the flow keys are built consistently in a particular order no matter what the direction of conversation. Several mechanisms may be used to achieve this. In the particular embodiment, the numerically lower address is always placed before the numerically higher address. Such least to highest order is used to get the best spread of signatures and hashes for the lookup
 20 operations. In this case, therefore, since we assume “S₁” < “C₁”, the order is address “S₁” followed by client address “C₁”. The next field used to build the signature is a protocol field 242 extracted from packet 206’s field 216, and thus is the protocol “p¹”. The next field used for the signature is field 243, which contains the destination source port number shown as a crosshatched pattern from the field 218 of the packet 206. This
 25 pattern will be recognized in the payload of packets to derive how this packet or sequence of packets exists as a flow. In practice, these may be TCP port numbers, or a combination of TCP port numbers. In the case of the Sun RPC example, the crosshatch represents a set of port numbers of UDS for p¹ that will be used to recognize this flow (*e.g.*, port 111). Port 111 indicates this is Sun RPC. Some applications, such as the Sun
 30 RPC Bind Lookups, are directly determinable (“known”) at the parser level. So in this case, the signature KEY-1 points to a known application denoted “a¹” (Sun RPC Bind

Lookup), and a next-state that the state processor should proceed to for more complex recognition jobs, denoted as state "st_D" is placed in the field 245 of the flow-entry.

When the Sun RPC Bind Lookup reply is acquired, a flow signature is again built by the parser. This flow signature is identical to KEY-1. Hence, when the signature enters the analyzer subsystem 303 from the parser subsystem 301, the complete flow-entry is obtained, and in this flow-entry indicates state "st_D". The operations for state "st_D" in the state processor instruction database 326 instructs the state processor to build and store a new flow signature, shown as KEY-2 (212) in FIG. 2. This flow signature built by the state processor also includes the destination and a source addresses 250 and 251, respectively, for server "S₁" followed by (the numerically higher address) client "C₁". A protocol field 252 defines the protocol to be used, *e.g.*, "p²" which is obtained from the reply packet. A field 253 contains a recognition pattern also obtained from the reply packet. In this case, the application is Sun RPC, and field 254 indicates this application "a²". A next-state field 255 defines the next state that the state processor should proceed to for more complex recognition jobs, *e.g.*, a state "st¹". In this particular example, this is a final state. Thus, KEY-2 may now be used to recognize packets that are in any way associated with the application "a²". Two such packets 208 and 209 are shown, one in each direction. They use the particular application service requested in the original Bind Lookup Request, and each will be recognized because the signature KEY-2 will be built in each case.

The two flow signatures 210 and 212 always order the destination and source address fields with server "S₁" followed by client "C₁". Such values are automatically filled in when the addresses are first created in a particular flow signature. Preferably, large collections of flow signatures are kept in a lookup table in a least-to-highest order for the best spread of flow signatures and hashes.

Thereafter, the client and server exchange a number of packets, *e.g.*, represented by request packet 208 and response packet 209. The client 106 sends packets 208 that have a destination and source address S₁ and C₁, in a pair of fields 260 and 261. A field 262 defines the protocol as "p²", and a field 263 defines the destination port number.

Some network-server application recognition jobs are so simple that only a single state transition has to occur to be able to pinpoint the application that produced the packet. Others require a sequence of state transitions to occur in order to match a known and predefined climb from state-to-state.

5 Thus the flow signature for the recognition of application "a2" is automatically set up by predefining what packet-exchange sequences occur for this example when a relatively simple Sun Microsystems Remote Procedure Call bind lookup request instruction executes. More complicated exchanges than this may generate more than two flow signatures and their corresponding states. Each recognition may involve setting up a
10 complex state transition diagram to be traversed before a "final" resting state such as "st₁" in field 255 is reached. All these are used to build the final set of flow signatures for recognizing a particular application in the future.

The Cache Subsystem

Referring again to FIG. 11, the cache subsystem 1115 is connected to the lookup
15 update engine (LUE) 1107, the state processor the state processor (SP) 1108 and the flow insertion/deletion engine (FIDE) 1110. The cache 1115 keeps a set of flow-entries of the flow-entry database stored in memory 1123, so is coupled to memory 1123 via the unified memory controller 1119. According to one aspect of the invention, these entries in the cache are those likely-to-be-accessed next.

20 It is desirable to maximize the hit rate in a cache system. Typical prior-art cache systems are used to expedite memory accesses to and from microprocessor systems. Various mechanisms are available in such prior art systems to predict the lookup such that the hit rate can be maximized. Prior art caches, for example, can use a lookahead mechanism to predict both instruction cache lookups and data cache lookups. Such
25 lookahead mechanisms are not available for the packet monitoring application of cache subsystem 1115. When a new packet enters the monitor 300, the next cache access, for example from the LUE 1107, may be for a totally different flow than the last cache lookup, and there is no way ahead of time of knowing what flow the next packet will belong to.

30 One aspect of the present invention is a cache system that replaces a least recently

used (LRU) flow-entry when a cache replacement is needed. Replacing least recently used flow-entries is preferred because it is likely that a packet following a recent packet will belong to the same flow. Thus, the signature of a new packet will likely match a recently used flow record. Conversely, it is not highly likely that a packet associated with
5 the least recently used flow-entry will soon arrive.

Furthermore, after one of the engines that operate on flow-entries, for example the LUE 1107, completes an operation on a flow-entry, it is likely that the same or another engine will soon use the same flow-entry. Thus it is desirable to make sure that recently used entries remain in the cache.

10 A feature of the cache system of the present invention is that most recently used (MRU) flow-entries are kept in cache whenever possible. Since typically packets of the same flow arrive in bursts, and since MRU flow-entries are likely to be required by another engine in the analysis subsystem, maximizing likelihood of MRU flow-entries remaining in cache increases the likelihood of finding flow records in the cache, thus
15 increasing the cache hit rate.

Yet another aspect of the present cache invention is that it includes an associative memory using a set of content addressable memory cells (CAMs). The CAM contains an address that in our implementation is the hash value associated with the corresponding flow-entry in a cache memory (e.g., a data RAM) comprising memory cells. In one
20 embodiment, each memory cell is a page. Each CAM also includes a pointer to a cache memory page. Thus, the CAM contents include the address and the pointer to cache memory. As is conventional, each CAM cell includes a matching circuit having an input. The hash is presented to the CAM's matching circuit input, and if the hash matches the hash in the CAM, the a match output is asserted indicating there is a hit. The CAM
25 pointer points to the page number (i.e., the address) in the cache memory of the flow-entry.

Each CAM also includes a cache address input, a cache pointer input, and a cache contents output for inputting and outputting the address part and pointer part of the CAM.

30 The particular embodiment cache memory stores flow-entries in pages of one bucket, i.e., that can store a single flow-entry. Thus, the pointer is the page number in the

cache memory. In one version, each hash value corresponds to a bin of N flow-entries (e.g., 4 buckets in the preferred embodiment of this version). In another implementation, each hash value points to a single flow record, i.e., the bin and bucket sizes correspond. For simplicity, this second implementation is assumed when describing the cache 1115.

5 Furthermore, as is conventional, the match output signal is provided to a corresponding location in the cache memory so that a read or write operation may take place with the location in the cache memory pointed to be the CAM.

One aspect of the present invention achieves a combination of associatively and true LRU replacement policy. For this, the CAMs of cache system 1115 are organized in
10 what we call a CAM stack (also CAM array) in an ordering, with a top CAM and a bottom CAM. The address and pointer output of each CAM starting from the top CAM is connected to the address and pointer input of the next cache up to the bottom.

In our implementation, a hash is used to address the cache. The hash is input to the CAM array, and any CAM that has an address that matches the input hash asserts its
15 match output indicating a hit. When there is a cache hit, the contents of the CAM that produced the hit (including the address and pointer to cache memory) are put in the top CAM of the stack. The CAM contents (cache address, and cache memory pointer) of the CAMs above the CAM that produced are shifted down to fill the gap.

If there is a miss, any new flow record is put in the cache memory element
20 pointed to by the bottom CAM. All CAM contents above the bottom are shifted down one, and then the new hash value and the pointer to cache memory of the new flow-entry are put in the top-most CAM of the CAM stack.

In this manner, the CAMs are ordered according to recentness of use, with the least recently used cache contents pointed to by the bottom CAM and the most recently
25 used cache contents pointed to by the top CAM.

Furthermore, unlike a conventional CAM-based cache, there is no fixed relationship between the address in the CAM and what element of cache memory it points to. CAM's relationship to a page of cache memory changes over time. For example, at one instant, the fifth CAM in the stack can include a pointer to one particular
30 page of cache memory, and some time later, that same fifth CAM can point to a different

cache memory page.

In one embodiment, the CAM array includes 32 CAMs and the cache memory includes 32 memory cells (e.g., memory pages), one page pointed to by each CAM contents. Suppose the CAMs are numbered $CAM_0, CAM_1, \dots, CAM_{31}$, respectively, with CAM_0 the top CAM in the array and CAM_{31} the bottom CAM.

The CAM array is controlled by a CAM controller implemented as a state machine, and the cache memory is controlled by a cache memory controller which also is implemented as a state machine. The need for such controllers and how to implement them as state machines or otherwise would be clear to one skilled in the art from this description of operation. In order not to confuse these controllers with other controllers, for example, with the unified memory controller, the two controllers will be called the CAM state machine and the memory state machine, respectively.

Consider as an example, that the state of the cache is that it is full. Suppose furthermore that the contents of the CAM stack (the address and the pointer to the cache memory) and of the cache memory at each page number address of cache memory are as shown in the following table.

CAM	Hash	Cache Point		Cache Addr.	Contents
CAM_0	hash ₀	page ₀		page ₀	entry ₀
CAM_1	hash ₁	page ₁		page ₁	entry ₁
CAM_2	hash ₂	page ₂		page ₂	entry ₂
CAM_3	hash ₃	page ₃		page ₃	entry ₃
CAM_4	hash ₄	page ₄		page ₄	entry ₄
CAM_5	hash ₅	page ₅		page ₅	entry ₅
CAM_6	hash ₆	page ₆		page ₆	entry ₆
CAM_7	hash ₇	page ₇		page ₇	entry ₇
...
CAM_{29}	hash ₂₉	page ₂₉		page ₂₉	entry ₂₉
CAM_{30}	hash ₃₀	page ₃₀		page ₃₀	entry ₃₀
CAM_{31}	hash ₃₁	page ₃₁		page ₃₁	entry ₃₁

This says that CAM_4 contains and will match with the hash value hash₄, and a lookup with hash₄ will produce a match and the address page₄ in cache memory. Furthermore,

page₄ in cache memory contains the flow-entry, entry₄, that in this notation is the flow-entry matching hash value hash₄. This table also indicates that hash₀ was more recently used than hash₁, hash₅ more recently than hash₂, and so forth, with hash₃₁ the least recently used hash value. Suppose further that the LUE 1107 obtains an entry from unified flow key buffer 1103 with a hash value hash₃₁. The LUE looks up the cache subsystem via the CAM array. CAM₃₁ gets a hit and returns the page number of the hit, i.e., page₃₁. The cache subsystem now indicates to the LUE 1007 that the supplied hash value produced a hit and provides a pointer to page₃₁ of the cache memory which contains the flow-entry corresponding to hash₃₁, i.e., flow₃₁. The LUE now retrieve the flow-entry flow₃₁ from the cache memory at address page₃₁. In the preferred embodiment, the lookup of the cache takes only one clock cycle.

The value hash₃₁ is the most recently used hash value. Therefore, in accordance with an aspect of the inventive cache system, the most recently used entry is put on top of the CAM stack. Thus hash₃₁ is put into CAM₀ (pointing to page₃₁). Furthermore, hash₃₀ is now the LRU hash value, so is moved to CAM₃₁. The next least recently used hash value, hash₂₉ is now moved to CAM₃₀, and so forth. Thus, all CAM contents are shifted one down after the MSU entry is put in the top CAM. In the preferred embodiment the shifting down on CAM entries takes one clock cycle. Thus, the lookup and the rearranging of the CAM array to maintain the ordering according to usage recentness. The following table shows the new contents of the CAM array and the (unchanged) contents of the cache memory.

CAM	Hash	Cache Point		Cache Addr.	Contents
CAM ₀	hash ₃₁	page ₃₁		page ₀	entry ₀
CAM ₁	hash ₀	page ₀		page ₁	entry ₁
CAM ₂	hash ₁	page ₁		page ₂	entry ₂
CAM ₃	hash ₂	page ₂		page ₃	entry ₃
CAM ₄	hash ₃	page ₃		page ₄	entry ₄
CAM ₅	hash ₄	page ₄		page ₅	entry ₅
CAM ₆	hash ₅	page ₅		page ₆	entry ₆
CAM ₇	hash ₆	page ₆		page ₇	entry ₇
...
CAM ₂₉	hash ₂₈	page ₂₈		page ₂₉	entry ₂₉
CAM ₃₀	hash ₂₉	page ₂₉		page ₃₀	entry ₃₀
CAM ₃₁	hash ₃₀	page ₃₀		page ₃₁	entry ₃₁

To continue with the example, suppose that some time later, the LUE 1007 looks up hash value hash₅. This produces a hit in CAM₆ pointing to page₅ of the cache memory. Thus, in one clock cycle, the cache subsystem 1115 provides LUE 1007 with an indication of a hit and the pointer to the flow-entry in the cache memory. The most recent entry is hash₅, so hash₅ and cache memory address page₆ are entered into CAM₀. The contents of the remaining CAMs are all shifted down one up to and including the entry that contained hash₅. That is, CAM₇, CAM₈, ..., CAM₃₁ remain unchanged. The CAM array contents and unchanged cache memory contents are now as shown in the following table.

CAM	Hash	Cache Point		Cache Addr.	Contents
CAM ₀	hash ₅	page ₅		page ₀	entry ₀
CAM ₁	hash ₃₁	page ₃₁		page ₁	entry ₁
CAM ₂	hash ₀	page ₀		page ₂	entry ₂
CAM ₃	hash ₁	page ₁		page ₃	entry ₃
CAM ₄	hash ₂	page ₂		page ₄	entry ₄
CAM ₅	hash ₃	page ₃		page ₅	entry ₅
CAM ₆	hash ₄	page ₄		page ₆	entry ₆
CAM ₇	hash ₆	page ₆		page ₇	entry ₇
...
CAM ₂₉	hash ₂₈	page ₂₈		page ₂₉	entry ₂₉
CAM ₃₀	hash ₂₉	page ₂₉		page ₃₀	entry ₃₀
CAM ₃₁	hash ₃₀	page ₃₀		page ₃₁	entry ₃₁

Thus in the case of cache hits, the CAM array always keeps used hash values in the order of recentness of use, with the most recently used hash value in the top CAM.

The operation of the cache subsystem when there is a cache hit will be described by continuing the example. Suppose there is a lookup (e.g., from LUE 1107) for hash value hash₄₃. The CAM array produces a miss that causes in a lookup using the hash in the external memory. The specific operation of our specific implementation is that the CAM state machine sends a GET message to the memory state machine that results in a memory lookup using the hash via the unified memory controller (UMC) 1119. However, other means of achieving a memory lookup when there is a miss in the CAM array would be clear to those in the art.

The lookup in the flow-entry database 324 (i.e., external memory) results in a hit or a miss. Suppose that the database 324 of flow-entries does not have an entry matching hash value hash₄₃. The memory state machine indicates the miss to the CAM state machine which then indicates the miss to the LUE 1007. Suppose, on the other hand that there is a flow-entry—entry₄₃— in database 324 matching hash value hash₄₃. In this case, the flow-entry is brought in to be loaded into the cache.

In accordance with another aspect of the invention, the bottom CAM entry CAM₃₁ always points to the LRU address in the cache memory. Thus, implementing a true LRU replacement policy includes flushing out the LRU cache memory entry and

inserting a new entry into that LRU cache memory location pointed to by the bottom CAM. The CAM entry also is modified to reflect the new hash value of the entry in the pointed to cache memory element. Thus, hash value hash₄₃ is put in CAM₃₁ and flow-entry entry₄₃ is placed in the cache page pointed to by CAM 31. The CAM array and

5 now changed cache memory contents are now

CAM	Hash	Cache Point		Cache Addr.	Contents
CAM ₀	hash ₅	page ₅		page ₀	entry ₀
CAM ₁	hash ₃₁	page ₃₁		page ₁	entry ₁
CAM ₂	hash ₀	page ₀		page ₂	entry ₂
CAM ₃	hash ₁	page ₁		page ₃	entry ₃
CAM ₄	hash ₂	page ₂		page ₄	entry ₄
CAM ₅	hash ₃	page ₃		page ₅	entry ₅
CAM ₆	hash ₄	page ₄		page ₆	entry ₆
CAM ₇	hash ₆	page ₆		page ₇	entry ₇
...
CAM ₂₉	hash ₂₈	page ₂₈		page ₂₉	entry ₂₉
CAM ₃₀	hash ₂₉	page ₂₉		page ₃₀	entry ₄₃
CAM ₃₁	hash ₄₃	page ₃₀		page ₃₁	entry ₃₁

Note that the inserted entry is now the MRU flow-entry. So, the contents of CAM₃₁ are now moved to CAM₀ and the entries previously in the top 30 CAMs moved down so that once again, the bottom CAM points to the LRU cache memory page.

CAM	Hash	Cache Point		Cache Addr.	Contents
CAM ₀	hash ₄₃	page ₃₀		page ₀	entry ₀
CAM ₁	hash ₅	page ₅		page ₁	entry ₁
CAM ₂	hash ₃₁	page ₃₁		page ₂	entry ₂
CAM ₃	hash ₀	page ₀		page ₃	entry ₃
CAM ₄	hash ₁	page ₁		page ₄	entry ₄
CAM ₅	hash ₂	page ₂		page ₅	entry ₅
CAM ₆	hash ₃	page ₃		page ₆	entry ₆
CAM ₇	hash ₄	page ₄		page ₇	entry ₇
	hash ₆	page ₆			
...			
			
CAM ₂₉				page ₂₉	entry ₂₉
CAM ₃₀	hash ₂₈	page ₂₈		page ₃₀	entry ₄₃
CAM ₃₁	hash ₂₉	page ₂₉		page ₃₁	entry ₃₁

Note that the inserted entry is now the MRU flow-entry. So, the contents of CAM₃₁ are now moved to CAM₀ and the entries previously in the top 30 CAMs moved

In addition to looking up entries of database 324 via the cache subsystem 1115 for retrieval of an existing flow-entry, the LUE, SP, or FIDE engines also may update the flow-entries via the cache. As such, there may be entries in the cache that are updated flow-entries. Until such updated entries have been written into the flow-entry database 324 in external memory, the flow-entries are called "dirty." As is common in cache systems, a mechanism is provided to indicate dirty entries in the cache. A dirty entry cannot, for example, be flushed out until the corresponding entry in the database 324 has been updated.

Suppose in the last example, that the entry in the cache was modified by the operation. That is, hash₄₃ is in MRU CAM₀, CAM₀ correctly points to page₃₀, but the information in page₃₀ of the cache, entry₄₃, does not correspond to entry₄₃ in database 324. That is, the contents of cache page page₃₀ is dirty. There is now a need to update the database 324. This is called backing up or cleaning the dirty entry.

As is common in cache systems, there is an indication provided that a cache memory entry is dirty using a dirty flag. In the preferred embodiment, there is a dirty flag for each word in cache memory.

Another aspect of the inventive cache system is cleaning cache memory contents according to the entry most likely to be first flushed out of the cache memory. In our LRU cache embodiment, the cleaning of the cache memory entries proceeds in the inverse order of recentness of use. Thus, LRU pages are cleaned first consistent with the
5 least likelihood that these are the entries likely to be flushed first.

In our embodiment, the memory state machine, whenever it is idle, is programmed to scan the CAM array in reverse order of recentness, i.e., starting from the bottom of the CAM array, and look for dirty flags. Whenever a dirty flag is found, the cache memory contents are backed up to the database 324 in external memory.

10 Note that once a page of cache memory is cleaned, it is kept in the cache in case it is still needed. The page is only flushed when more cache memory pages are needed. The corresponding CAM also is not changed until a new cache memory page is needed. In this way, efficient lookups of all cache memory contents, including clean entries are still possible. Furthermore, whenever a cache memory entry is flushed, a check is first
15 made to ensure the entry is clean. If the entry is dirty, it is backed up prior to flushing the entry.

The cache subsystem 1115 can service two read transfers at one time. If there are more than two read requests active at one time the Cache services them in a particular order as follows:

- 20 (1) LRU dirty write back. The cache writes back the least recently used cache memory entry if it is dirty so that there will always be a space for the fetching of cache misses.
- (2) Lookup and update engine 1107.
- (3) State processor 1108.
- 25 (4) Flow insertion and deletion engine 1110.
- (5) Analyzer host interface and control 1118.
- (6) Dirty write back from LRU -1 to MRU; when there is nothing else pending, the cache engine writes dirty entries back to external memory.

FIG. 19 shows the cache memory component 1900 of the cache subsystem 1115.

Cache memory subsystem 1900 includes a bank 1903 of dual ported memories for the pages of cache memory. In our preferred embodiment there are 32 pages. Each page of memory is dual ported. That is, it includes two sets of input ports each having address and data inputs, and two sets of output ports, one set of input and output ports are
5 coupled to the unified memory controller (UMC) 1119 for writing to and reading from the cache memory from and into the external memory used for the flow-entry database 324. Which of the output lines 1909 is coupled to UMC 1119 is selected by a multiplexor 1911 using a cache page select signal 1913 from CAM memory subsystem part of cache system 1115. Updating cache memory from the database 324 uses a cache
10 data signal 1917 from the UMC and a cache address signal 1915.

Looking up and updating data from and to the cache memory from the lookup/update engine (LUE) 1107, state processor (SP) 1108 or flow insertion/deletion engine (FIDE) 1110 uses the other input and output ports of the cache memory pages 1903. A bank of input selection multiplexors 1905 and a set of output selector
15 multiplexors 1907 respectively select the input and output engine using a set of selection signals 1919.

FIG. 20 shows the cache CAM state machine 2001 coupled to the CAM array 2005 and to the memory state machine 2003, together with some of the signals that pass between these elements. The signal names are self-explanatory, and how to implement
20 these controllers as state machines or otherwise would be clear from the description herein above.

While the above description of operation of the CAM array is sufficient for one skilled in the art to design such a CAM array, and many such designs are possible, FIG. 21 shows one such design. Referring to that figure, the CAM array 2005 comprises one
25 CAM, e.g., CAM[7] (2107), per page of CAM memory. The lookup port or update port depend which of the LUE, SP or FIDE are accessing the cache subsystem. The input data for a lookup is typically the hash, and shown as REF-DATA 2103. Loading, updating or evicting the cache is achieved using the signal 2105 that both selects the CAM input data using a select multiplexor 2109, such data being the hit page or the LRU page (the
30 bottom CAM in according to an aspect of the invention). Any loading is done via a 5 to 32 decoder 2111. The results of the CAM lookup for all the CAMs in the array is

provided to a 32-5 low to high 32 to 5 encoder 2113 that outputs the hit 2115, and which
CAM number 2117 produced the hit. The CAM hit page 2119 is an output of a MUX
2121 that has the CAM data of each CAM as input and an output selected by the signal
2117 of the CAM that produced the hit. Maintenance of dirty entries is carried out
5 similarly from the update port that coupled to the CAM state machine 2001. A MUX
2123 has all CAMs' data input and a scan input 2127. The MUX 2123 produces the dirty
data 2125.

Although the present invention has been described in terms of the presently
preferred embodiments, it is to be understood that the disclosure is not to be interpreted
10 as limiting. Various alterations and modifications will no doubt become apparent to
those of ordinary skill in the art after having read the above disclosure. Accordingly, it is
intended that the claims be interpreted as covering all alterations and modifications as
fall within the true spirit and scope of the present invention.

CLAIMS

What is claimed is:

claim A1

1. A packet monitor for examining packets passing through a connection point on a computer network, each packets conforming to one or more protocols, the monitor comprising:
 - (a) a packet acquisition device coupled to the connection point and configured to receive packets passing through the connection point;
 - (b) a memory for storing a database comprising none or more flow-entries for previously encountered conversational flows to which a received packet may belong;
 - (c) a cache subsystem coupled to the flow-entry database memory providing for fast access of flow-entries from the flow-entry database; and
 - (d) a lookup engine coupled to the packet acquisition device and to the cache subsystem and configured to lookup whether a received packet belongs to a flow-entry in the flow-entry database, the looking up being in the cache subsystem.
2. A packet monitor according to claim 1, further comprising:

a parser subsystem coupled to the packet acquisition device and to the lookup engine such that the acquisition device is coupled to the lookup engine via the parser subsystem, the parser subsystem configured to extract identifying information from a received packet,

wherein each flow-entry is identified by identifying information stored in the flow-entry, and wherein the cache lookup uses a function of the extracted identifying information.
3. A packet monitor according to claim 2, wherein the cache subsystem is an associative cache subsystem including one or more content addressable memory cells (CAMs).
4. A packet monitor according to claim 2, wherein the cache subsystem includes:

- (i) a set of cache memory elements coupled to the flow-entry database memory, each cache memory element including an input port to input an flow-entry and configured to store a flow-entry of the flow-entry database;
- (ii) a set of content addressable memory/cells (CAMs) connected according to an order of connections from a top CAM to a bottom CAM, each CAM containing an address and a pointer to one of the cache memory elements, and including:
- a matching circuit having an input such that the CAM asserts a match output when the input is the same as the address in the CAM cell, an asserted match output indicating a hit,
 - a CAM input configured to accept an address and a pointer, and
 - a CAM address output and a CAM pointer output;
- (iii) a CAM controller coupled to the CAM set; and
- (iv) a memory controller coupled to the CAM controller, to the cache memory set, and to the flow-entry memory,

wherein the matching circuit inputs of the CAM cells are coupled to the lookup engine such that that an input to the matching circuit inputs produces a match output in any CAM cell that contains an address equal to the input, and

wherein the CAM controller is configured such that which cache memory element a particular CAM points to changes over time.

5. A packet monitor according to claim 4, wherein the CAM controller is configured such that the bottom CAM points to the least recently used cache memory element.

6. A packet monitor according to claim 5, wherein the address and pointer output of each CAM starting from the top CAM is coupled to the address and pointer input of the next CAM, the final next CAM being the bottom CAM, and wherein the CAM controller is configured such that when there is a cache hit, the address and pointer contents of the CAM that produced the hit are put in the top CAM of the stack, the address and pointer contents of the CAMs above the CAM that produced the asserted match output are shifted down, such that the CAMs are ordered according to recentness of use, with the least recently used cache memory element pointed to by the bottom CAM and the most recently used cache memory element pointed to by the top CAM.
7. A cache system for looking up one or more elements of an external memory, comprising:
- (a) a set of cache memory elements coupled to the external memory, each cache memory element including an input port to input an element of the external memory and configured to store the input external memory element;
 - (b) a set of content addressable memory cells (CAMs) connected according to an order of connections from a top CAM to a bottom CAM, each CAM containing an address and a pointer to one of the cache memory elements, and including
 - (i) a matching circuit having an input such that the CAM asserts a match output when the input is the same as the address in the CAM cell, an asserted match output indicating a hit,
 - (ii) a CAM input configured to accept an address and a pointer, and
 - (iii) a CAM address output and a CAM pointer output, and
 - (c) a CAM controller coupled to the CAM set;
 - (d) a memory controller coupled to the CAM controller, to the cache memory set, and to the external memory,

wherein the matching circuit inputs of the CAM cells are coupled such that that an input to the matching circuit inputs produces a match output in any CAM cell that contains an address equal to the input, and

wherein the CAM controller is configured such that which cache memory element a particular CAM points to changes over time.

- 5
8. A cache system according to claim 7, wherein the CAM controller is configured such that the bottom CAM points to the least recently used cache memory element, and wherein the CAM controller is configured to implement a least recently used replacement policy such that least recently used cache memory element is the first
- 10
9. A cache system according to claim 8, wherein the address and pointer output of each CAM starting from the top CAM is coupled to the address and pointer input of the next CAM, the final next CAM being the bottom CAM, and wherein the CAM controller is configured such than when there is a cache hit, the address and pointer contents of the CAM that produced the hit are put in the top CAM of the stack, the
- 15
- address and pointer contents of the CAMs above the CAM that produced the asserted match output are shifted down, such that the CAMs are ordered according to recentness of use, with the least recently used cache memory element pointed to by the bottom CAM and the most recently used cache memory element pointed to
- 20
- by the top CAM.
10. A cache system according to claim 9, wherein the CAM controller is configured such that replacing any cache memory elements occurs according to the inverse order of recentness of use, with the least recently used entry being the first flushed cache memory entry.
- 25
11. A cache system according to claim 7, wherein each memory element is a page of memory.
12. A cache system according to claim 7, wherein each cache memory element is provided with an indication of whether or not it is dirty, and wherein the CAM controller is configured to clean any dirty cache memory elements by backing up the
- 30
- dirty contents into the external memory.

13. A cache system according to claim 12, wherein the contents of any cache memory element are maintained after cleaning until such cache contents need to be replaced according to the LRU replacement policy.
14. A cache system according to claim 8, wherein each cache memory element is provided with an indication of whether or not it is dirty, and wherein the CAM controller is configured to clean any dirty cache memory elements by backing up the dirty contents into the external memory.
15. A cache system according to claim 14, wherein the CAM controller is further configured to clean any dirty cache memory elements prior to replacing the cache memory element contents.
16. A cache system according to claim 15, wherein the CAM controller is further configured to clean any dirty cache memory elements prior to replacing the cache memory element contents.
17. A cache system according to claim 9, wherein each cache memory element is provided with an indication of whether or not it is dirty, and wherein the CAM controller is configured to clean dirty cache memory elements by backing up the dirty contents into the external memory in reverse order of recentness of use.
18. A cache system according to claim 17, wherein said cleaning in reverse order of recentness of use automatically proceeds whenever the cache controller is idle.
19. A cache system for looking up one or more elements of an external memory, comprising:
- (a) a set of cache memory elements coupled to the external memory, each cache memory element including an input port to input an element of the external memory and configured to store the input external memory element;
 - and
 - (b) a set of content addressable memory cells (CAMs) containing an address and a pointer to one of the cache memory elements, and including a matching circuit having an input such that the CAM asserts a match output when the input is the same as the address in the CAM cell,

wherein which cache memory element a particular CAM points to changes over time.

20. A cache system according to claim 19, wherein the CAMs are connected in an order from top to bottom, and wherein the bottom CAM points to the least recently used cache memory element.

Add A' 

ABSTRACT

A cache system for looking up one or more elements of an external memory, ^{includes} ~~comprising~~ a set of cache memory elements coupled to the external memory, a set of content addressable memory cells (CAMs) containing an address and a pointer to one of the cache memory elements, and ~~including~~ a matching circuit having an input such that the CAM asserts a match output when the input is the same as the address in the CAM cell. ^{The} ~~Which~~ cache memory element ^{which} a particular CAM points to changes over time. In the preferred implementation, the CAMs are connected in an order from top to bottom, and the bottom CAM points to the least recently used cache memory element.

10

370/352
2662
E/ALLAM A

1/21

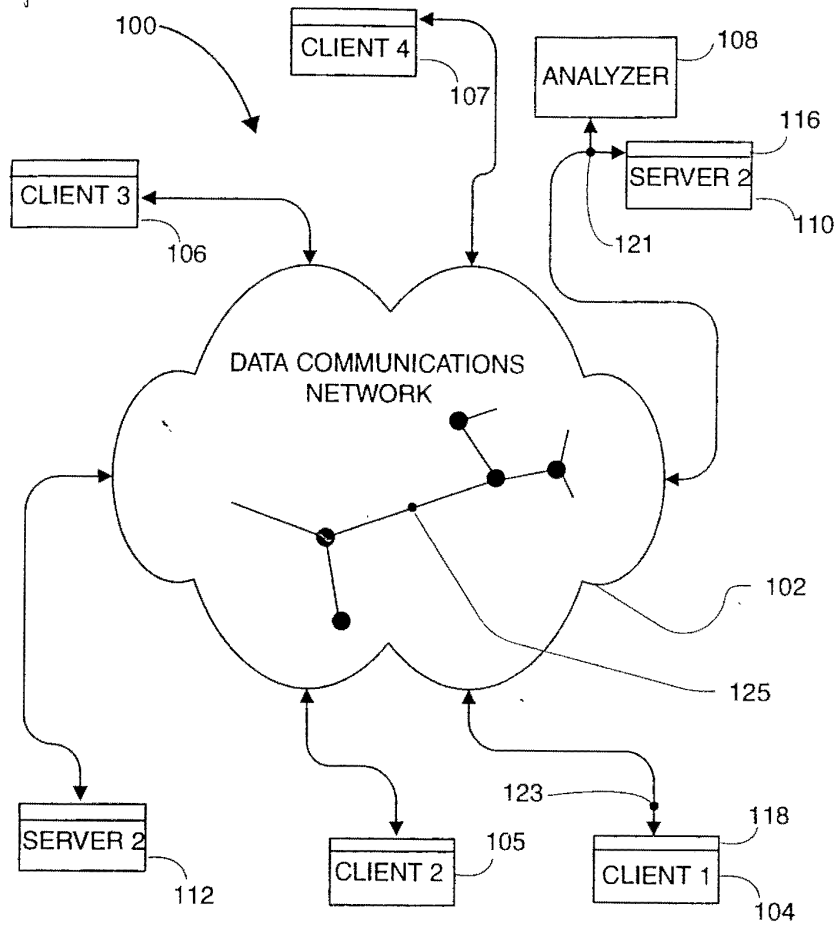


FIG. 1

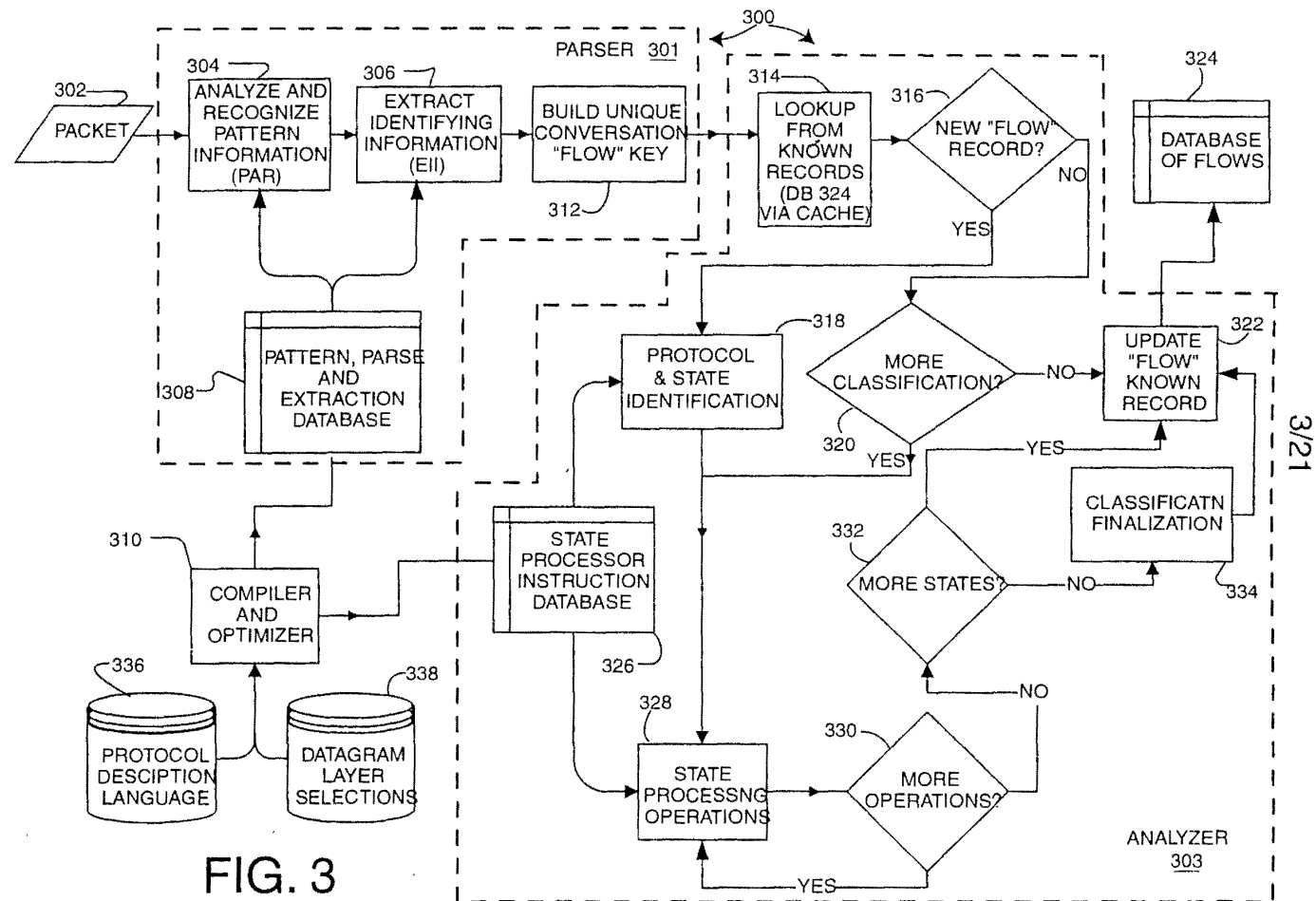


FIG. 3

PRINT OF DRAWINGS
AS ORIGINALLY 3D

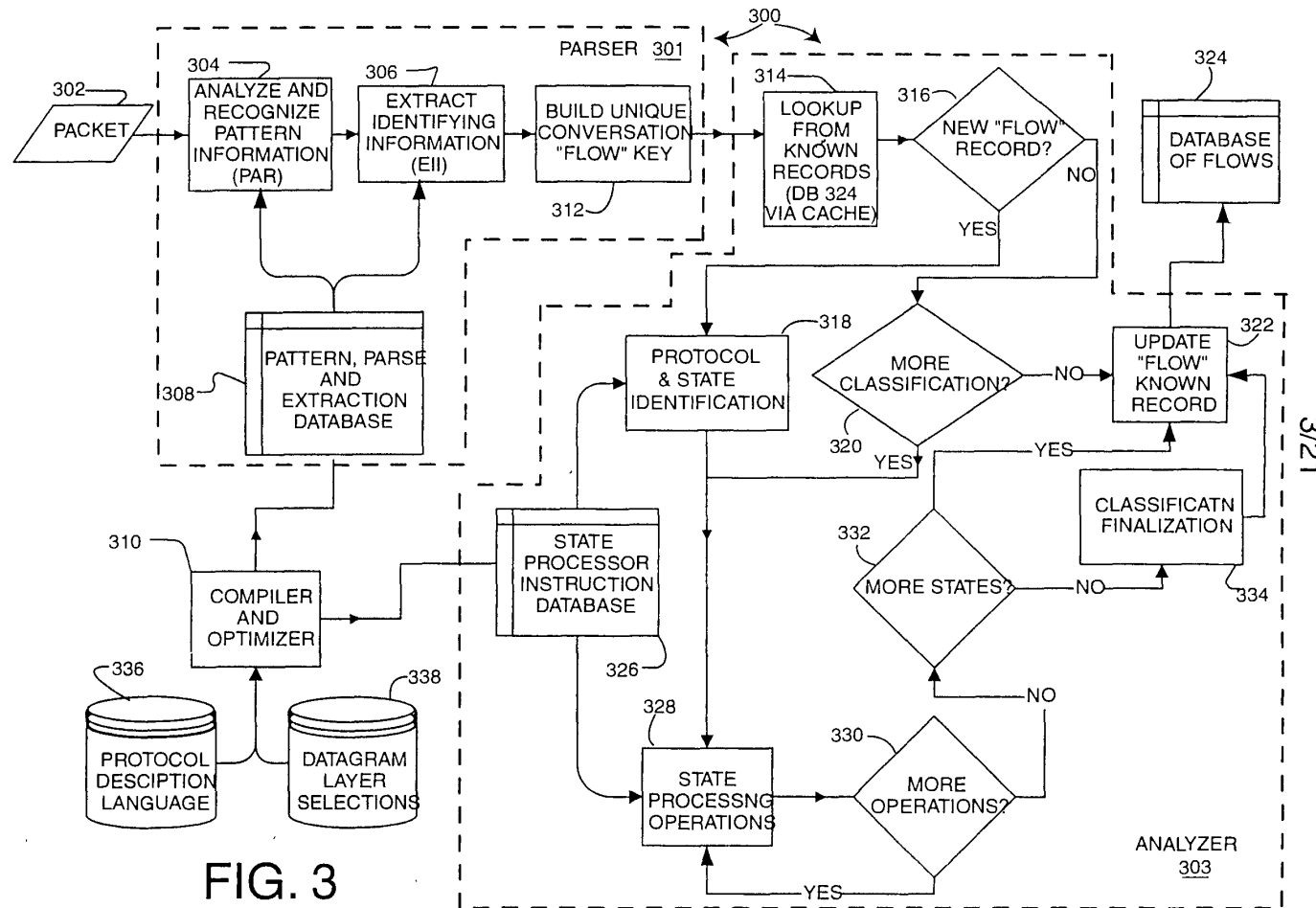


FIG. 3

PRINT OF DRAWINGS
AS ORIGINALLY FILED

4/21

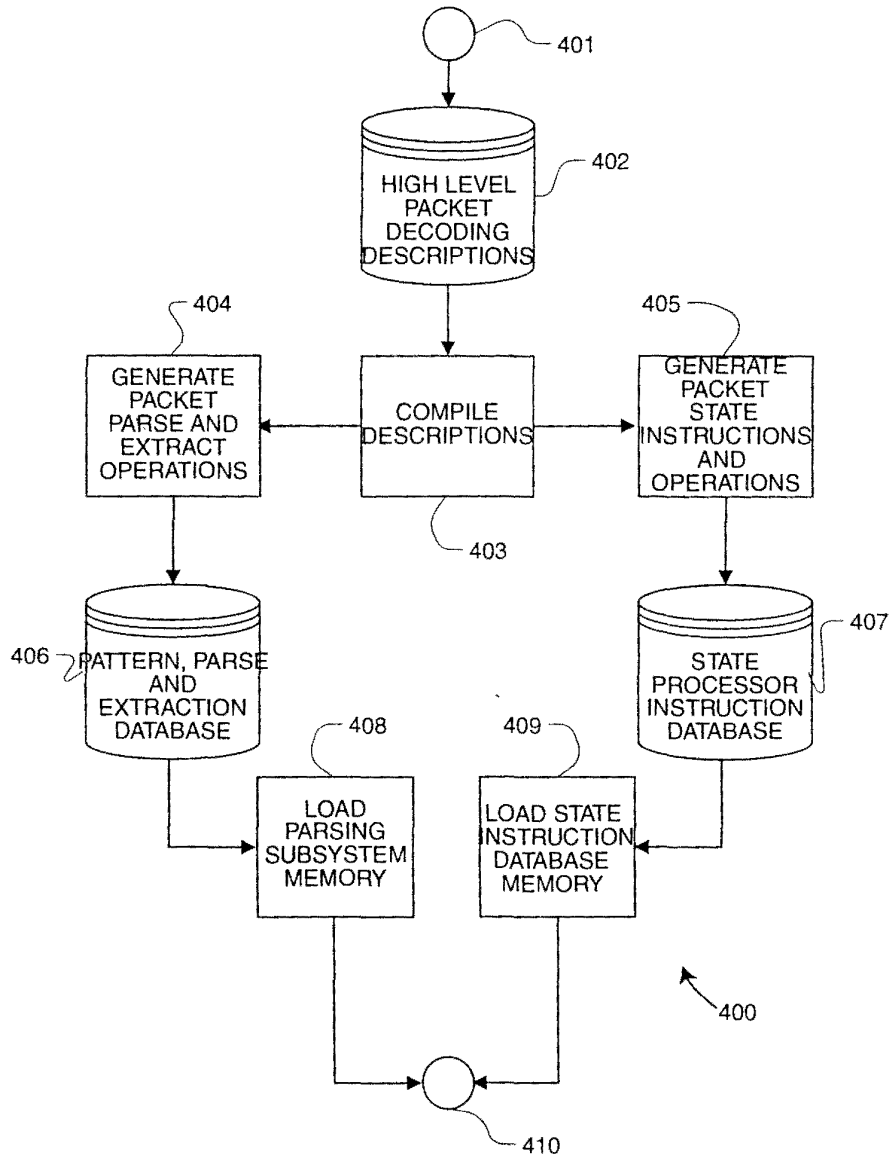


FIG. 4

5/21

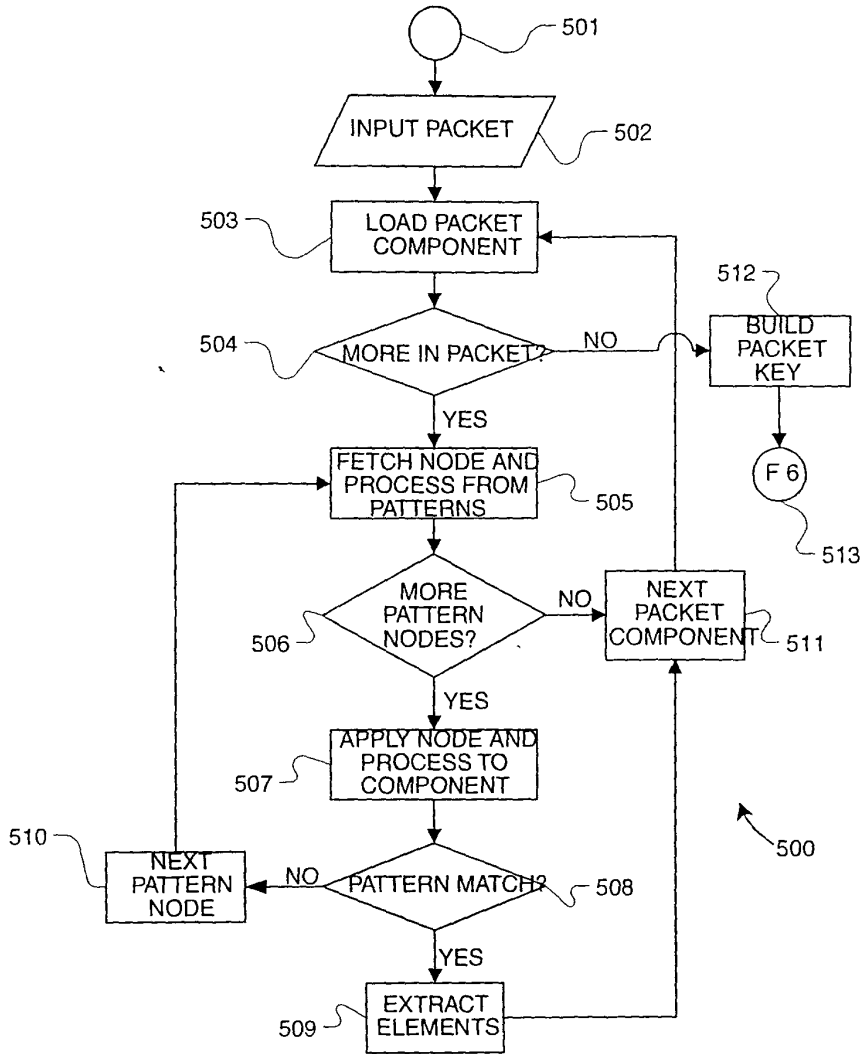


FIG. 5

6/21

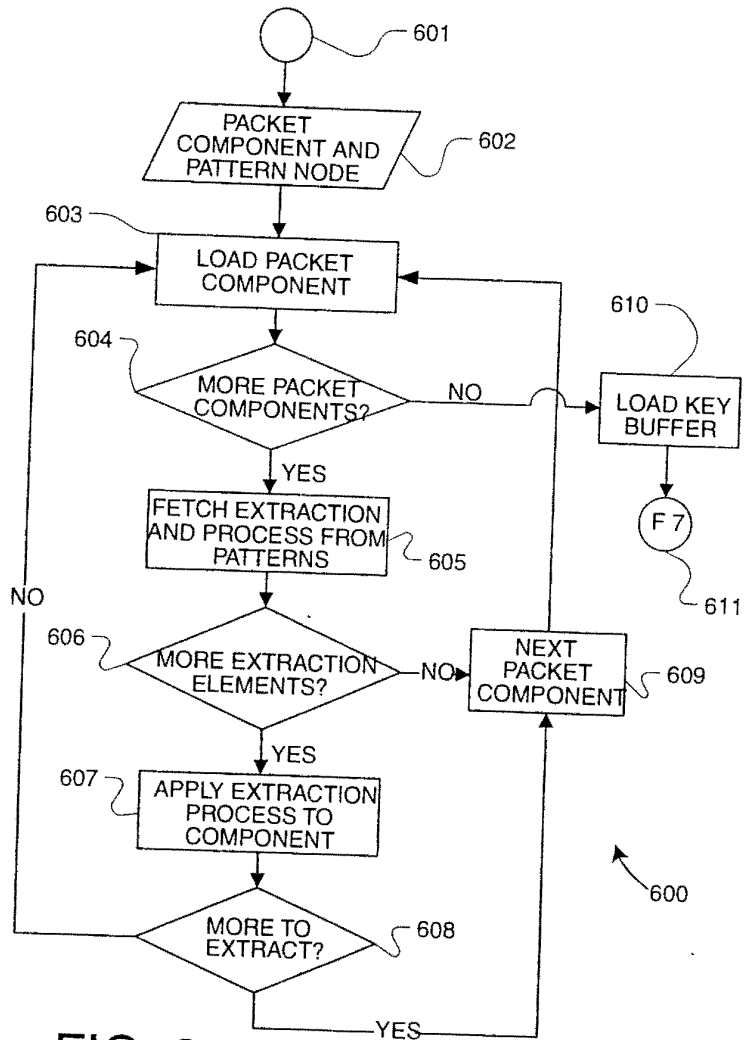


FIG. 6

7/21

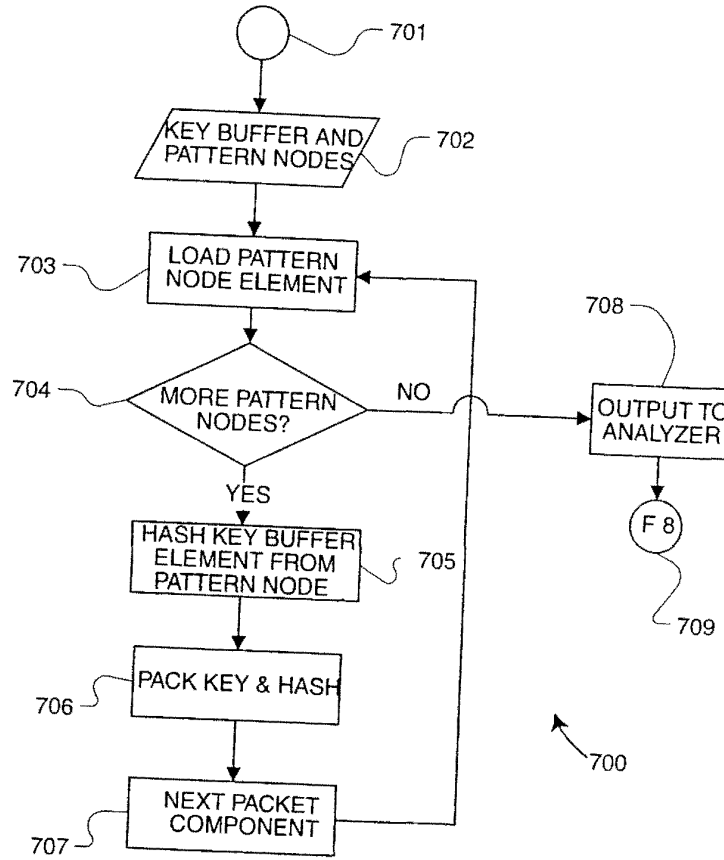
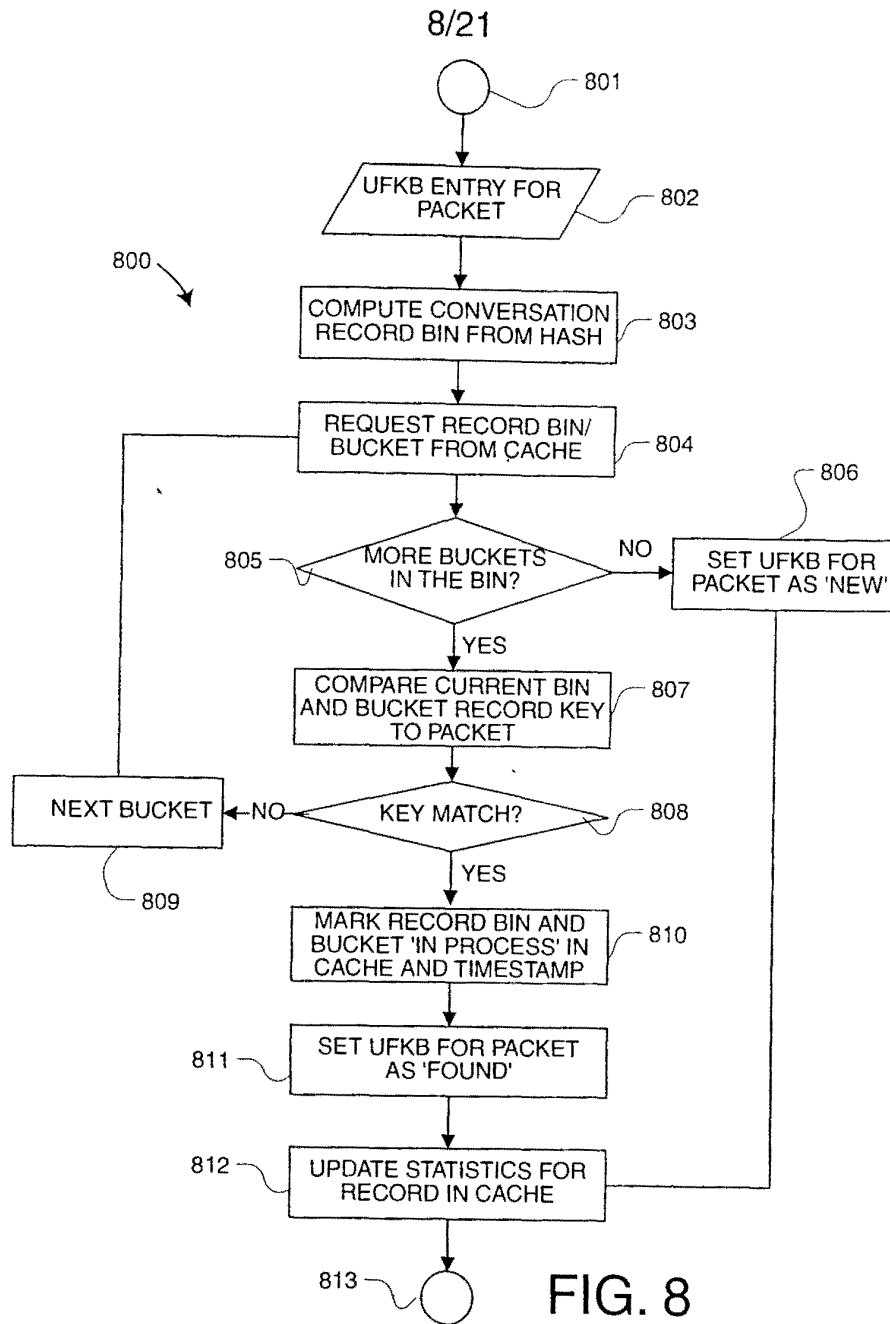


FIG. 7



9/21

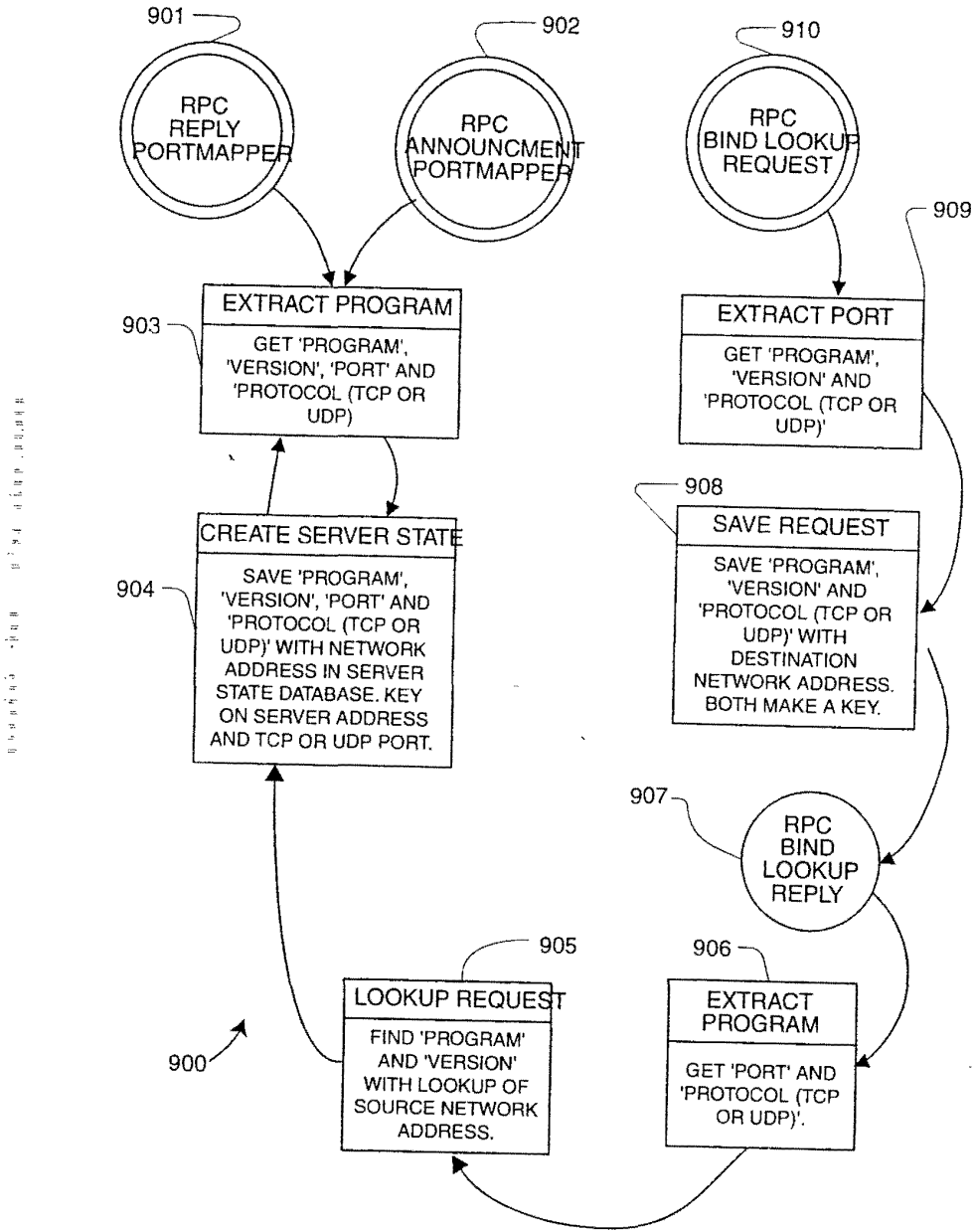


FIG. 9

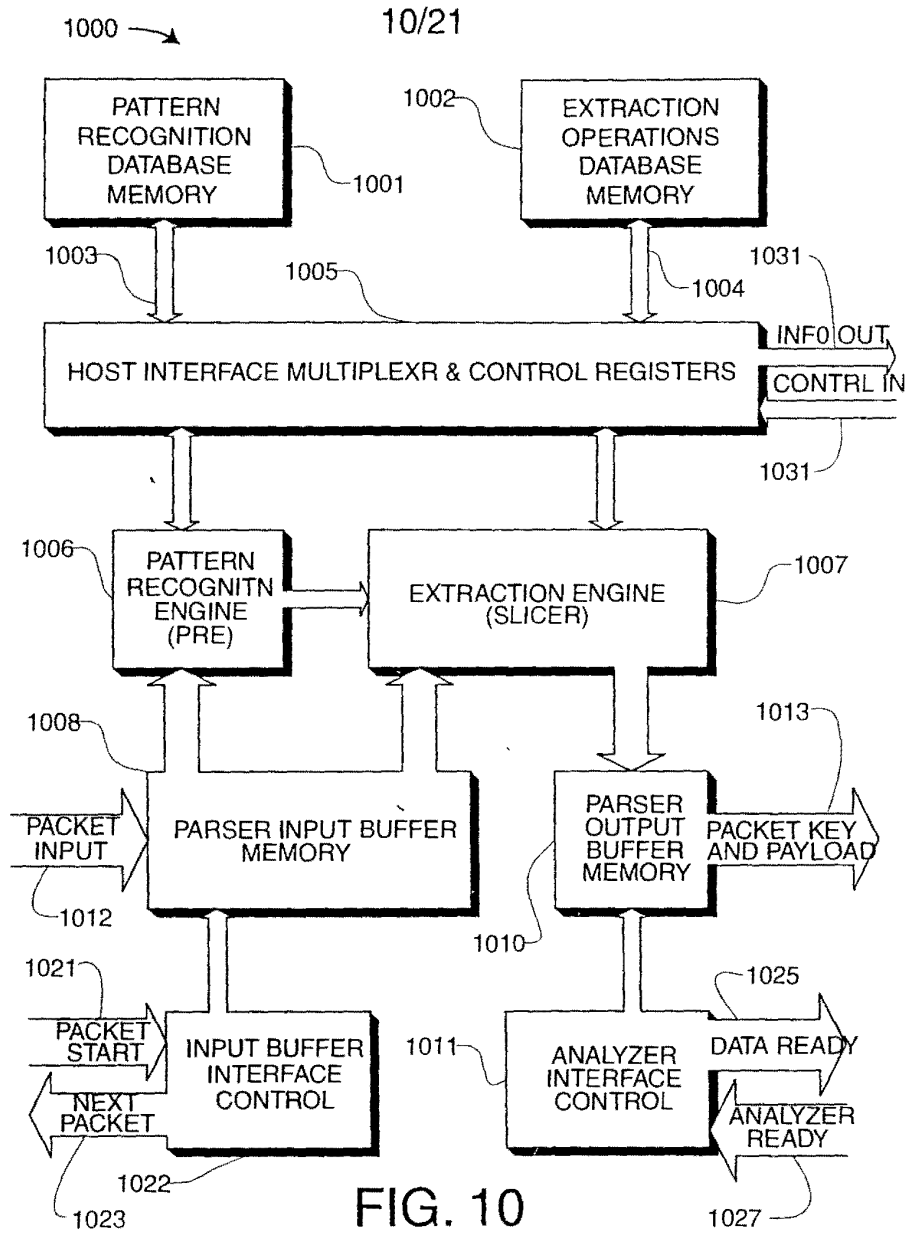


FIG. 10

11/21

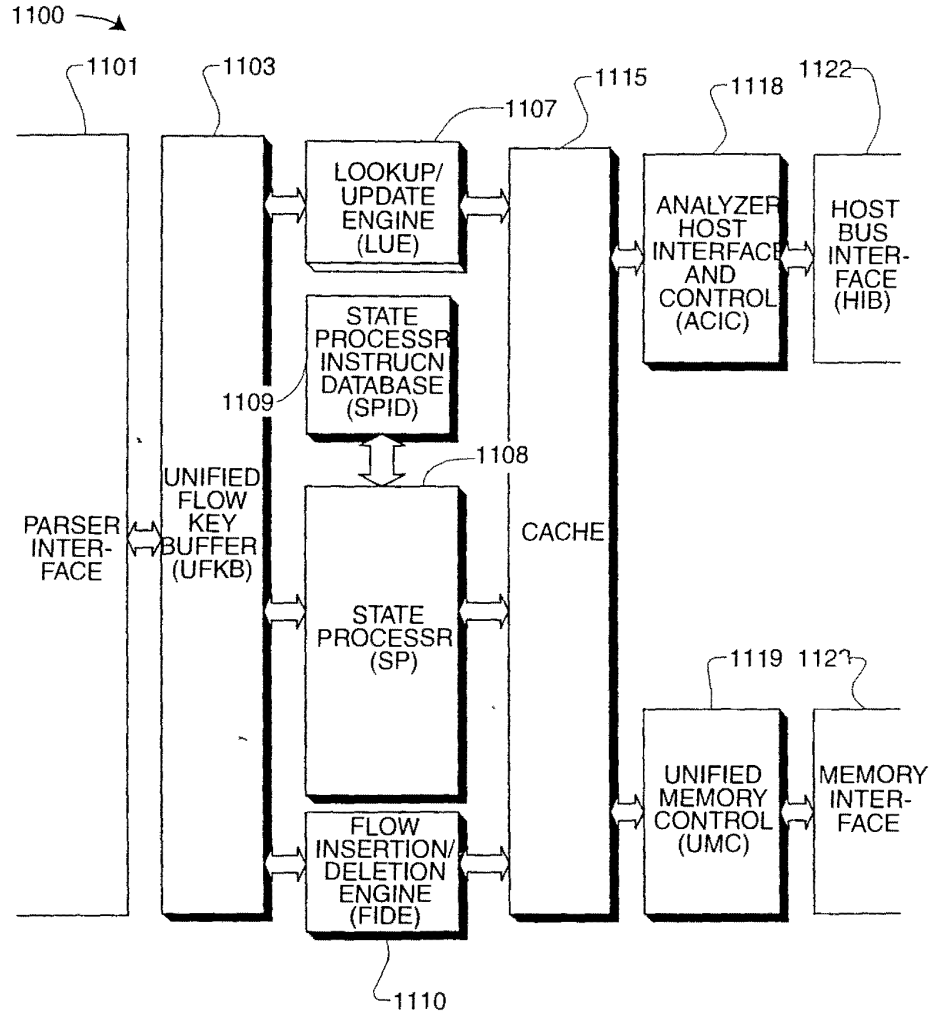


FIG. 11

12/21

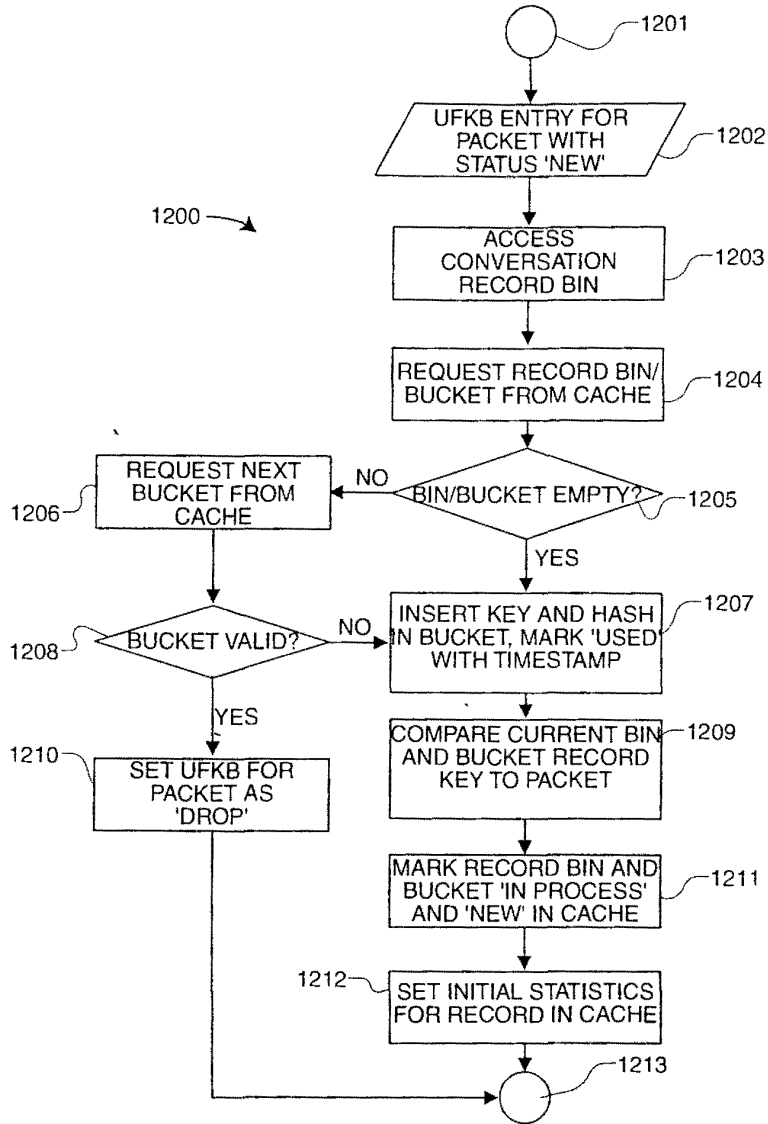


FIG. 12

13/21

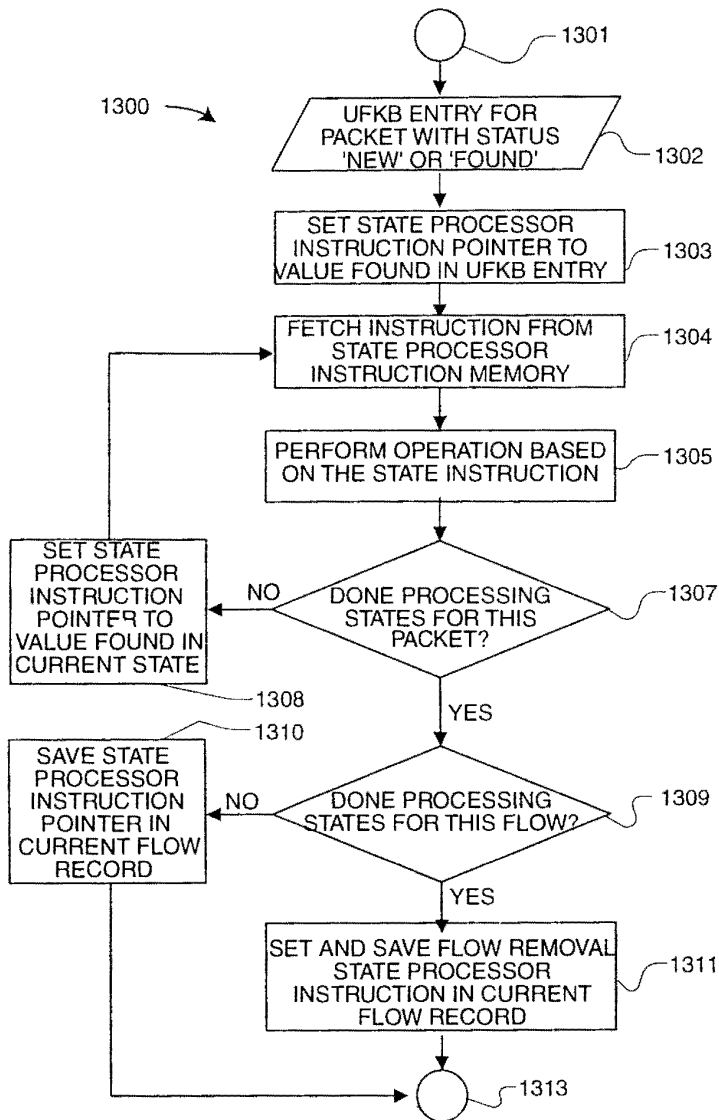


FIG. 13

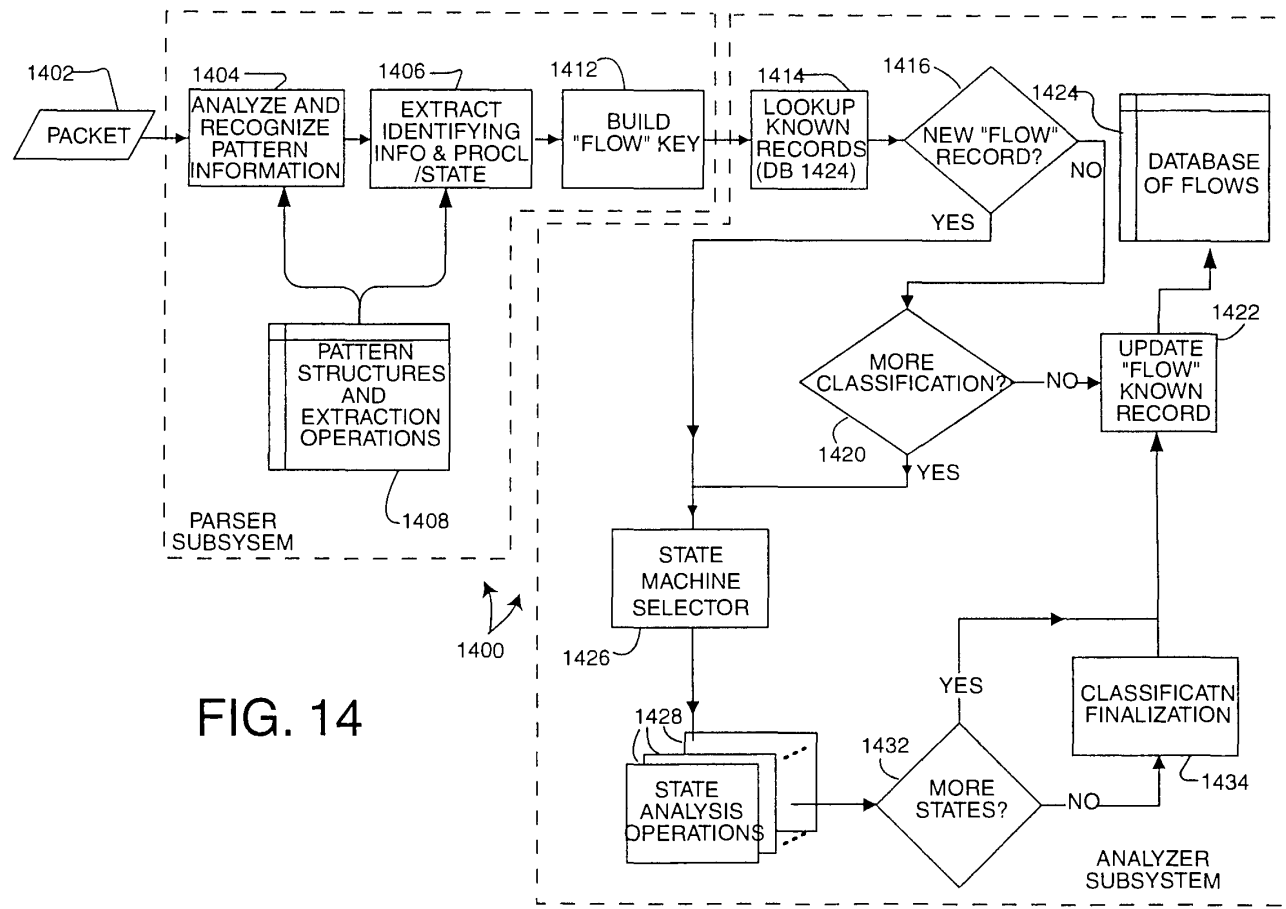


FIG. 14

PRINT OF DRAWINGS
AS ORIGINALLY

14/21

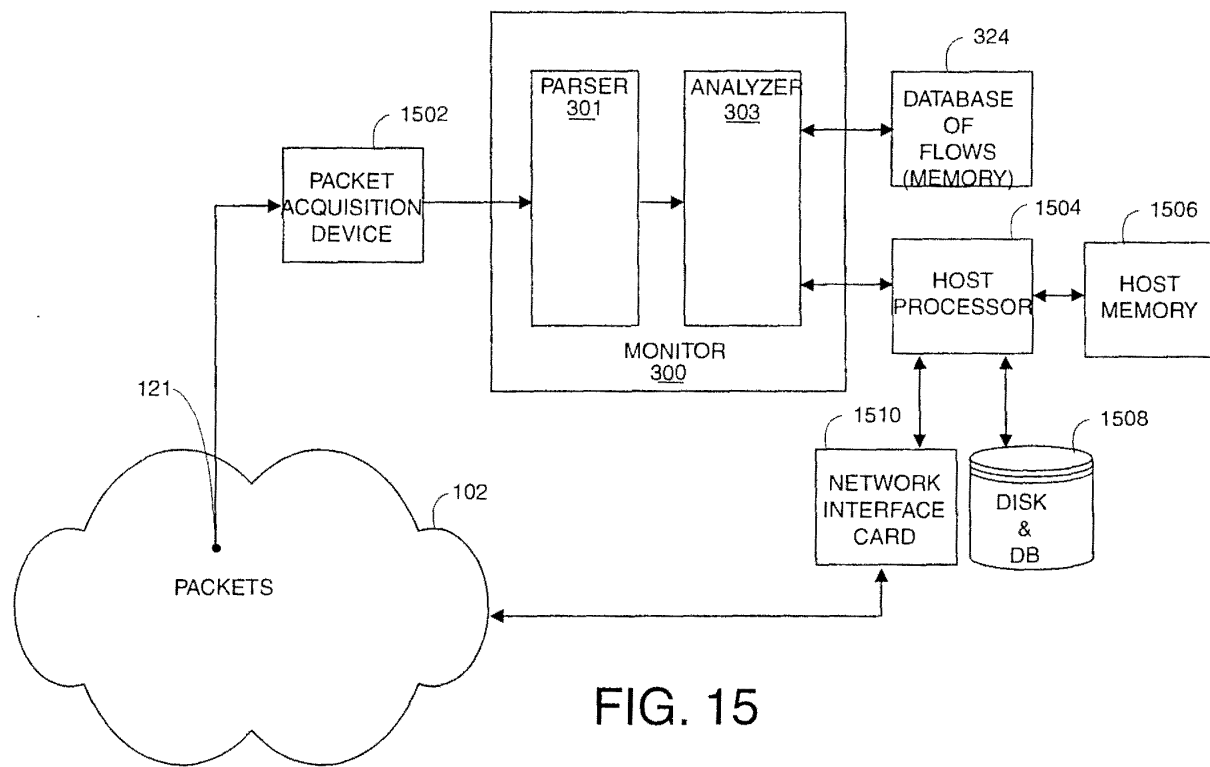


FIG. 15

15/21

PRINT OF DRAWINGS
AS ORIGINALLY

16/21

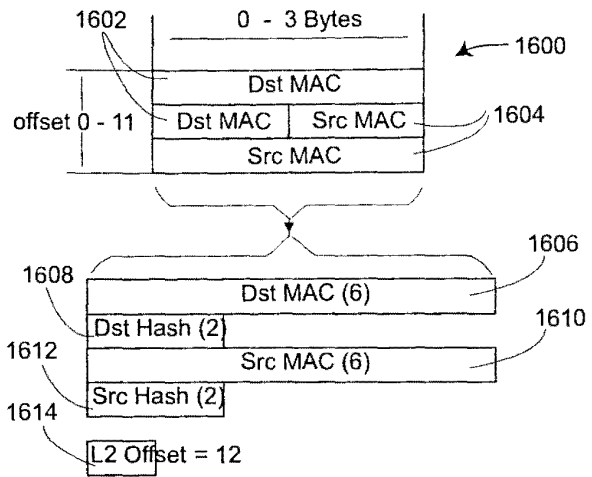


FIG. 16

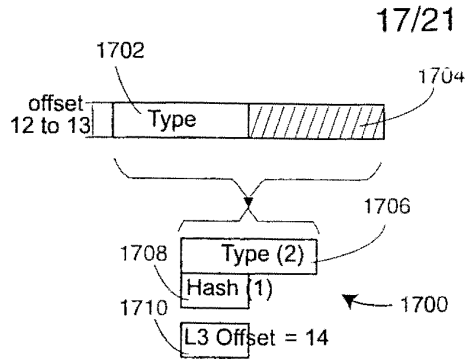


FIG. 17A

- IDP = 0x0600*
 - IP = 0x0800*
 - CHAOSNET = 0x0804
 - ARP = 0x0806
 - VIP = 0x0BAD*
 - VLOOP = 0x0BAE
 - VECHO = 0x0BAF
 - NETBIOS-3COM = 0x3C00 - 0x3C0D#
 - DEC-MOP = 0x6001
 - DEC-RC = 0x6002
 - DEC-DRP = 0x6003*
 - DEC-LAT = 0x6004
 - DEC-DIAG = 0x6005
 - DEC-LAVC = 0x6007
 - RARP = 0x8035
 - ATALK = 0x809B*
 - VLOOP = 0x80C4
 - VECHO = 0x80C5
 - SNA-TH = 0x80D5*
 - ATALKARP = 0x80F3
 - IPX = 0x8137*
 - SNMP = 0x814C#
 - IPv6 = 0x86DD*
 - LOOPBACK = 0x9000
 - Apple = 0x080007
- * L3 Decoding
L5 Decoding

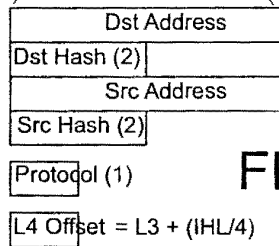
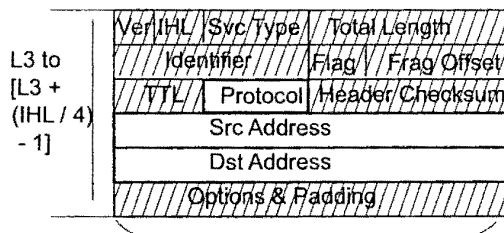


FIG. 17B

- ICMP = 1
 - IGMP = 2
 - GGP = 3
 - TCP = 6*
 - EGP = 8
 - IGRP = 9
 - PUP = 12
 - CHAOS = 16
 - UDP = 17*
 - IDP = 22#
 - ISO-TP4 = 29
 - DDP = 37#
 - ISO-IP = 80
 - VIP = 83#
 - EIGRP = 88
 - OSPF = 89
- * L4 Decoding
L3 Re-Decoding

18/21

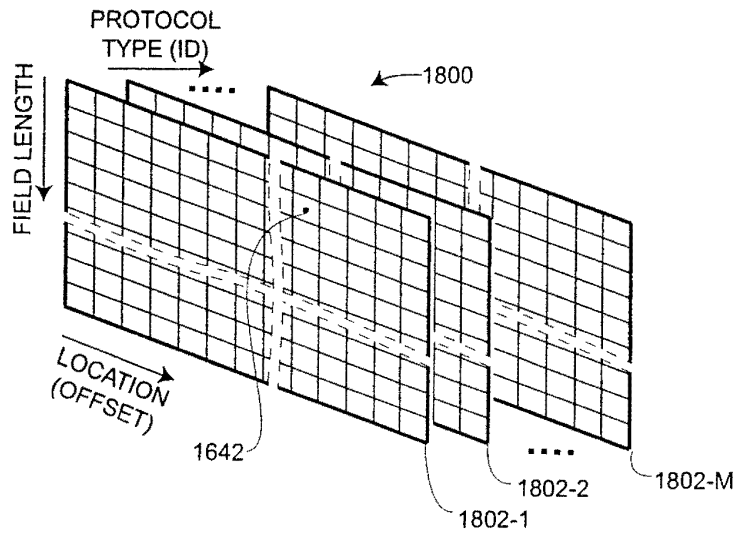


FIG. 18A

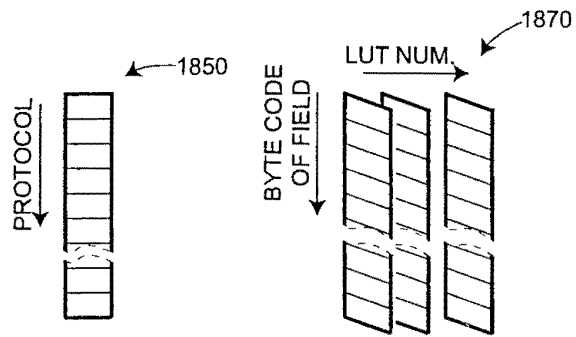
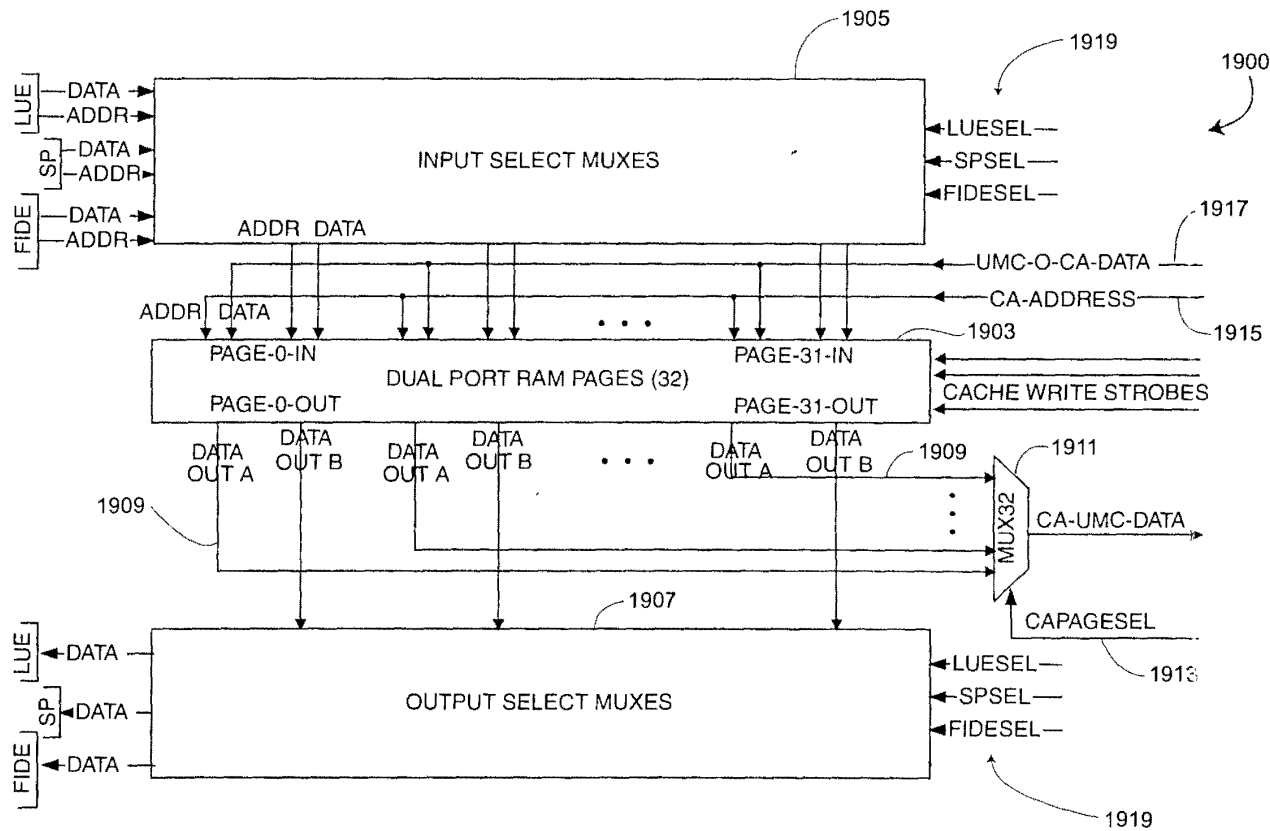


FIG. 18B



19/21

FIG. 19

20/21

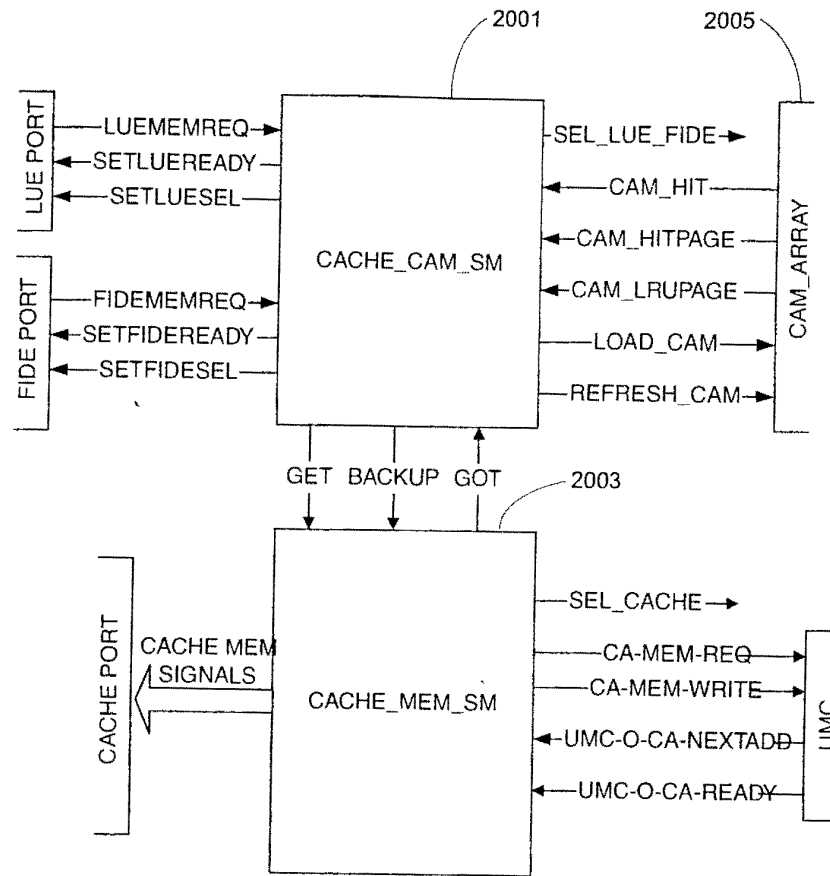


FIG. 20

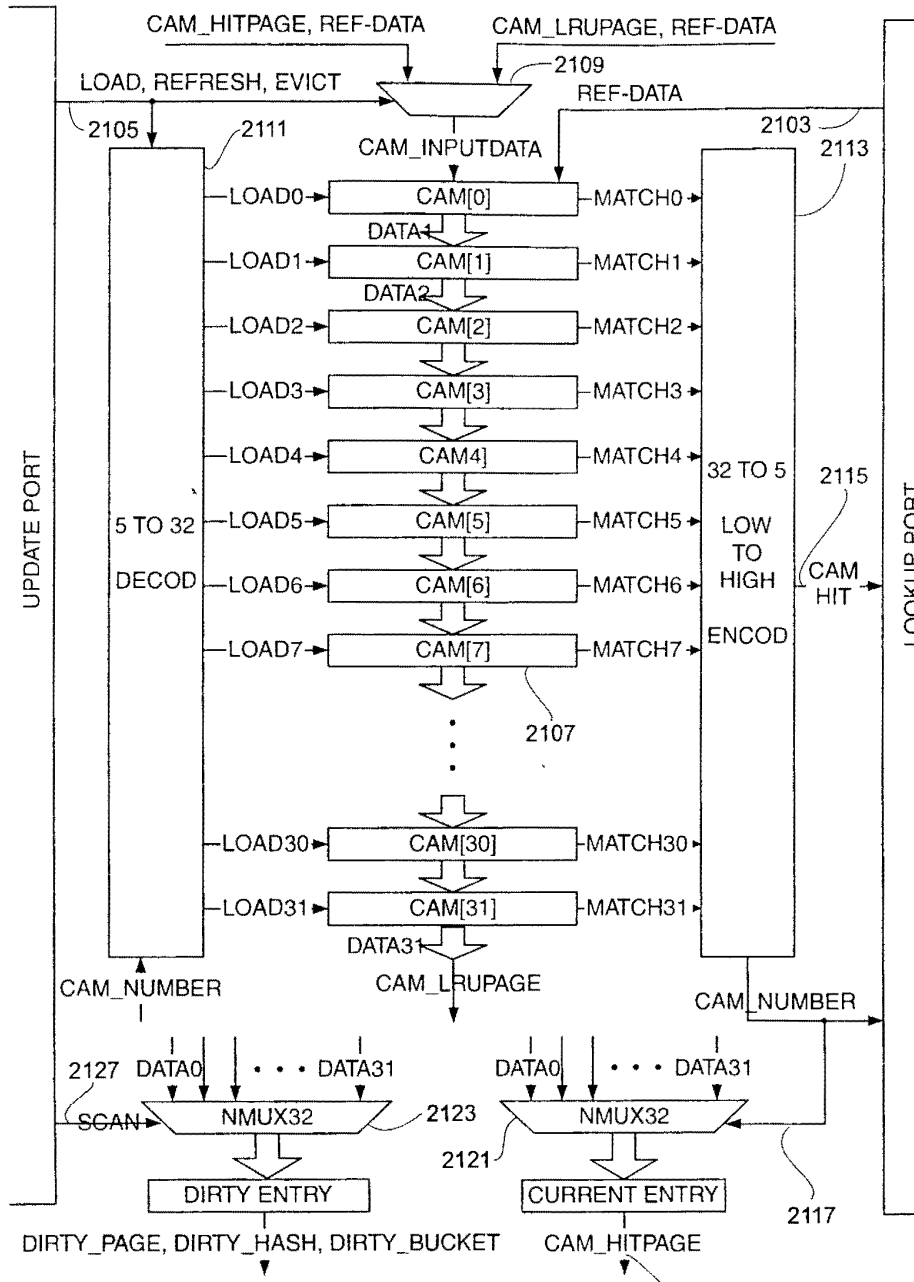


FIG. 21

1/21

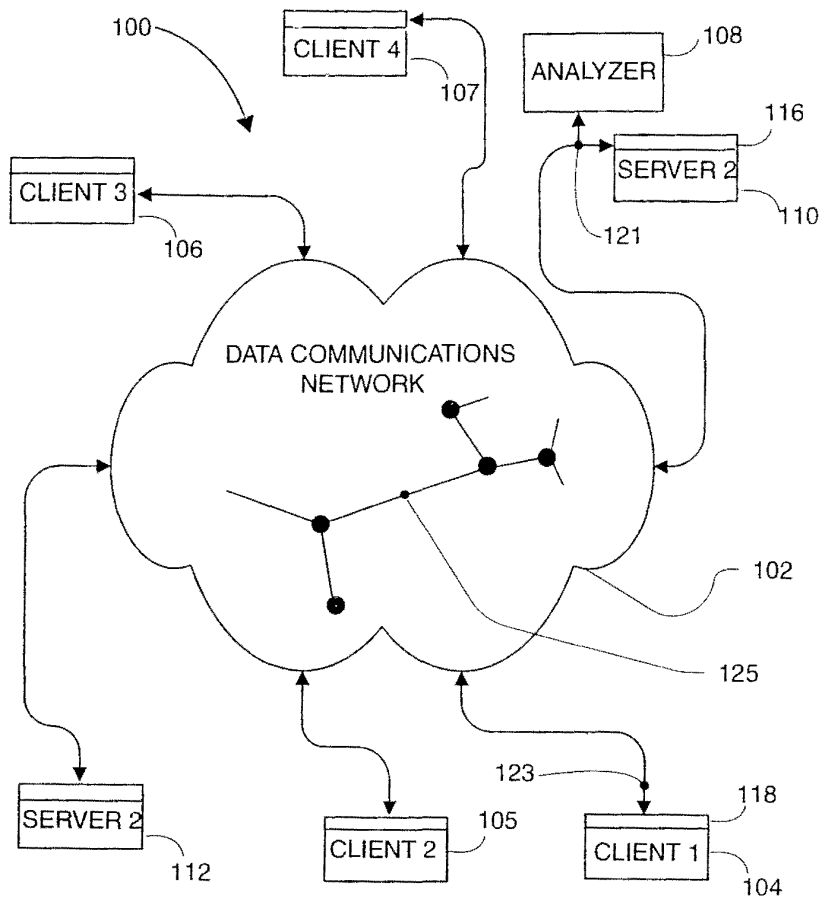


FIG. 1

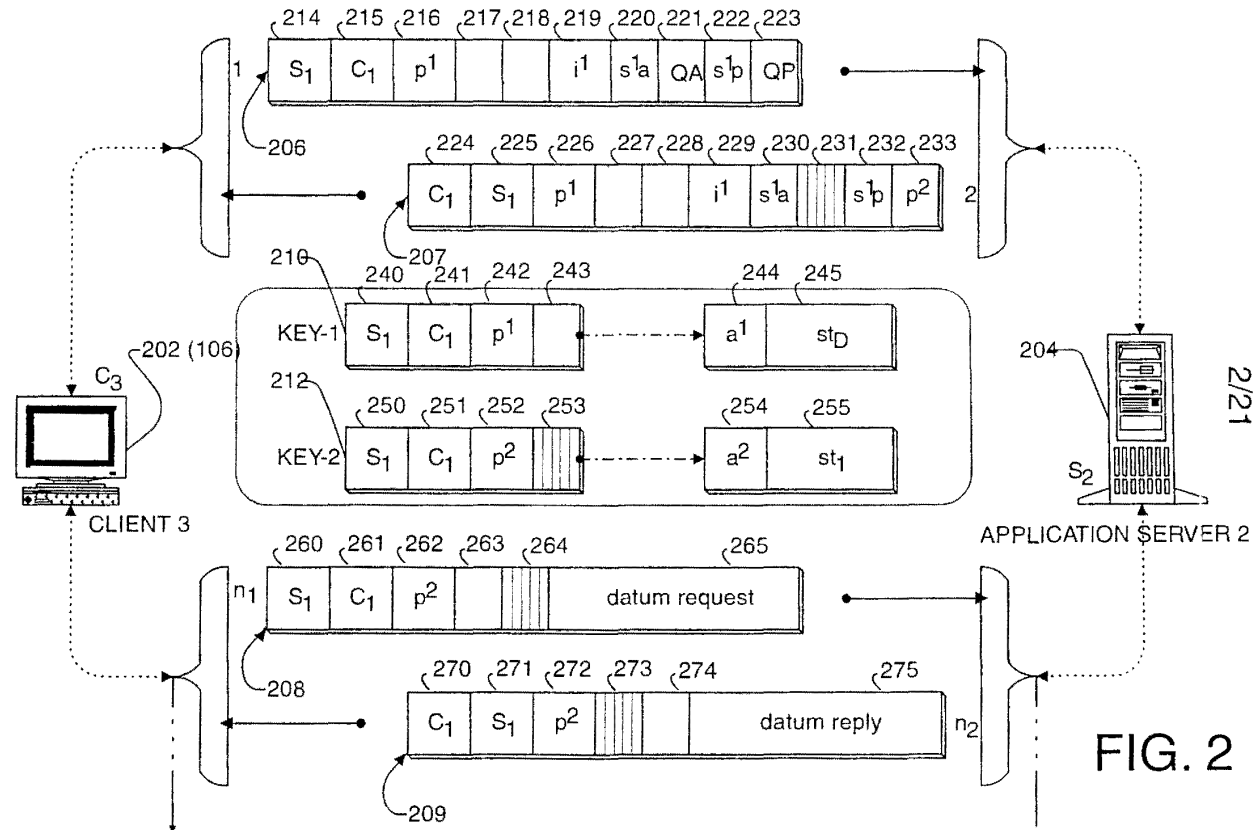


FIG. 2

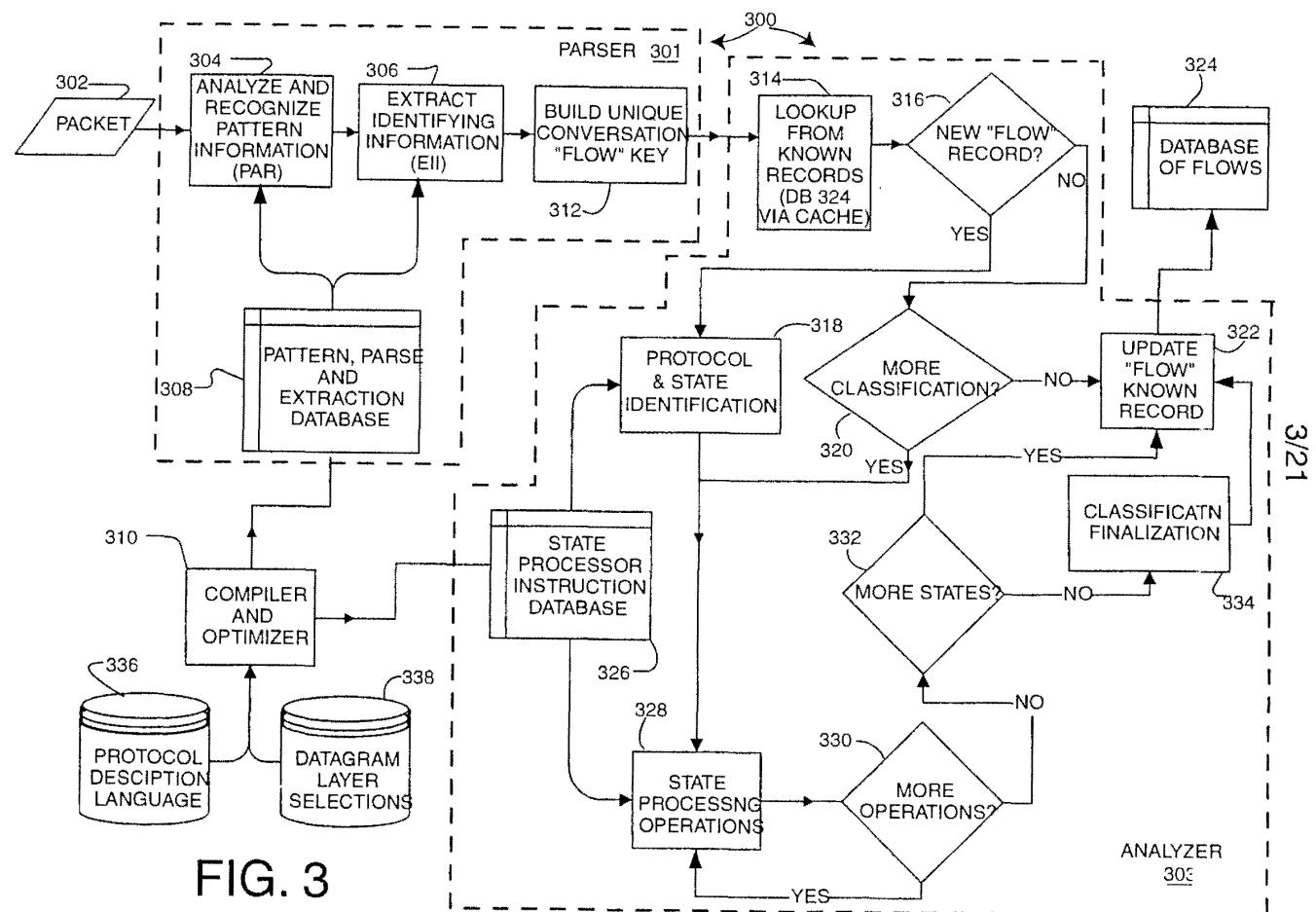


FIG. 3

PRINT OF DRAWING
AS ORIGINALLY

3/21

4/21

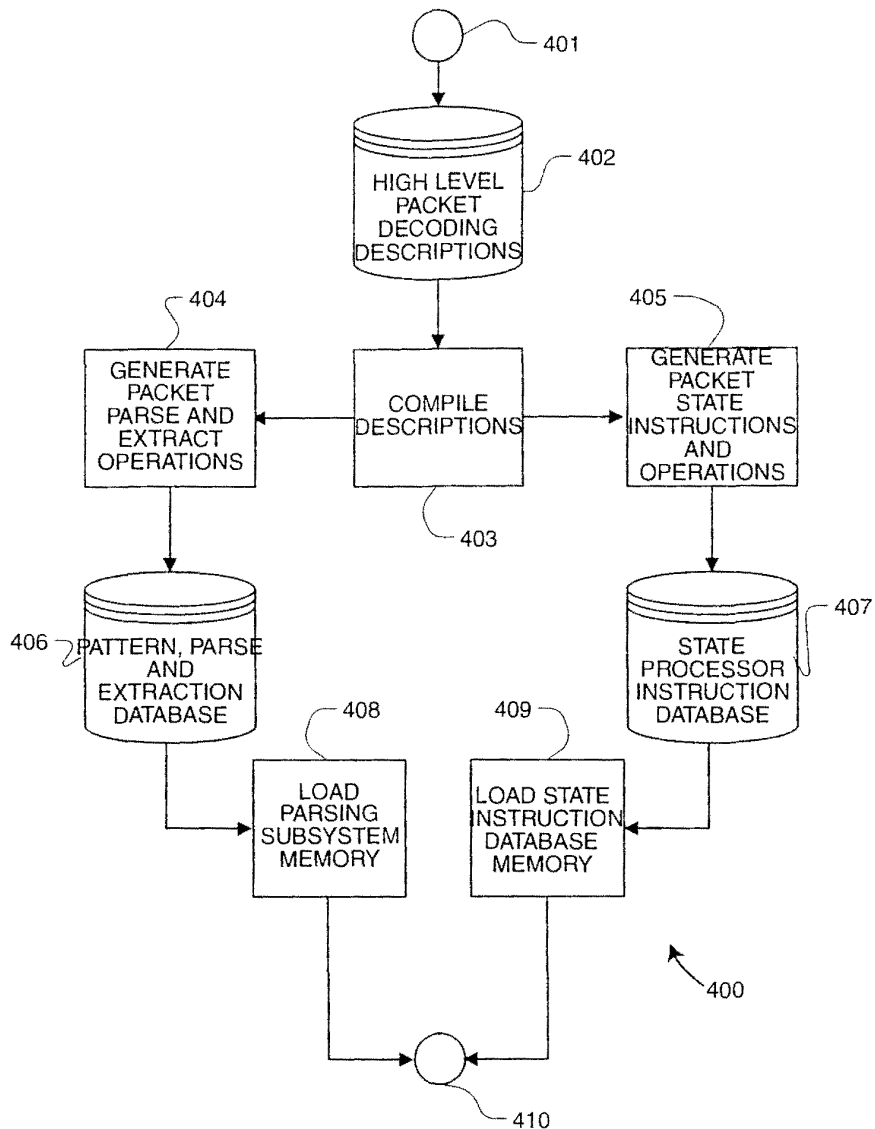


FIG. 4

5/21

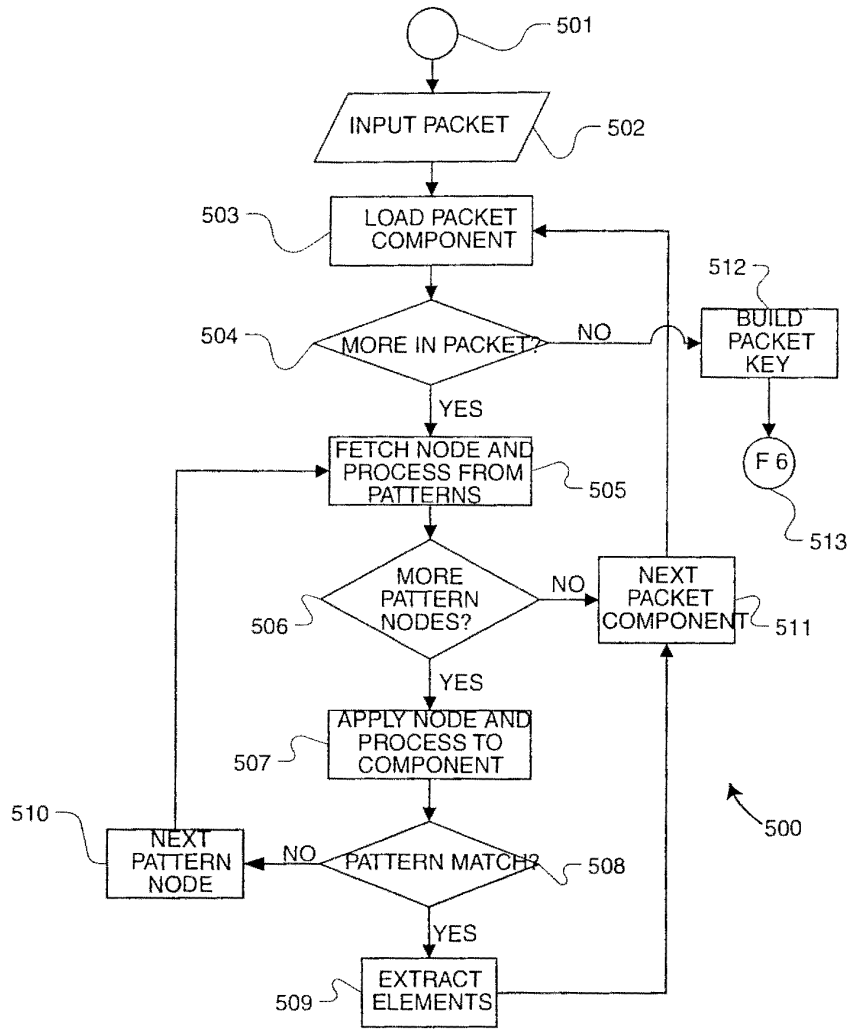


FIG. 5

6/21

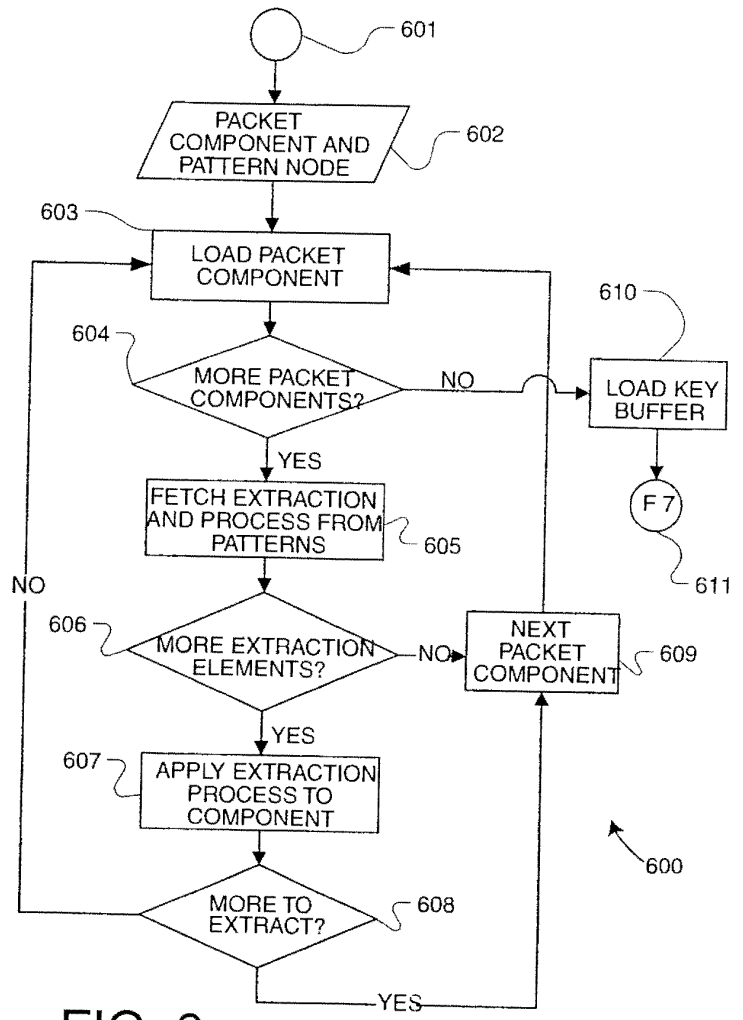


FIG. 6

7/21

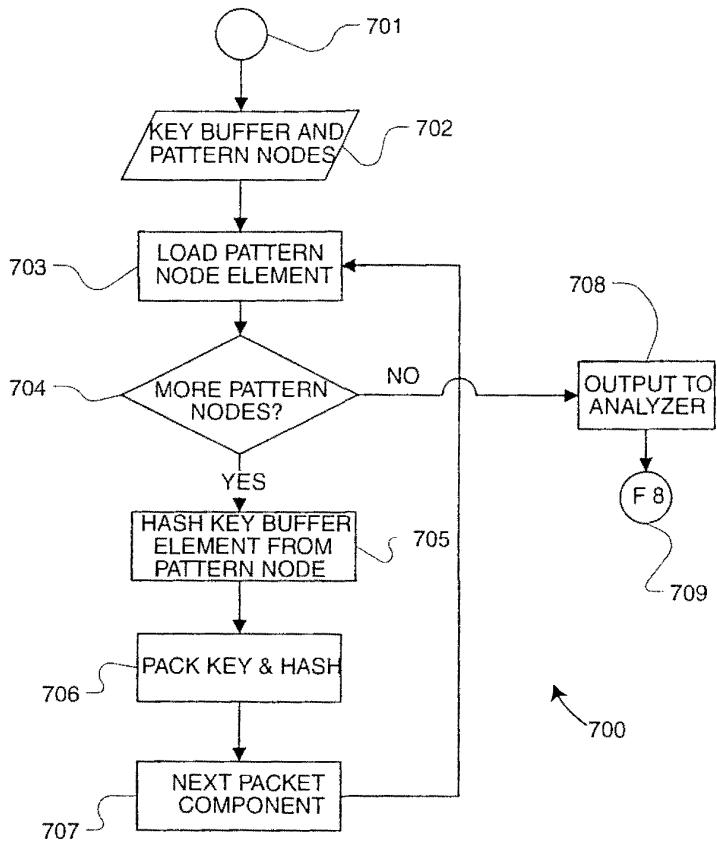


FIG. 7

8/21

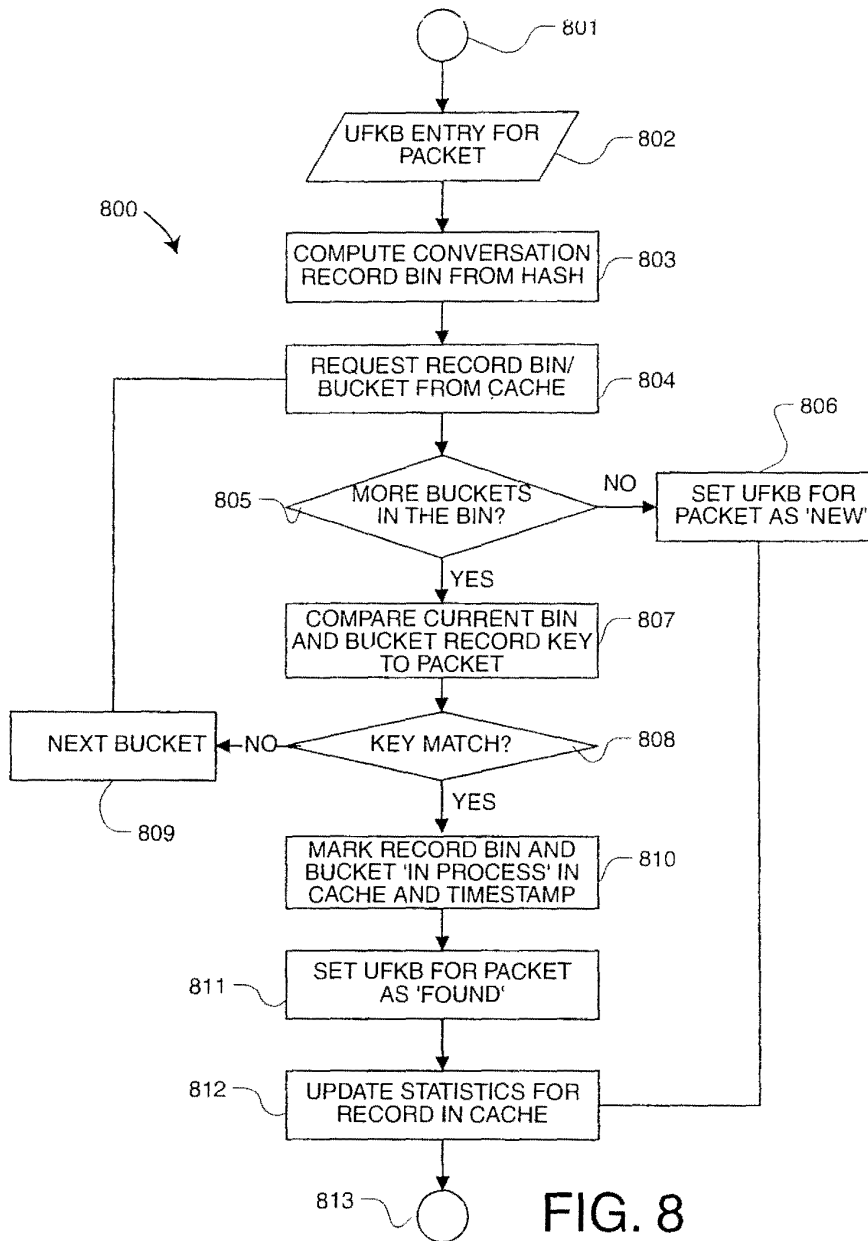


FIG. 8

9/21

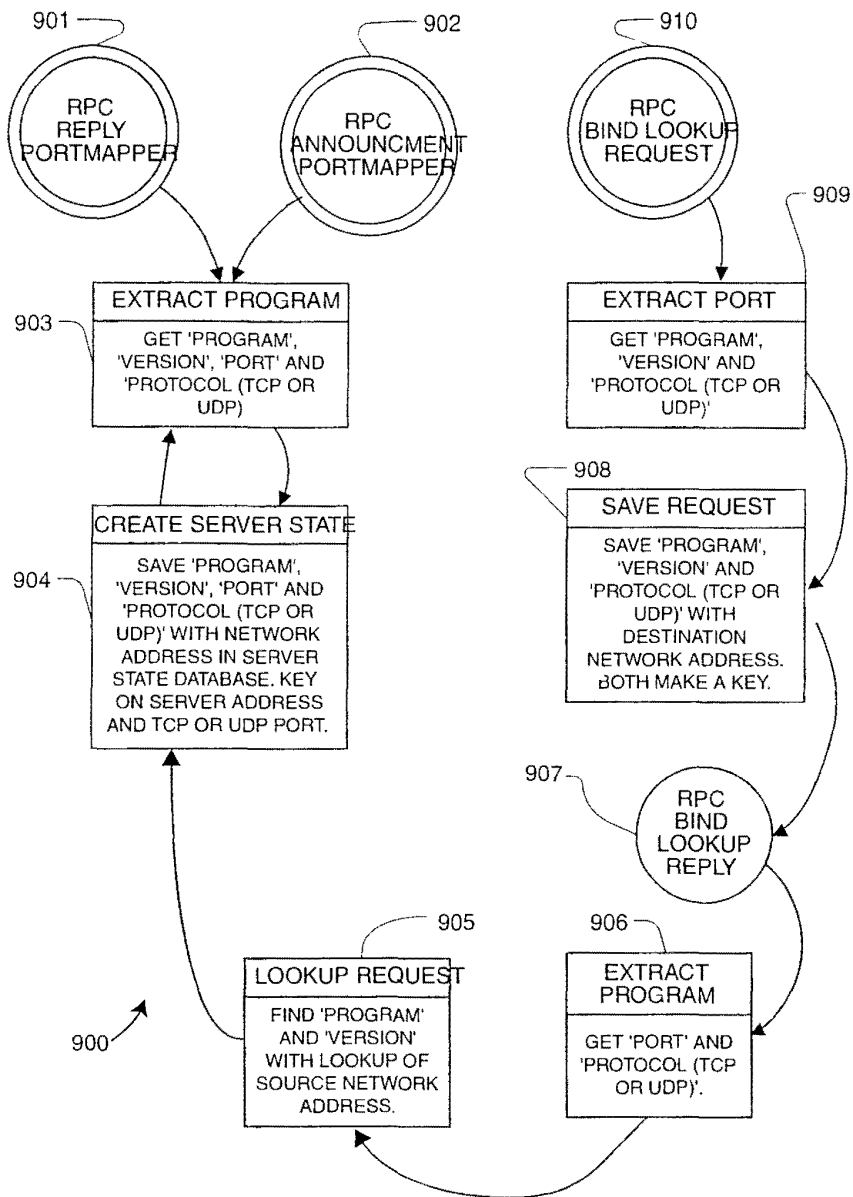
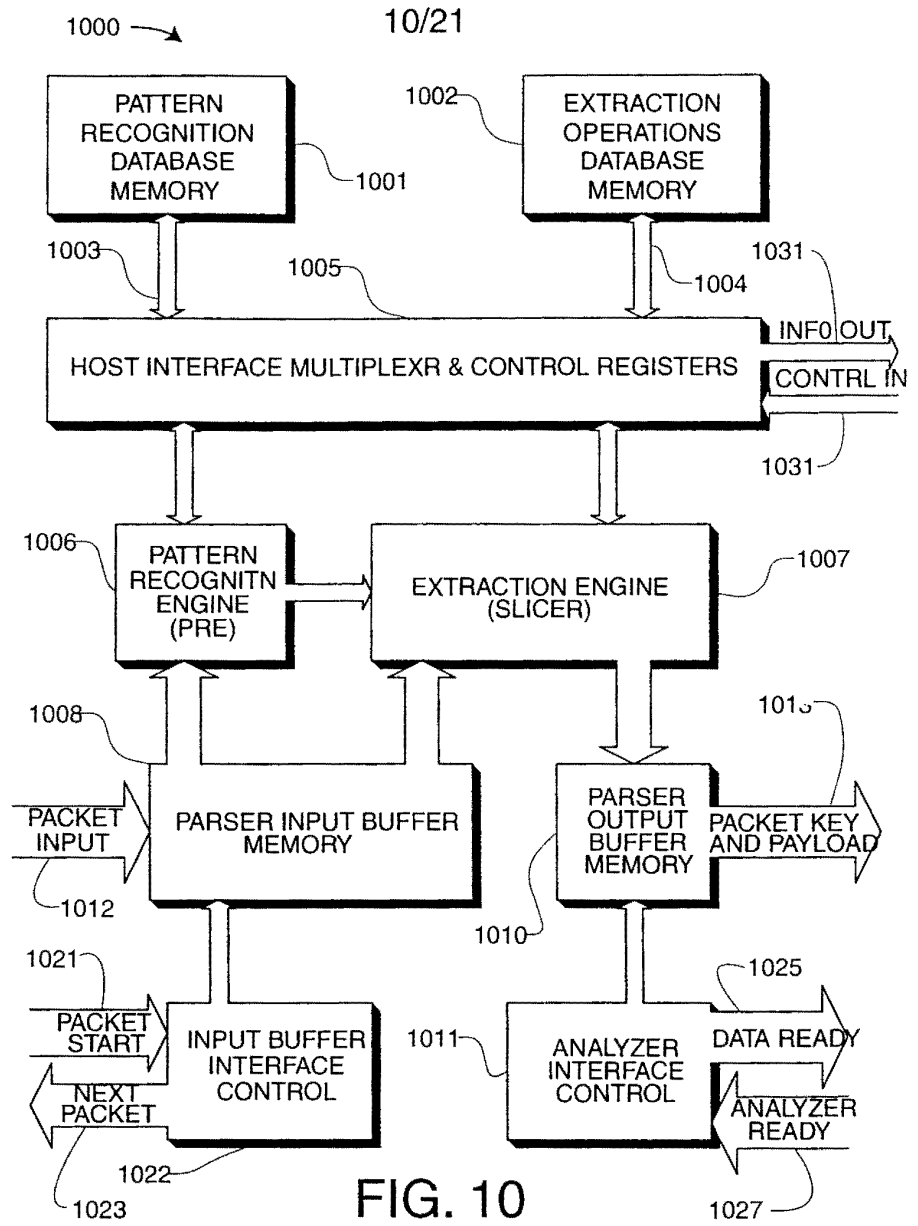


FIG. 9



11/21

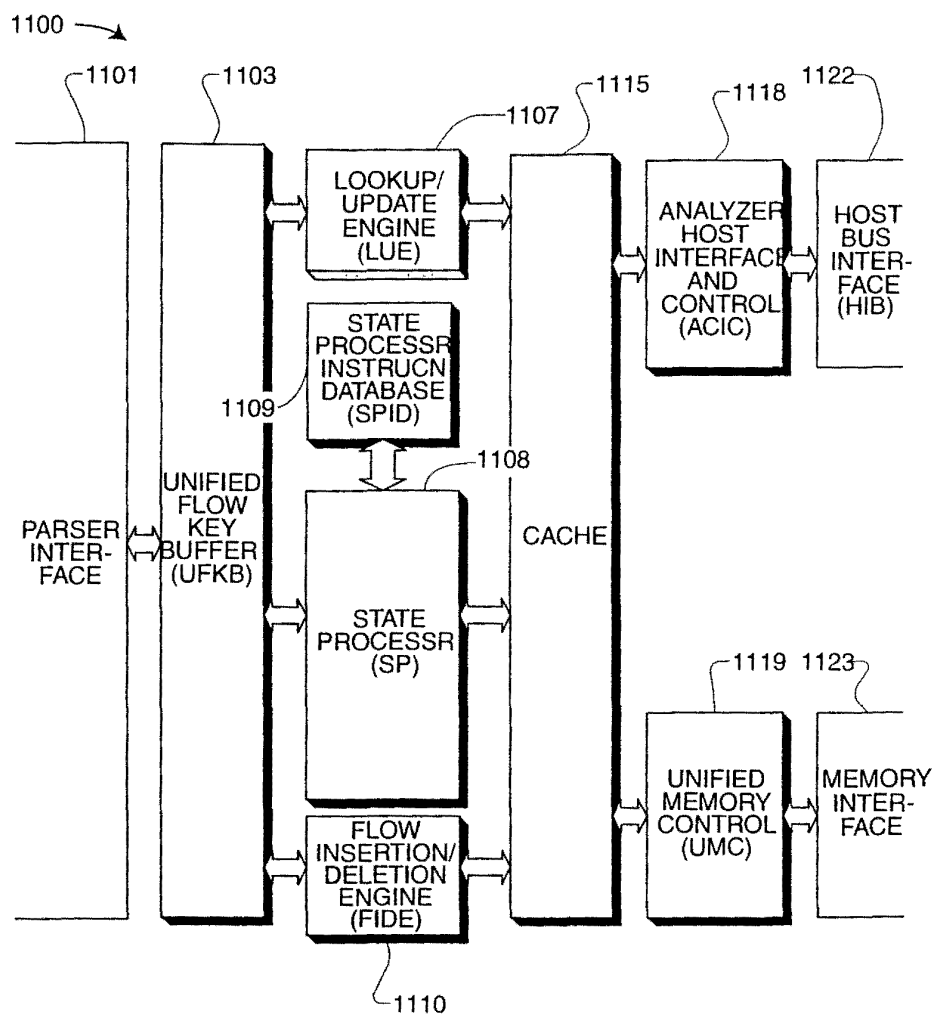


FIG. 11

12/21

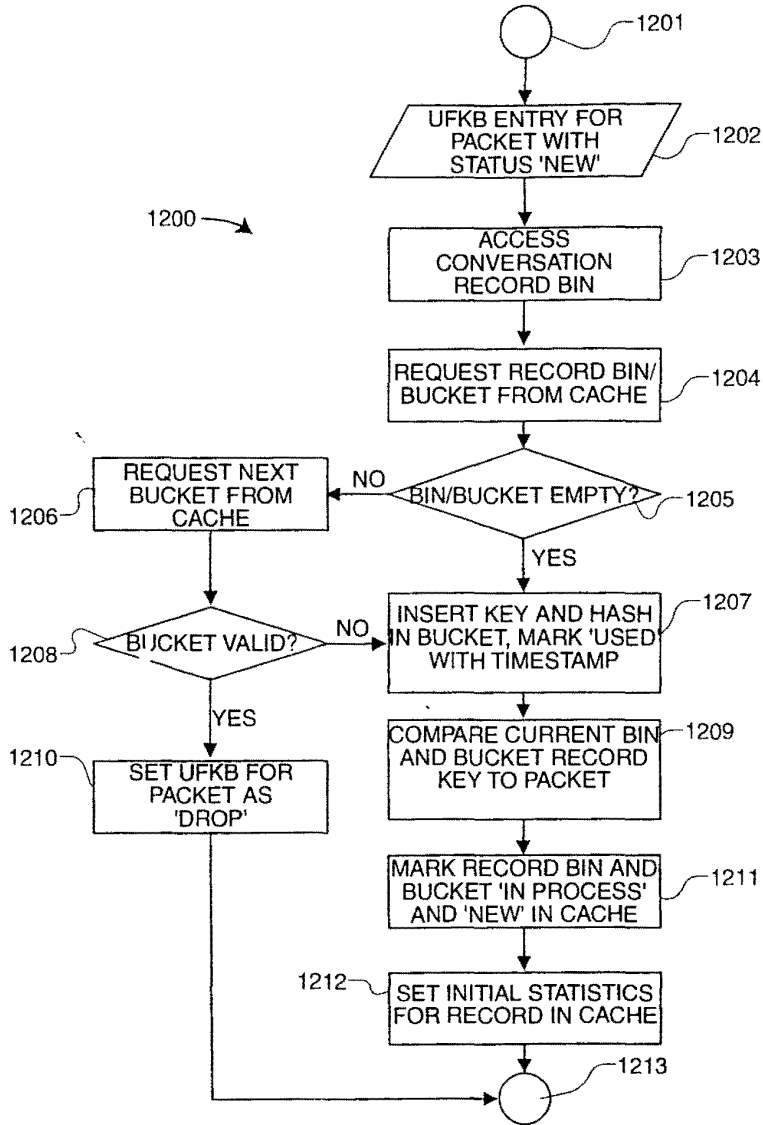


FIG. 12

13/21

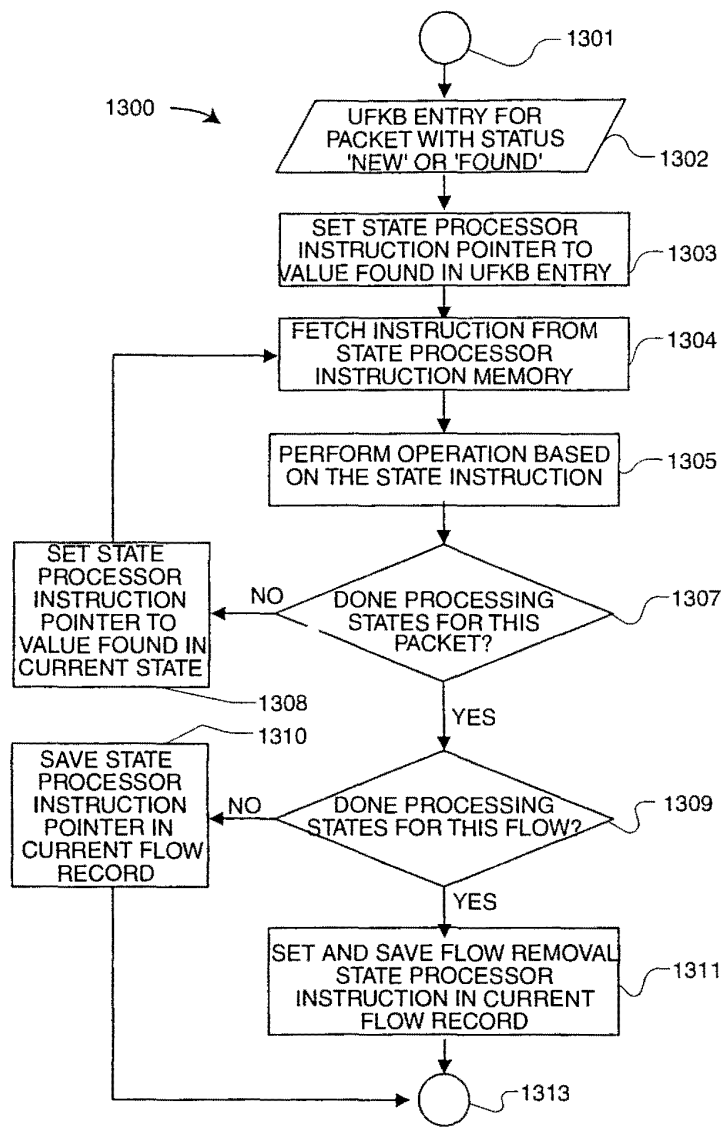
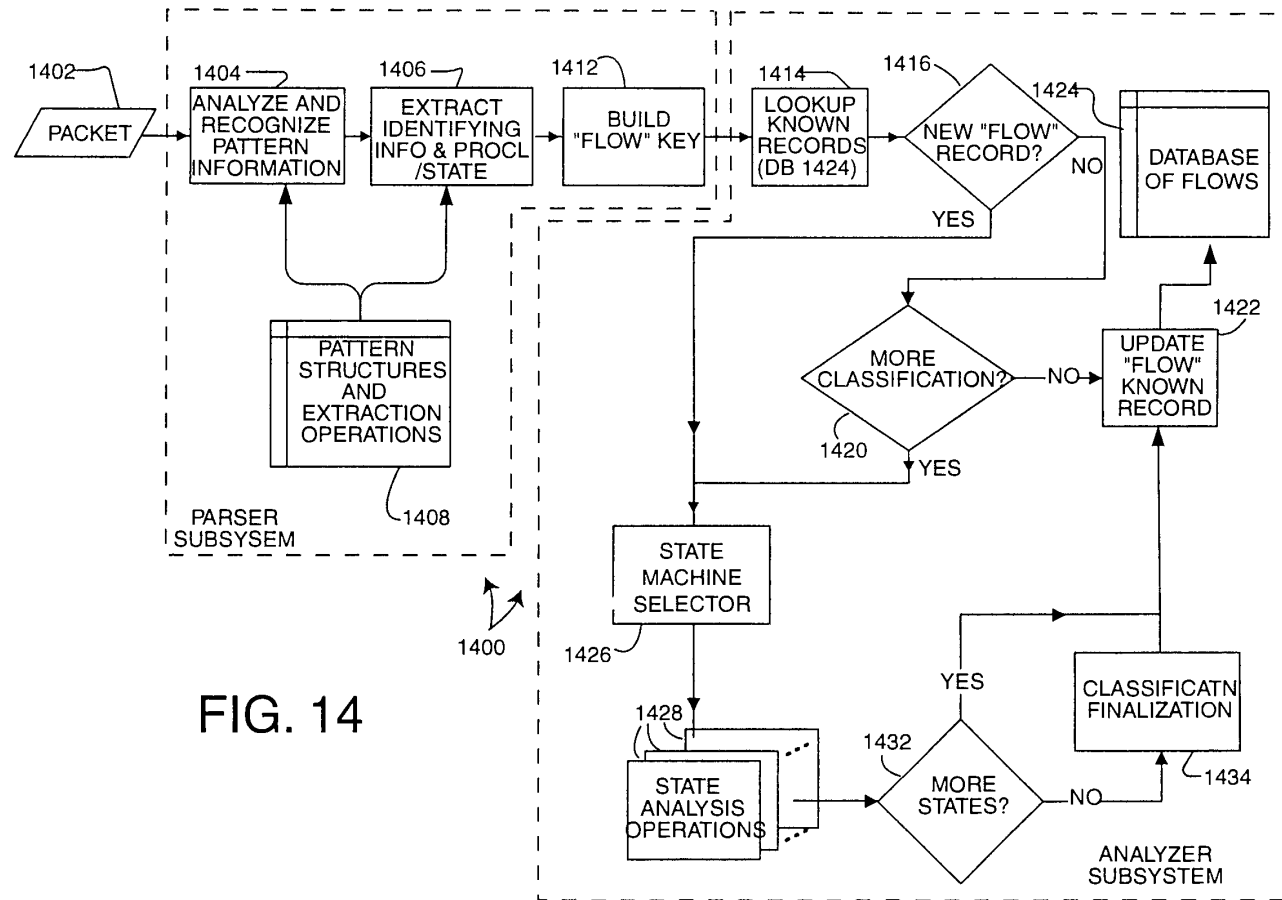


FIG. 13



PRINT OF DRAWINGS
AS ORIGINALLY

14/21

FIG. 15

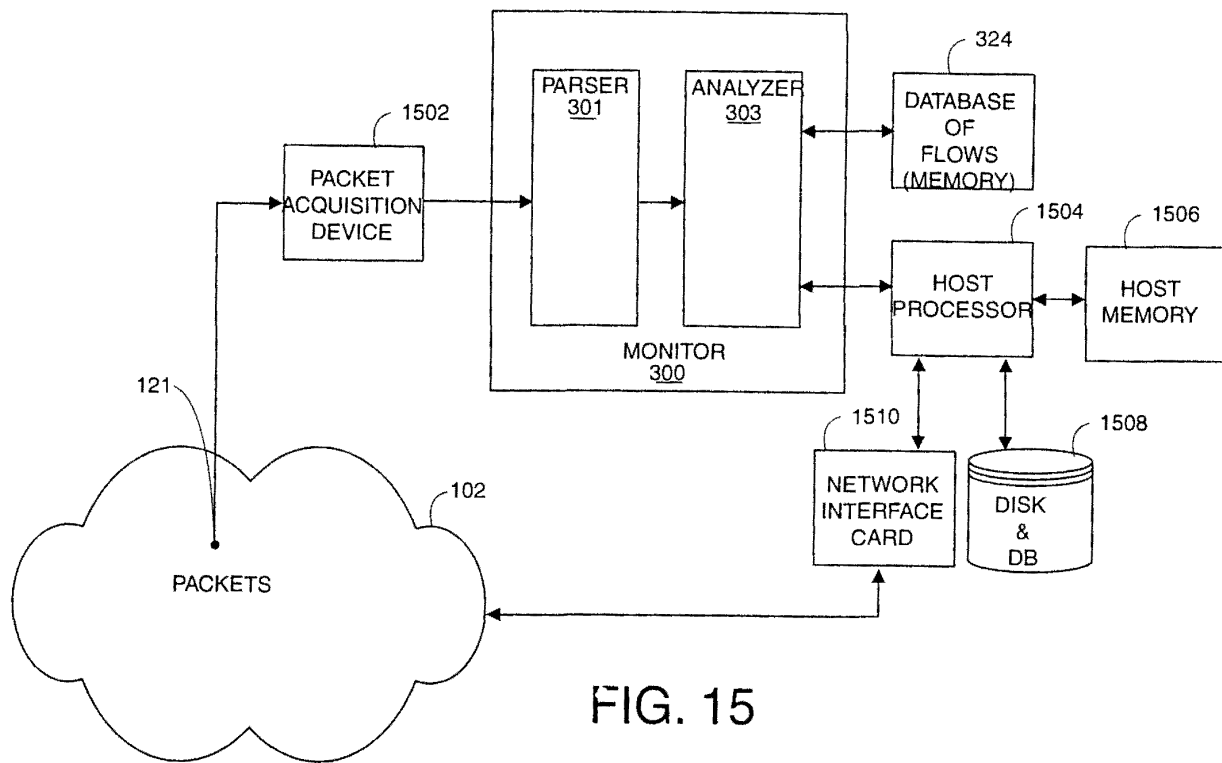


FIG. 15

PRINT OF DRAWINGS
AS ORIGINALLY

15/21

16/21

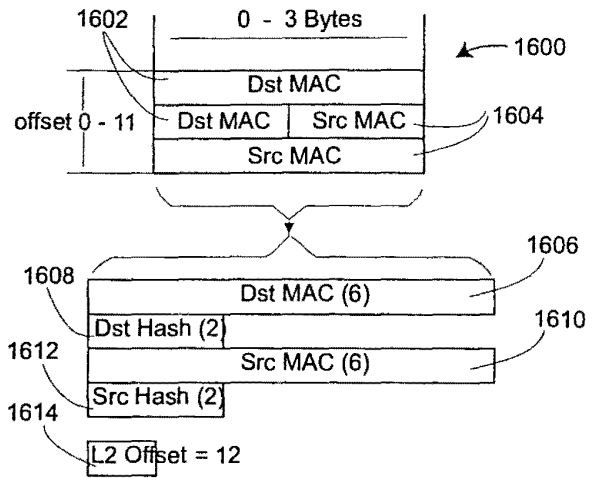


FIG. 16

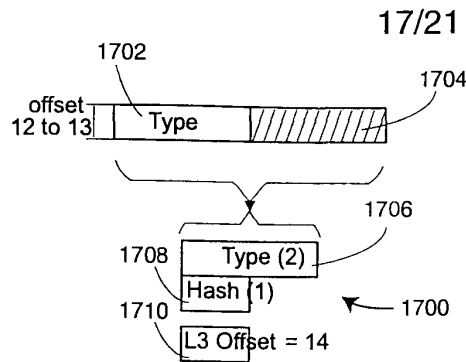


FIG. 17A

IDP	= 0x0600*
IP	= 0x0800*
CHAOSNET	= 0x0804
ARP	= 0x0806
VIP	= 0x0BAD*
VLOOP	= 0x0BAE
VECHO	= 0x0BAF
NETBIOS-3COM	= 0x3C00 -
	0x3C0D#
DEC-MOP	= 0x6001
DEC-RC	= 0x6002
DEC-DRP	= 0x6003*
DEC-LAT	= 0x6004
DEC-DIAG	= 0x6005
DEC-LAVC	= 0x6007
RARP	= 0x8035
ATALK	= 0x809B*
VLOOP	= 0x80C4
VECHO	= 0x80C5
SNA-TH	= 0x80D5*
ATALKARP	= 0x80F3
IPX	= 0x8137*
SNMP	= 0x814C#
IPv6	= 0x86DD*
LOOPBACK	= 0x9000
Apple	= 0x080007

* L3 Decoding
 # L5 Decoding

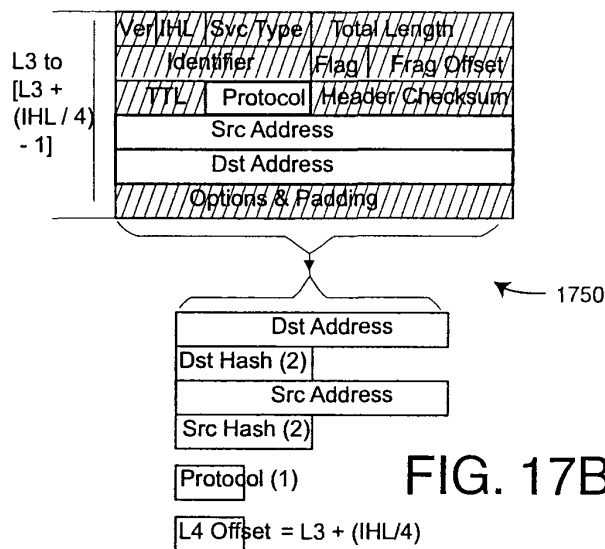


FIG. 17B

ICMP	= 1
IGMP	= 2
GGP	= 3
TCP	= 6*
EGP	= 8
IGRP	= 9
PUP	= 12
CHAOS	= 16
UDP	= 17*
IDP	= 22#
ISO-TP4	= 29
DDP	= 37#
ISO-IP	= 80
VIP	= 83#
EIGRP	= 88
OSPF	= 89

* L4 Decoding
 # L3 Re-Decoding

18/21

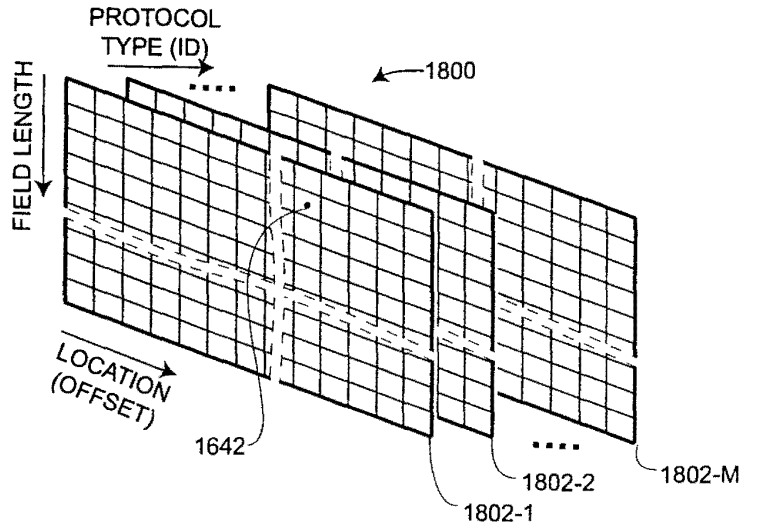


FIG. 18A

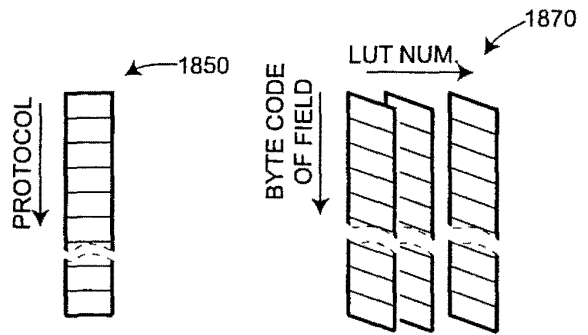


FIG. 18B

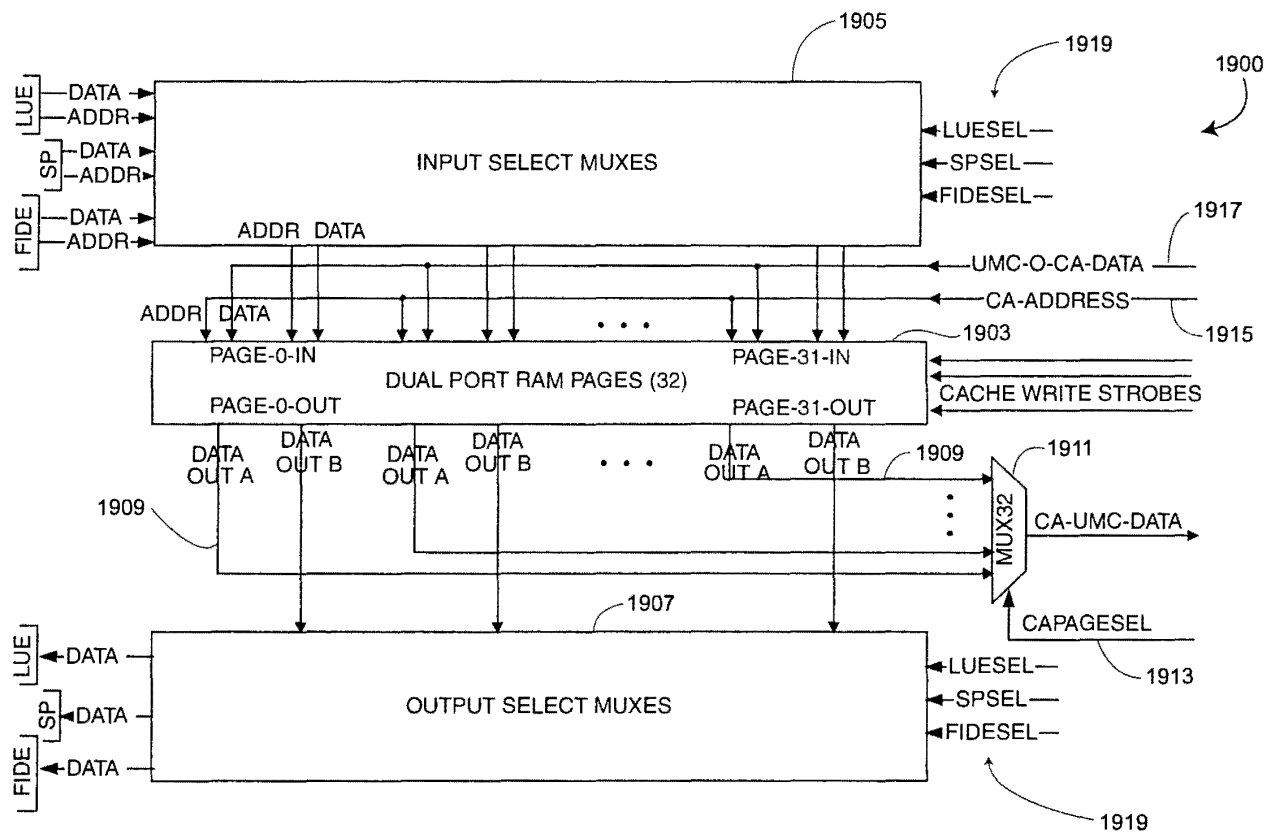


FIG. 19

PRINT OF DRAWINGS
 AS ORIGINALLY

19/21

20/21

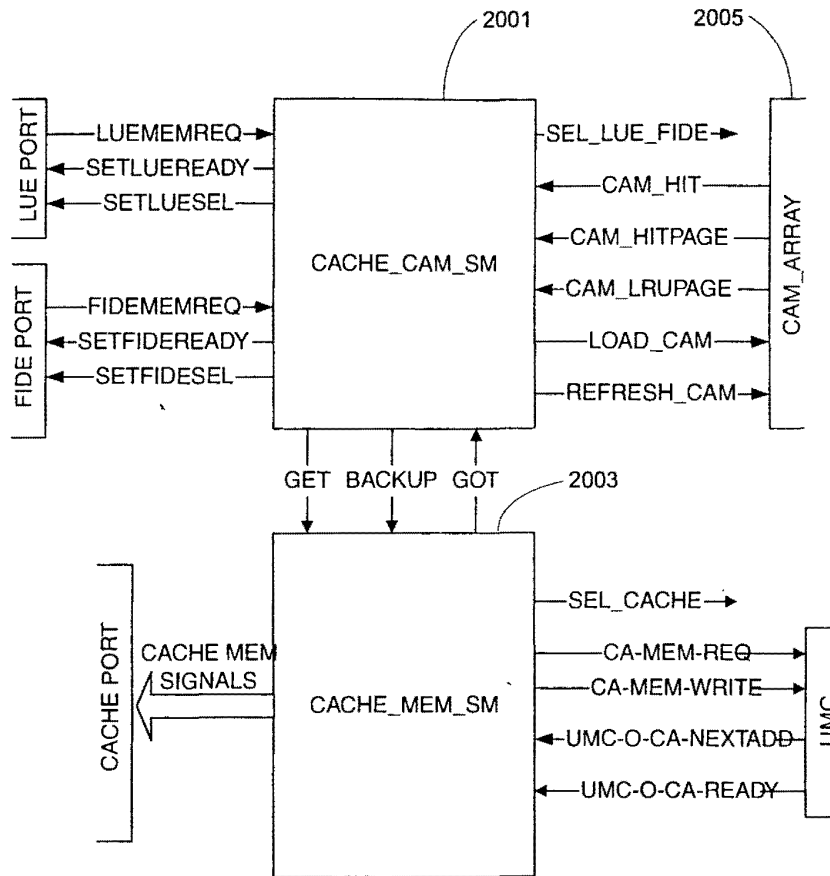


FIG. 20

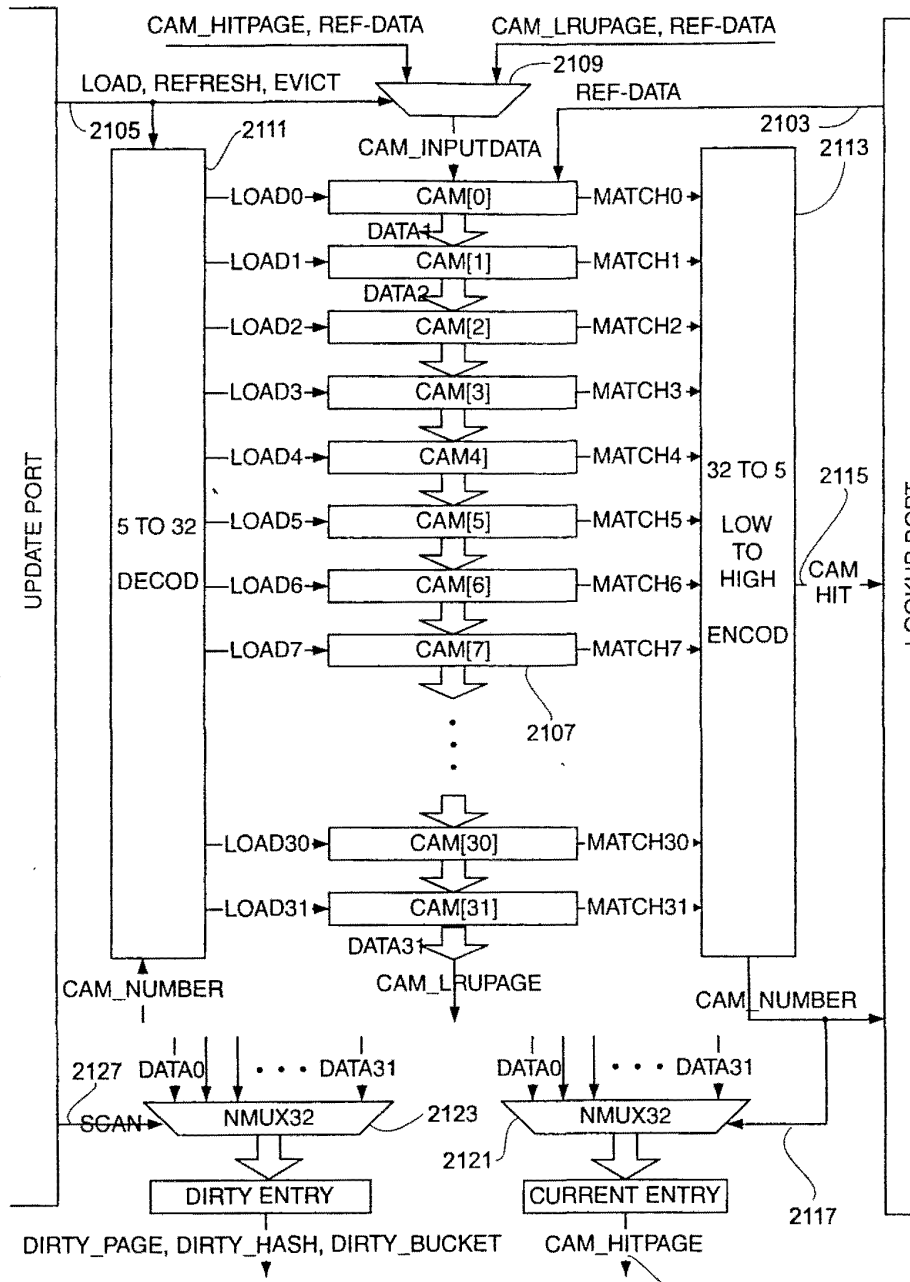


FIG. 21

6771646

1/21

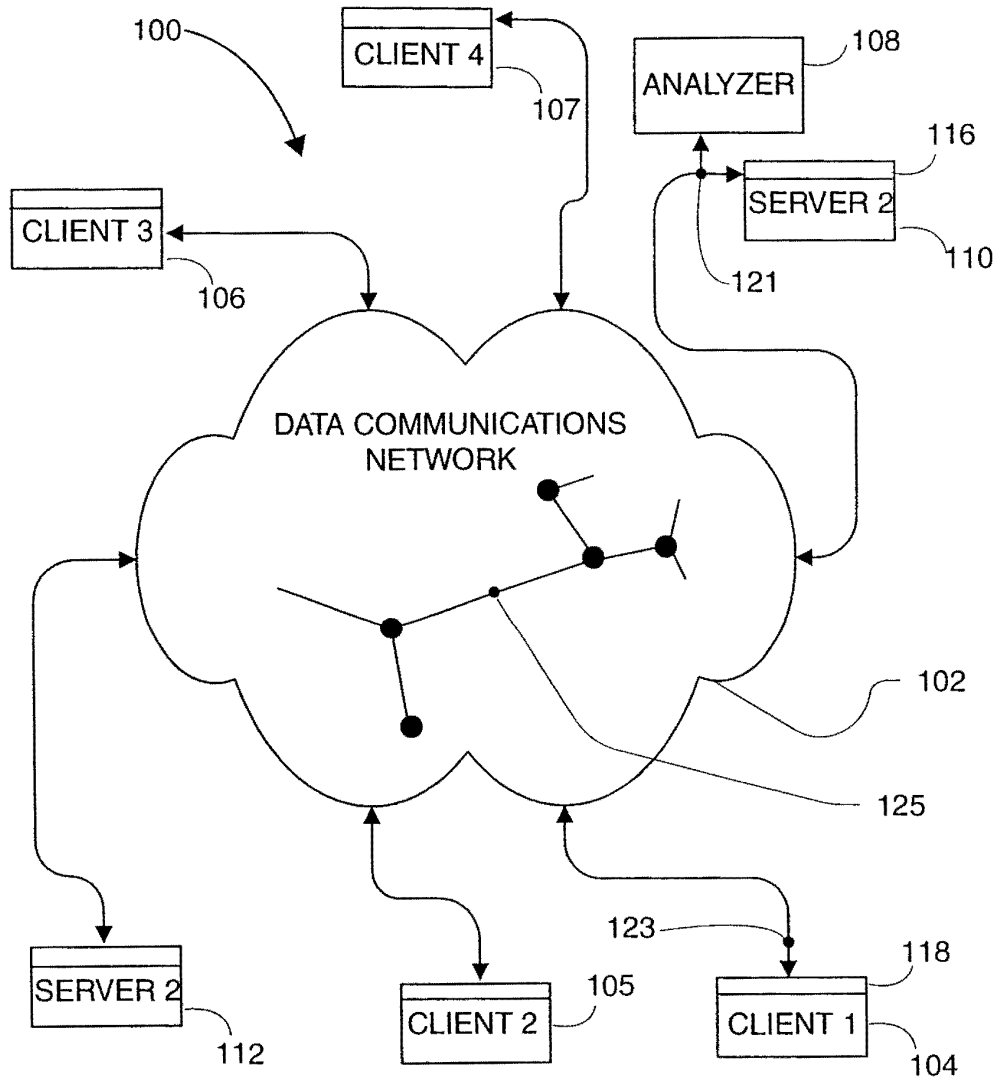


FIG. 1

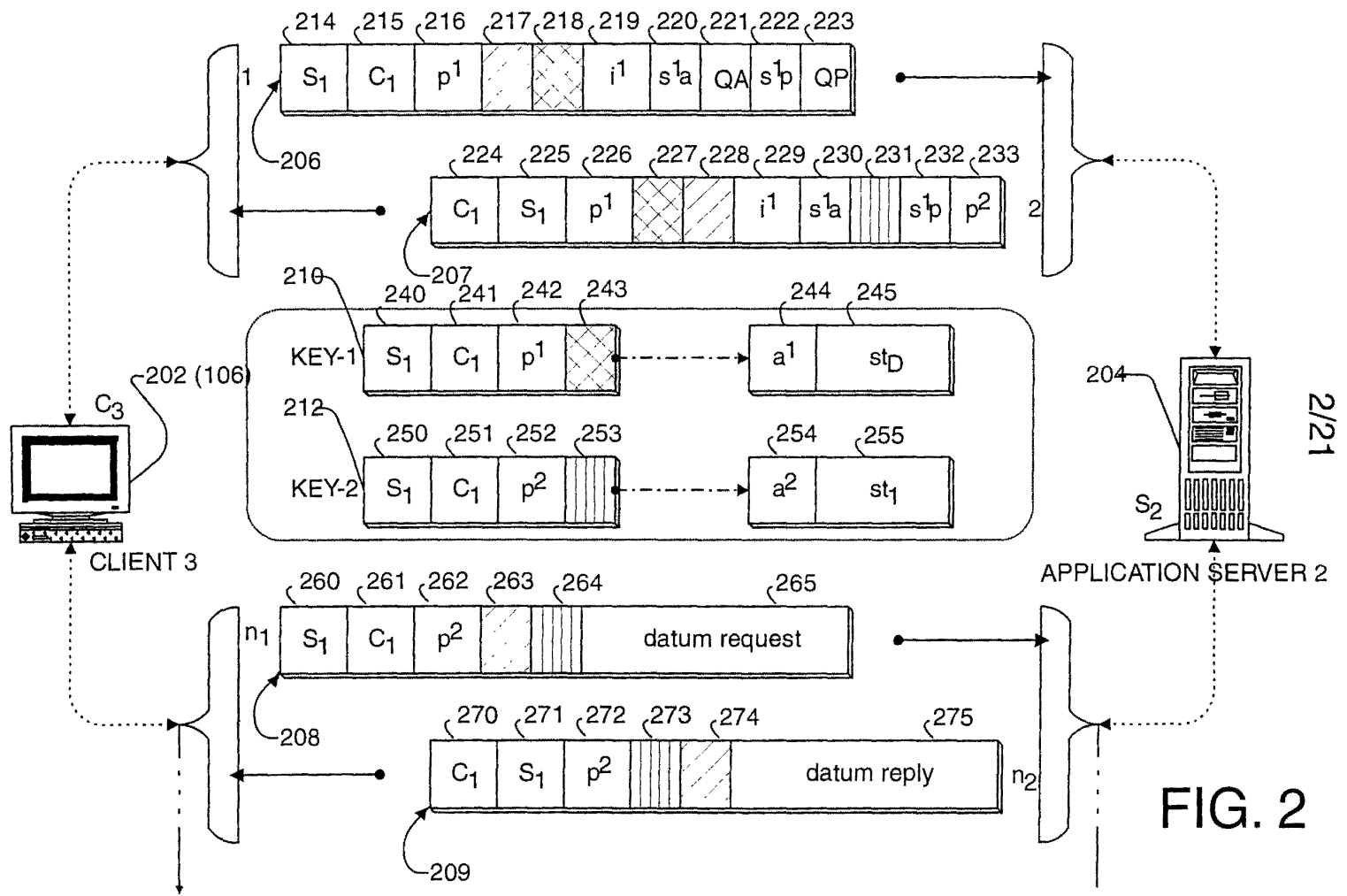


FIG. 2

2/21

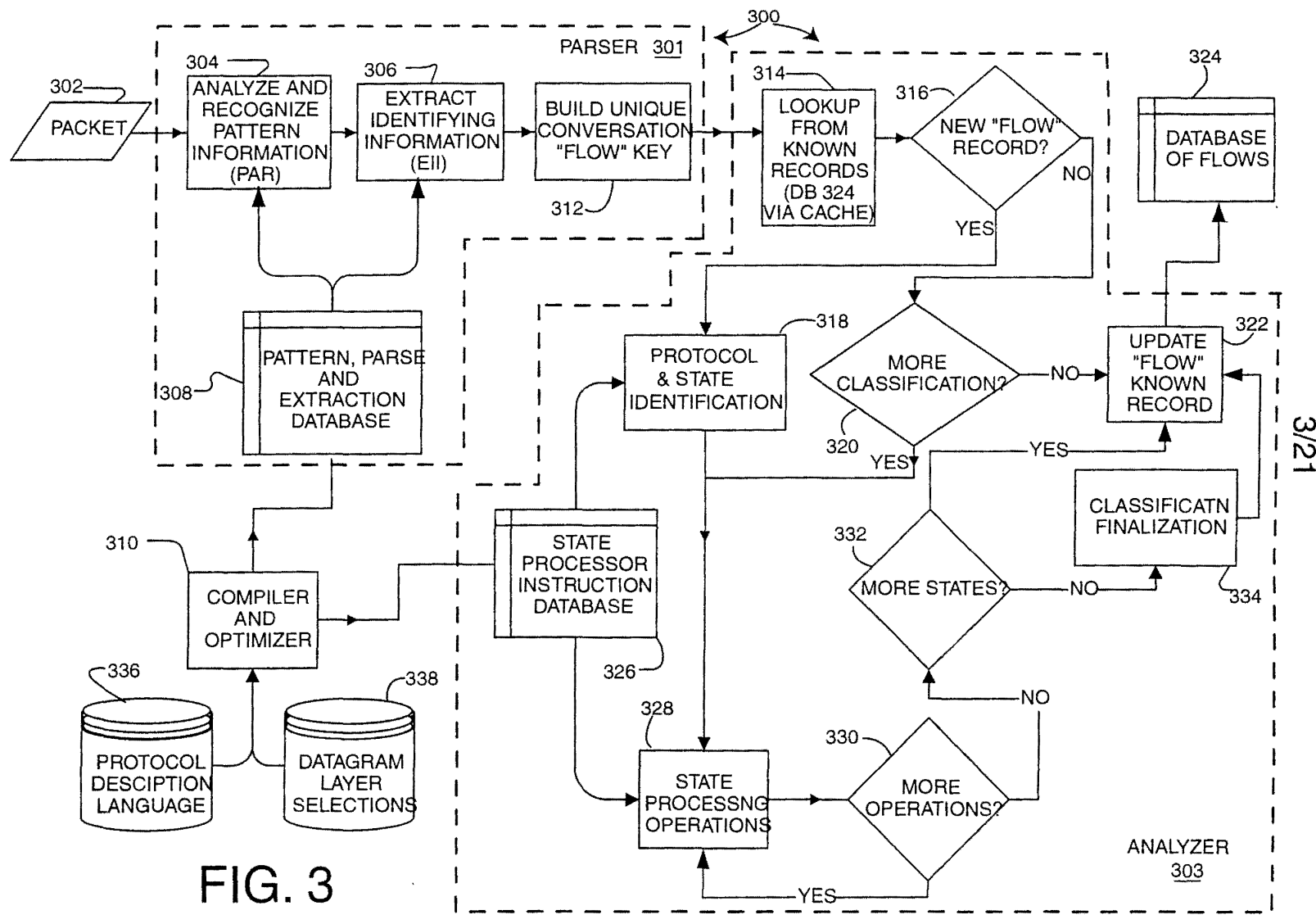


FIG. 3

4/21

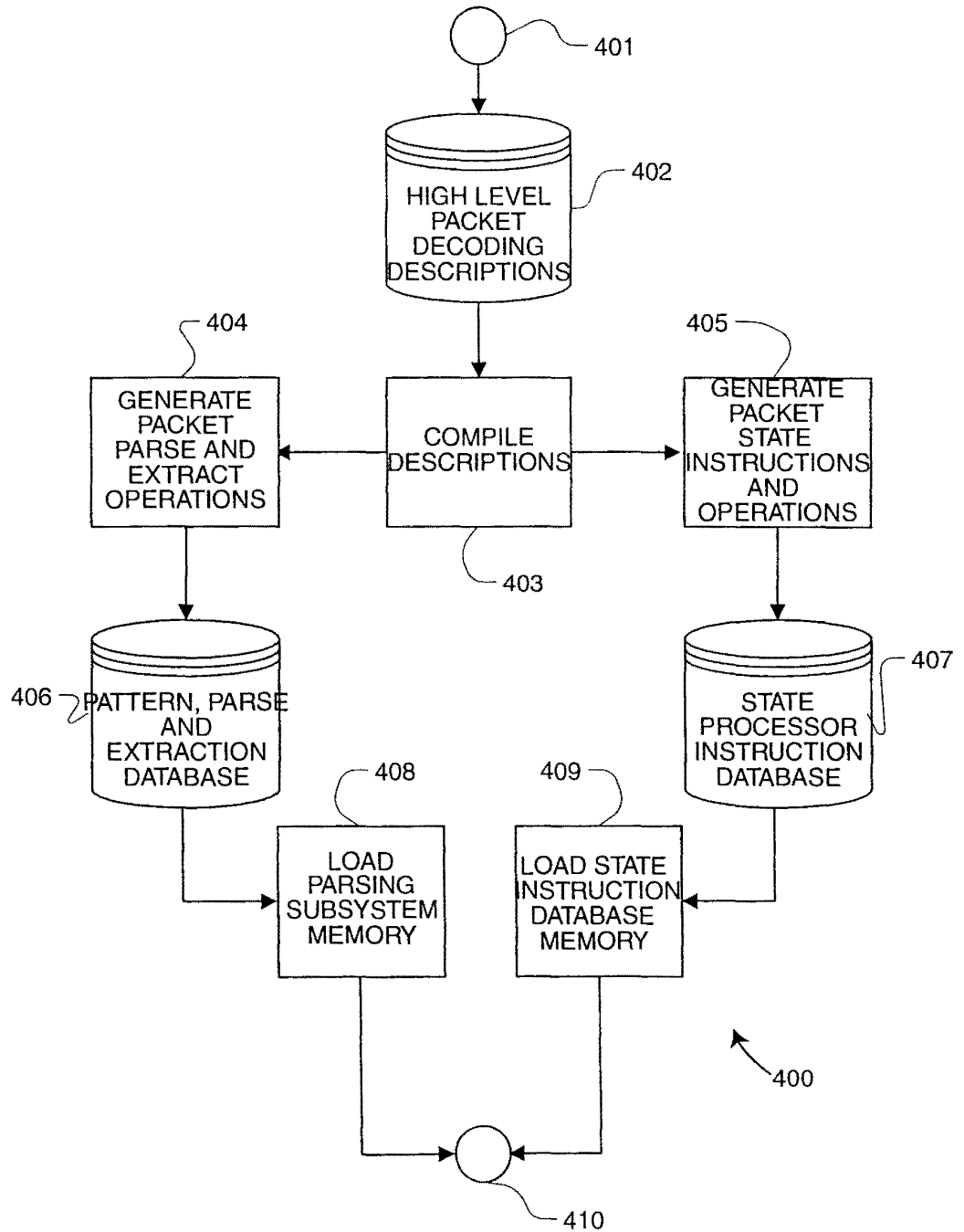


FIG. 4

5/21

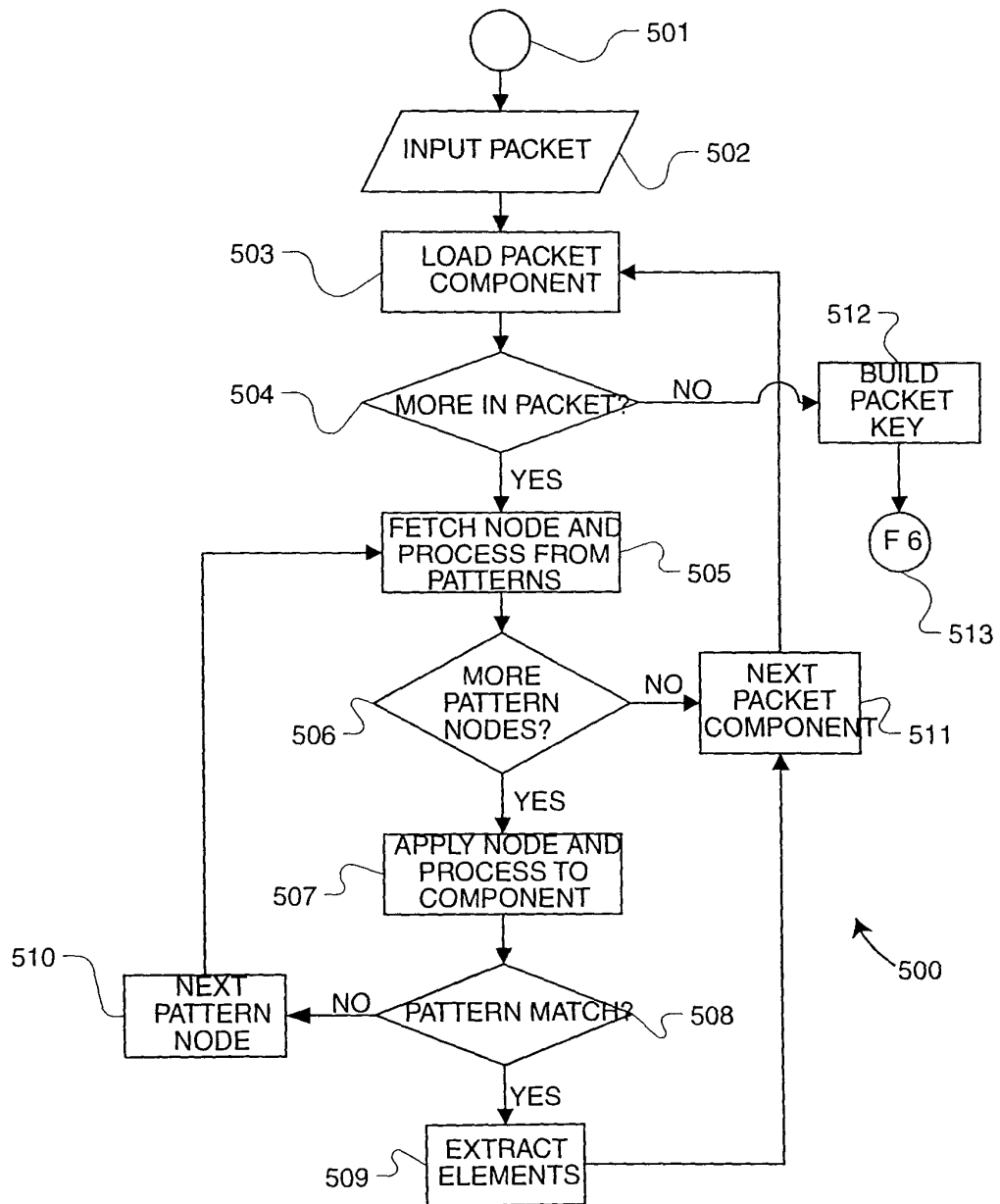


FIG. 5

6/21

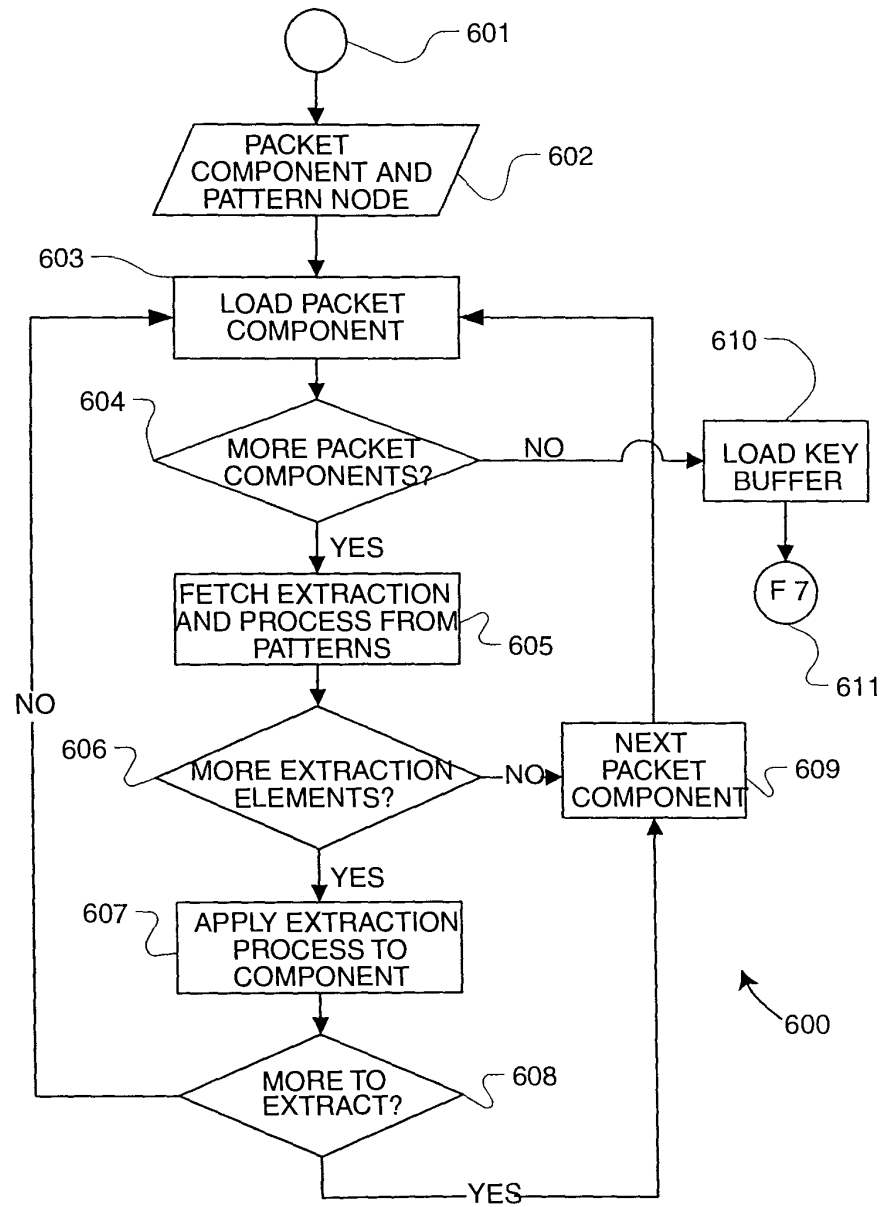


FIG. 6

7/21

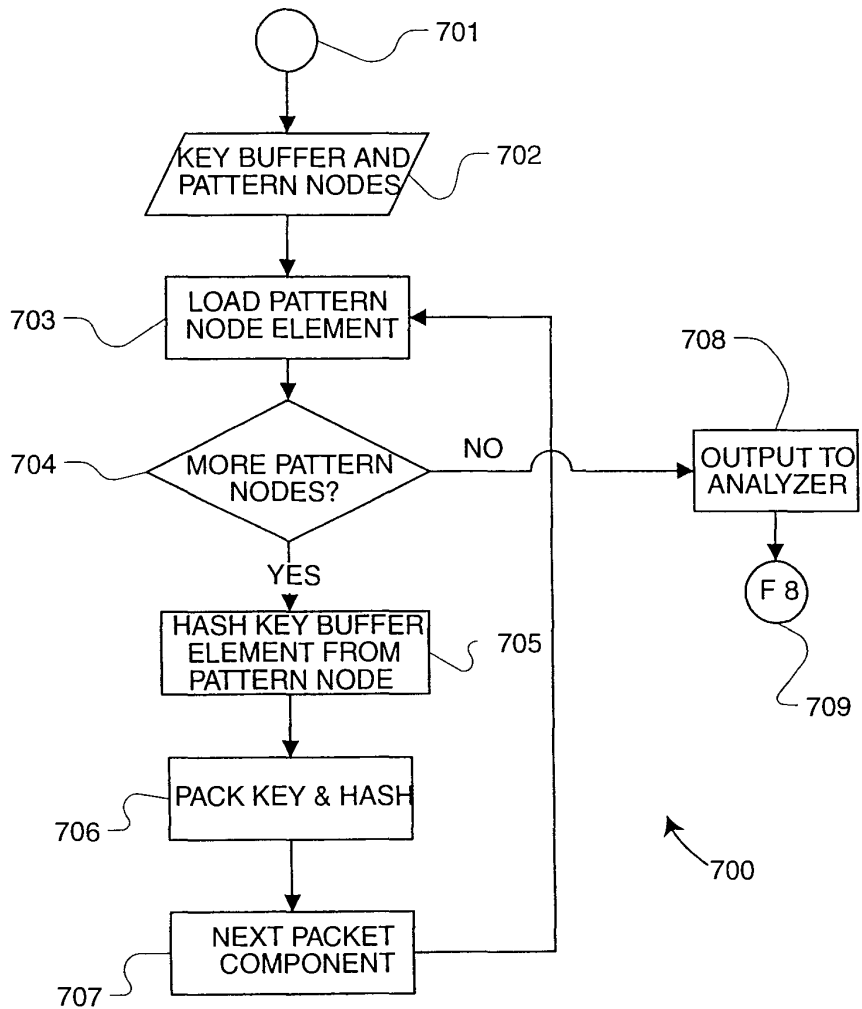


FIG. 7

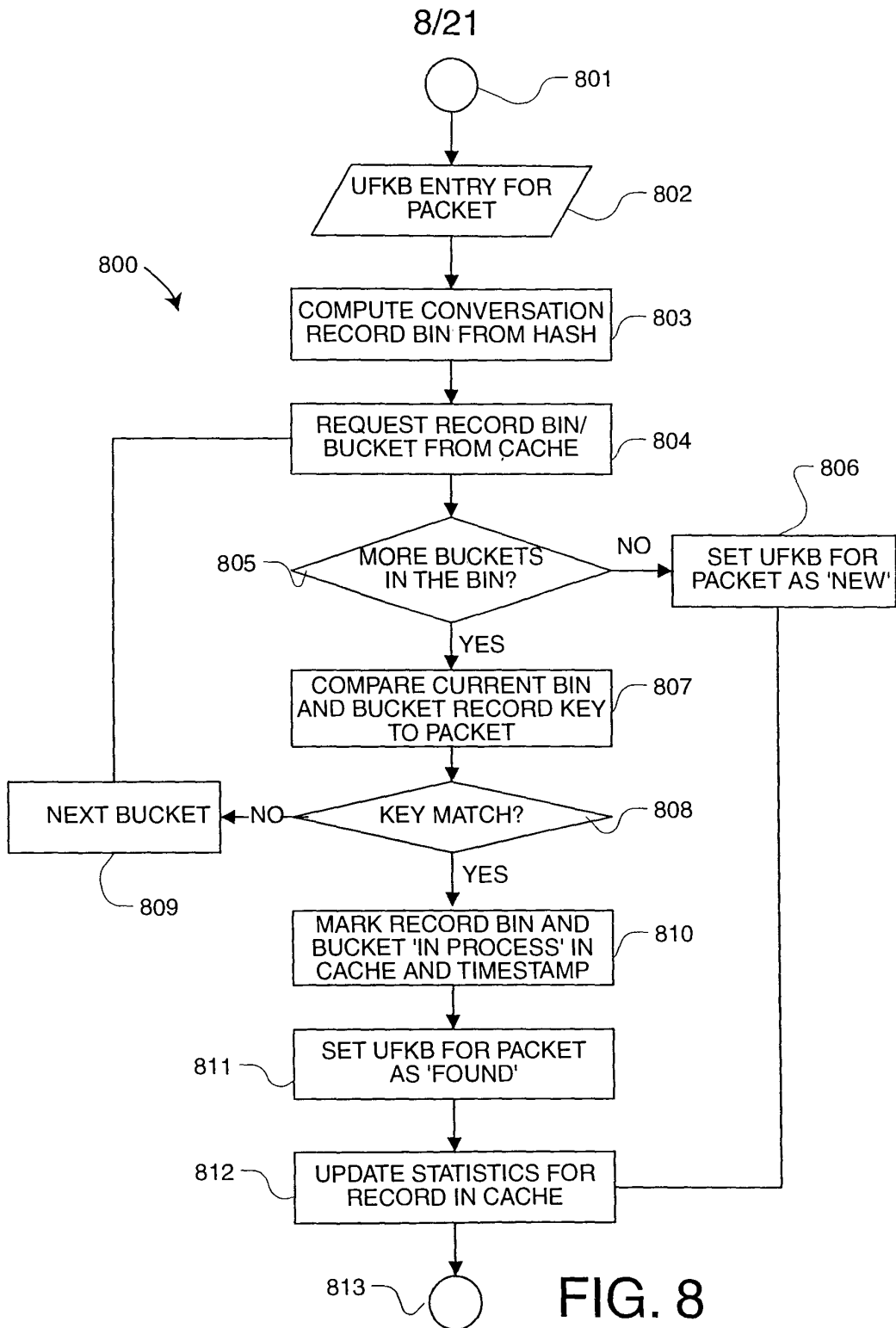


FIG. 8

9/21

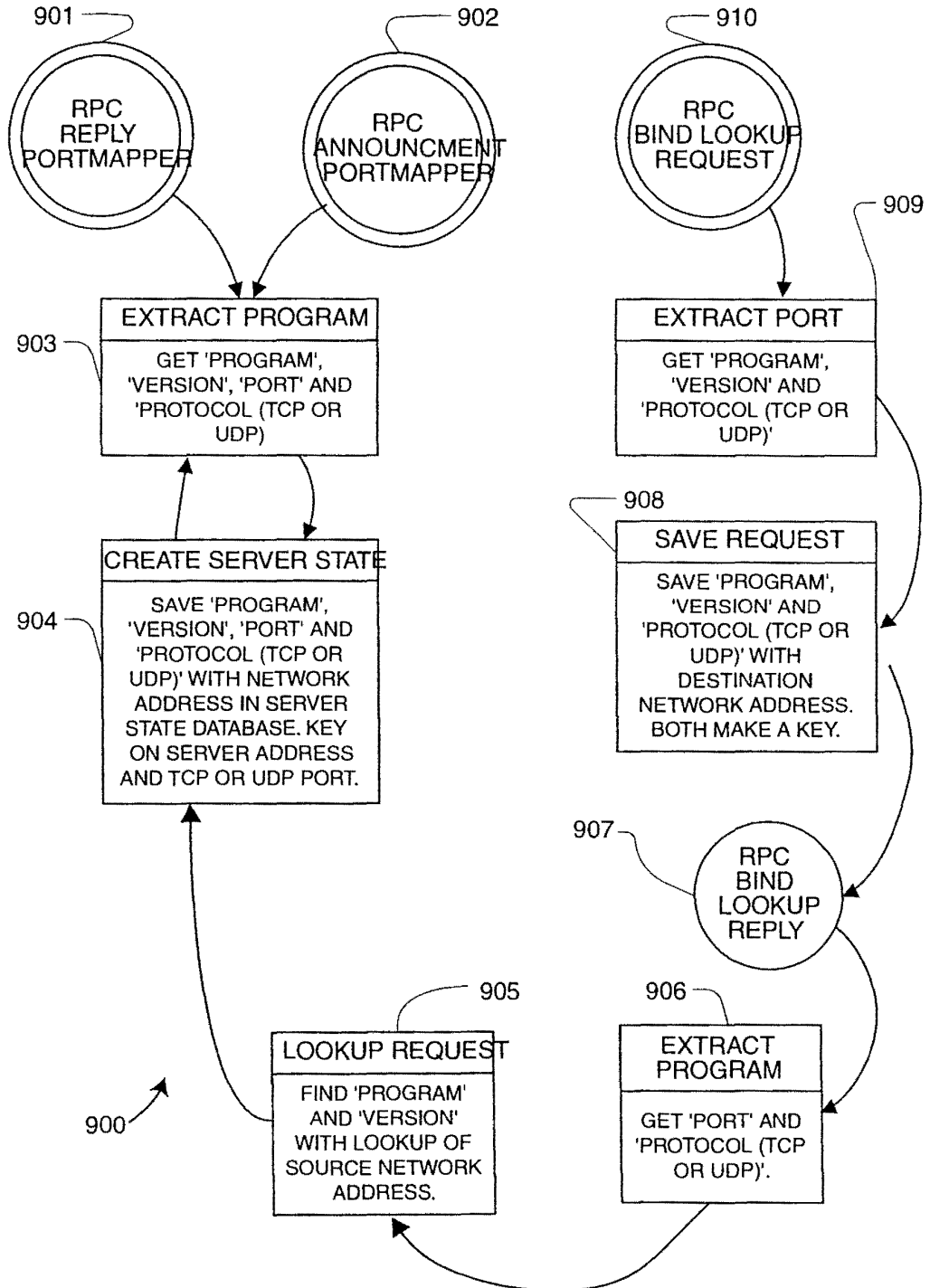


FIG. 9

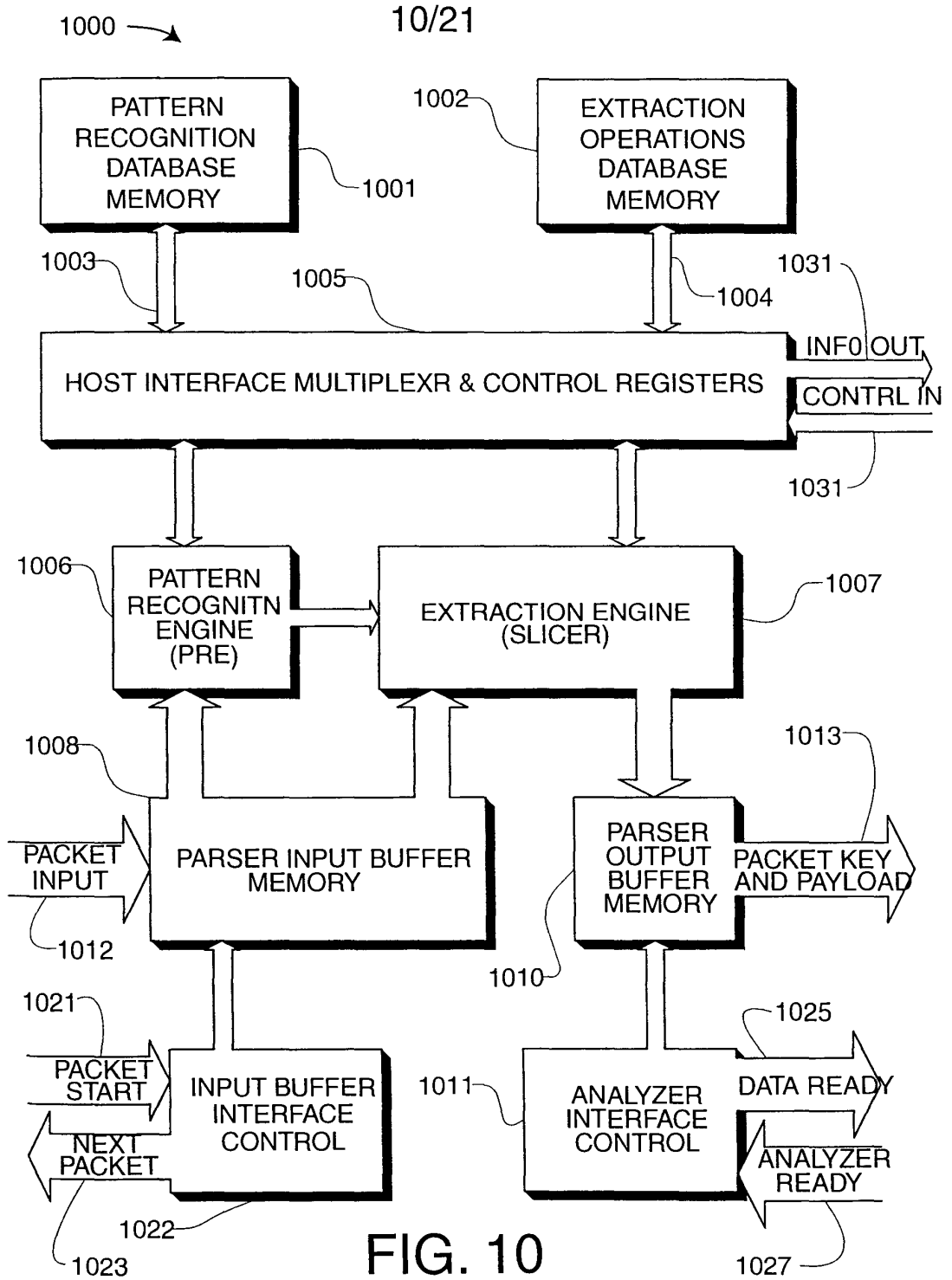


FIG. 10

11/21

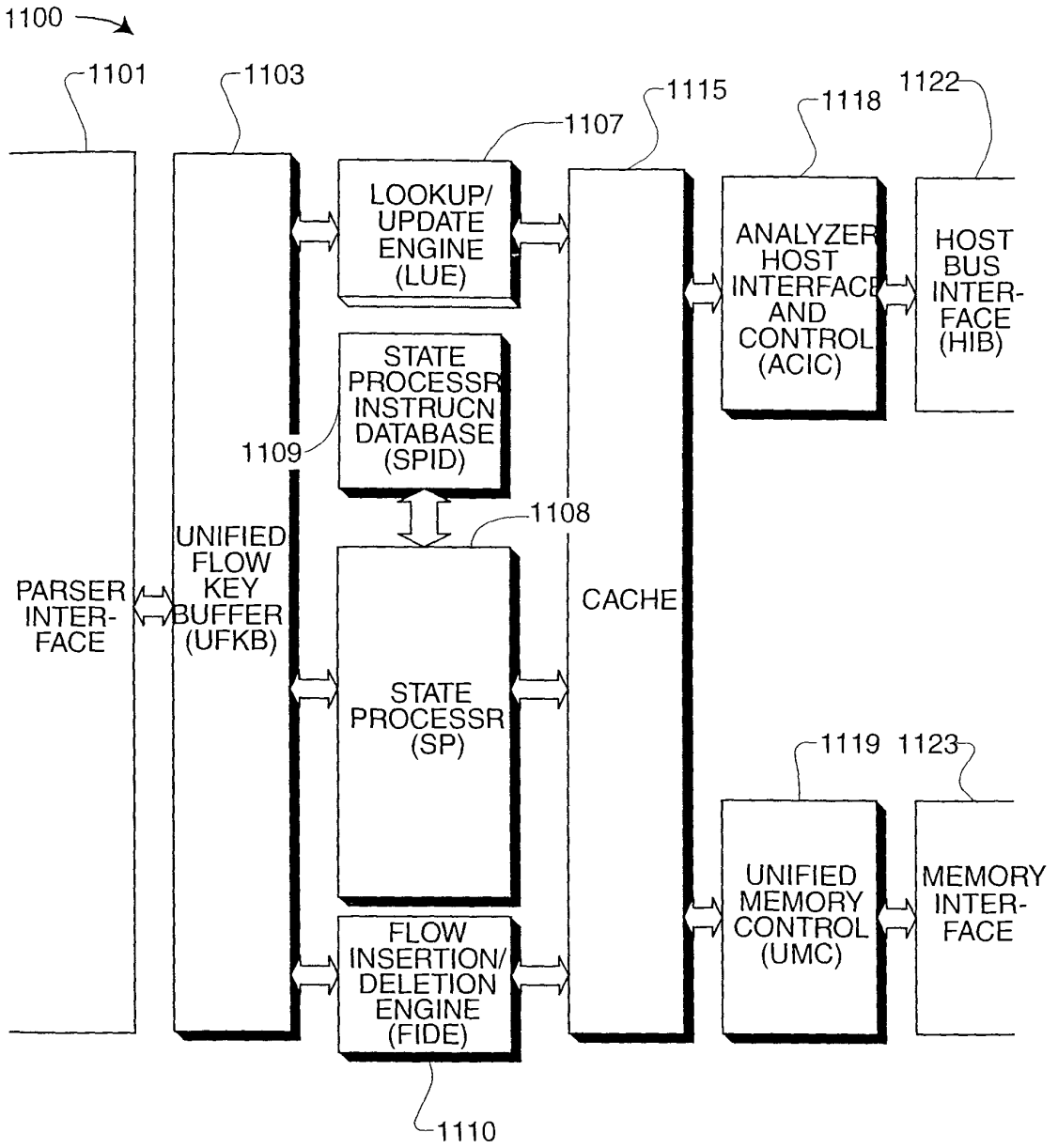


FIG. 11

12/21

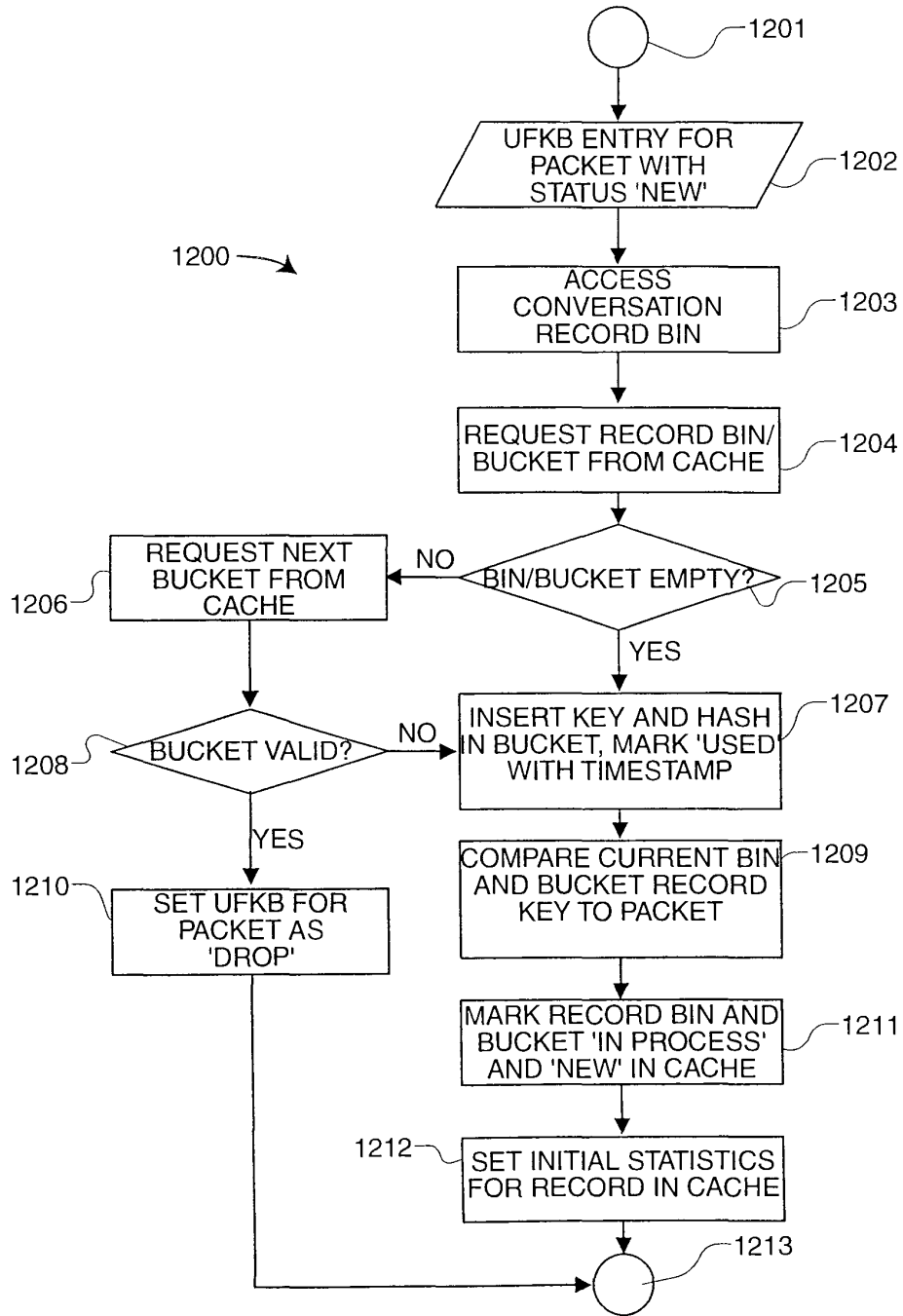


FIG. 12

13/21

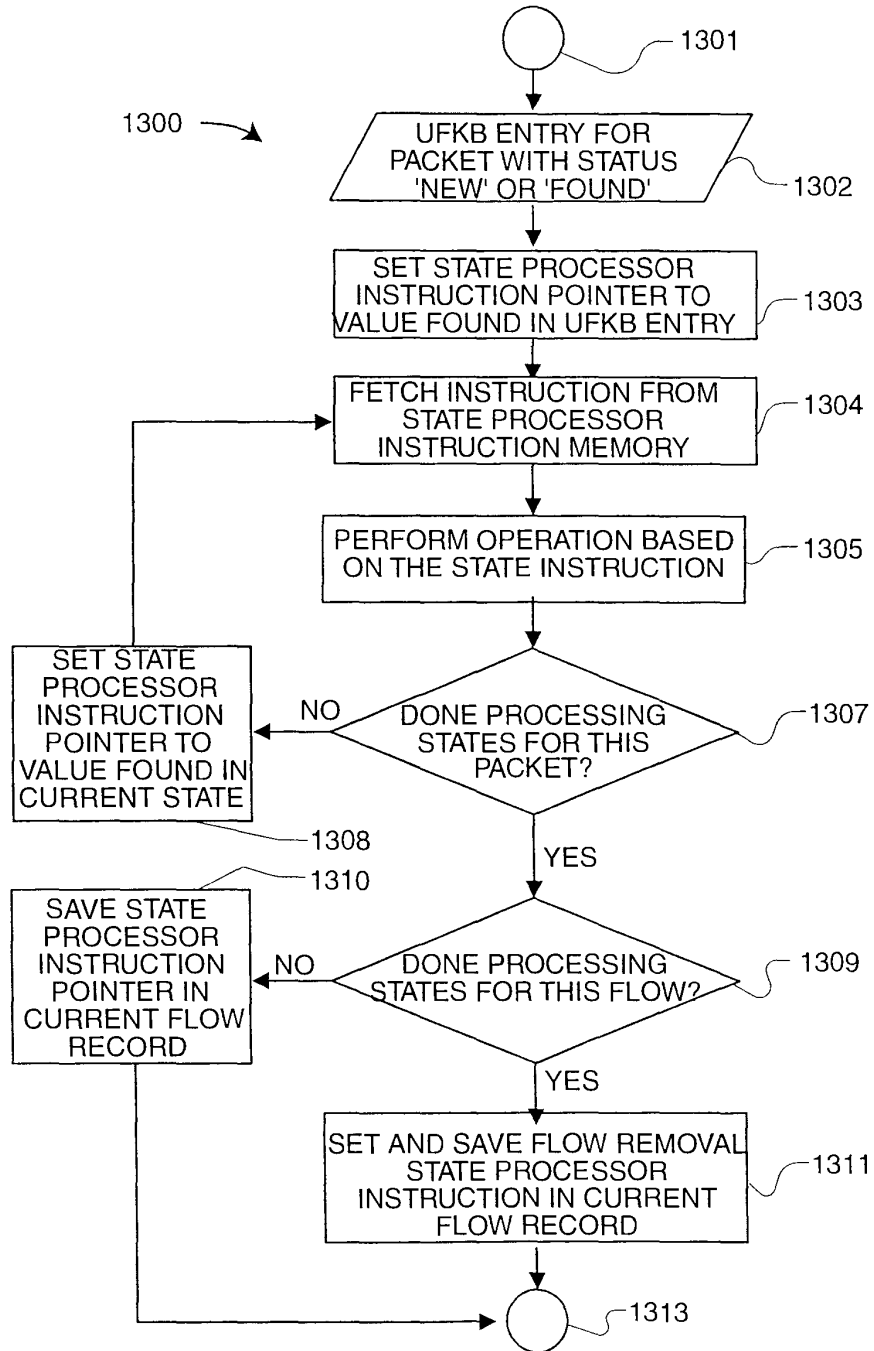


FIG. 13

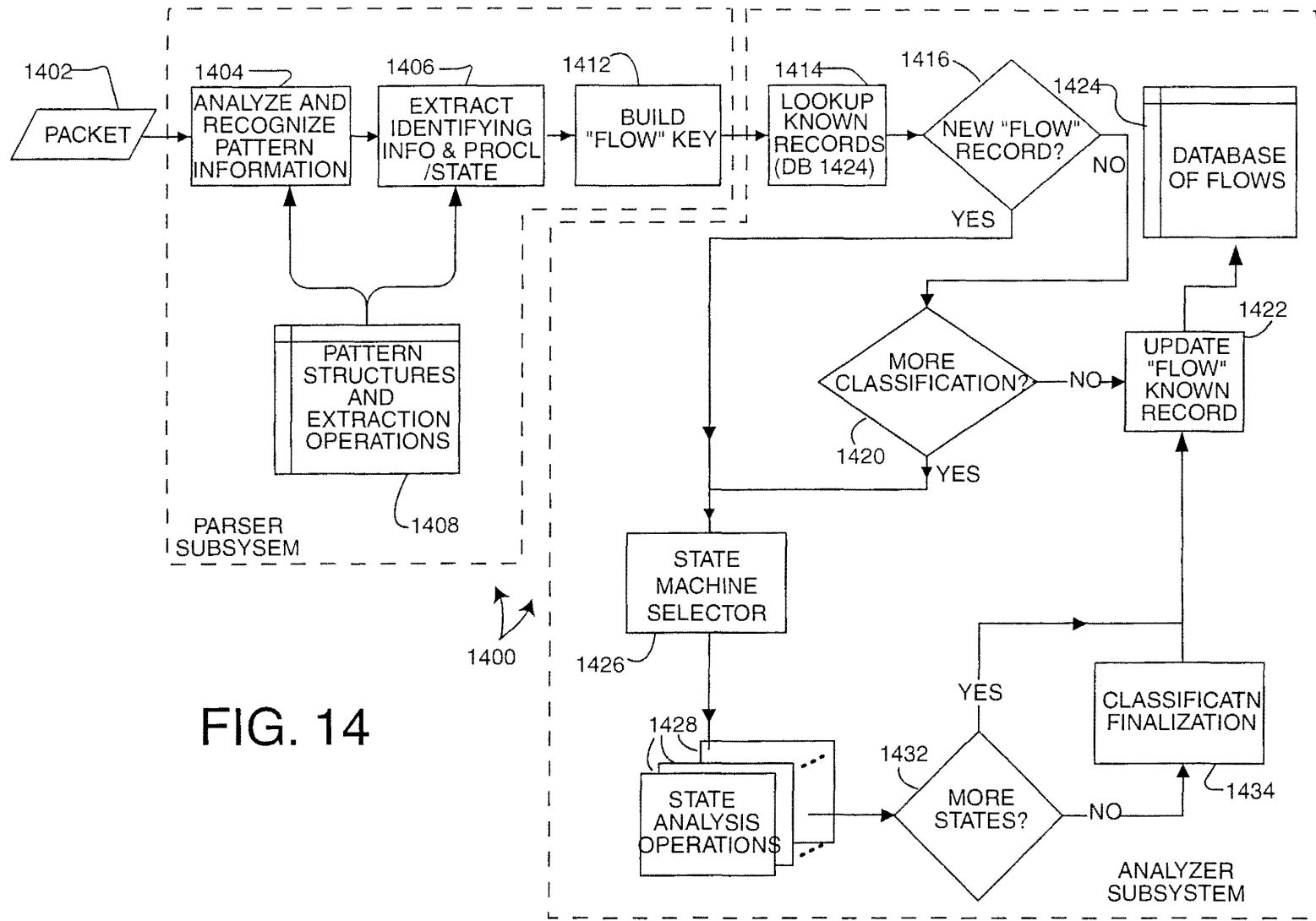


FIG. 14

14/21

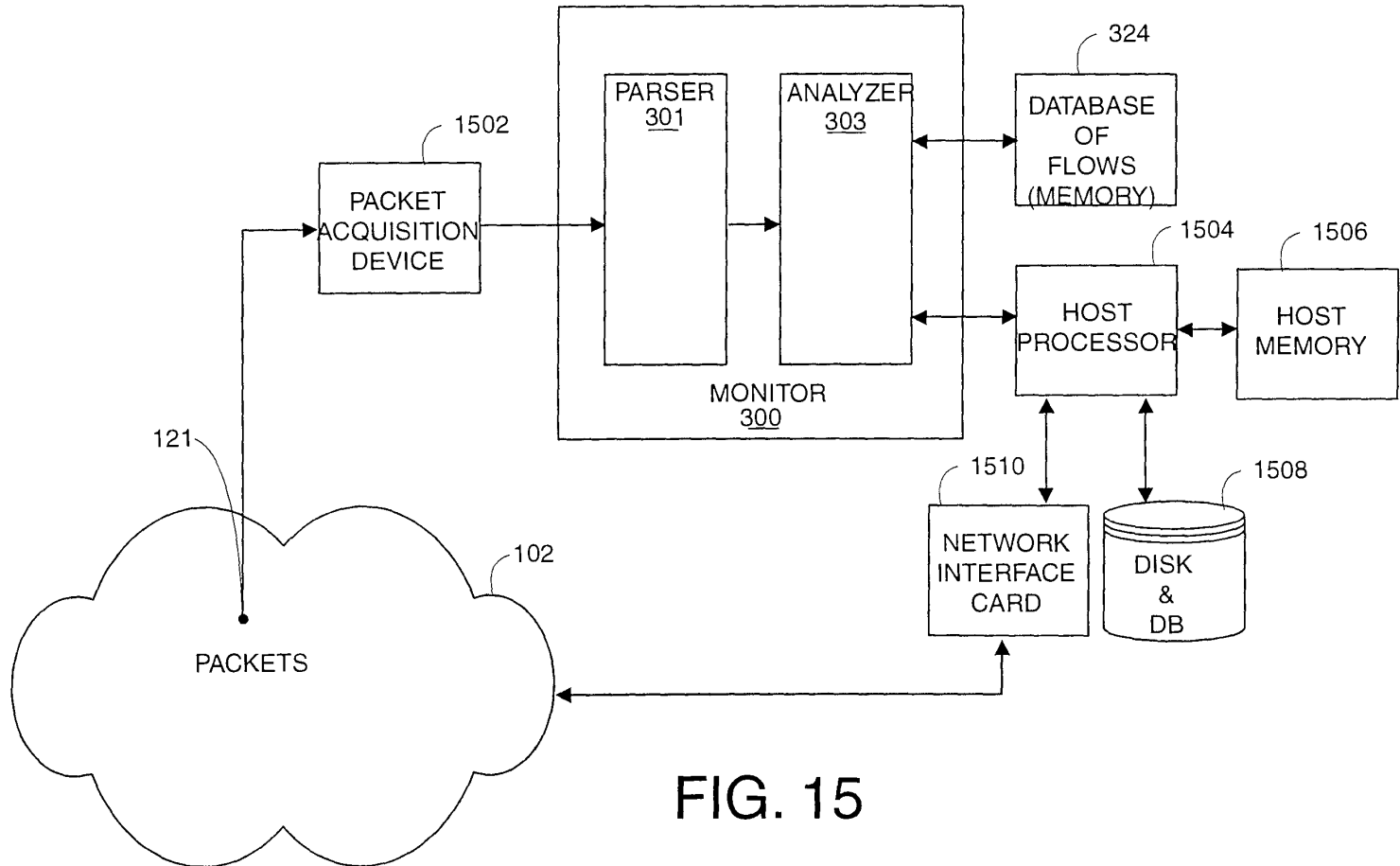


FIG. 15

15/21

16/21

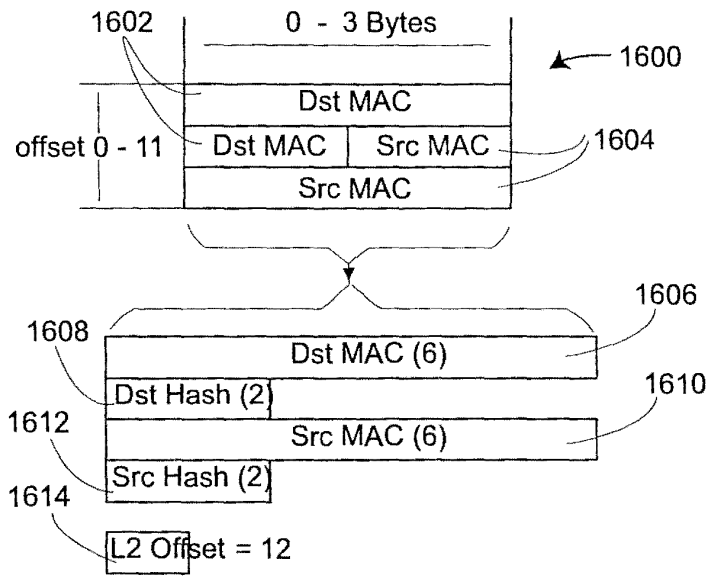


FIG. 16

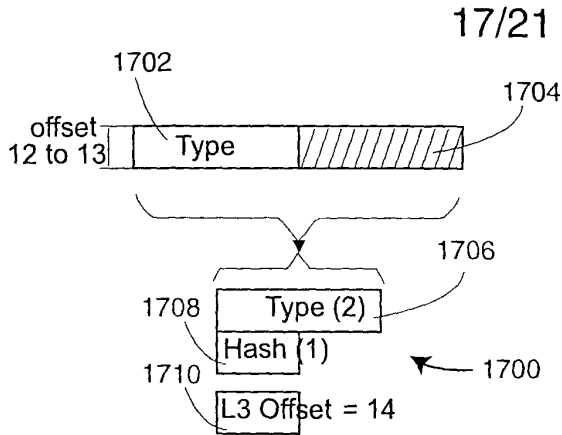


FIG. 17A

IDP	= 0x0600*
IP	= 0x0800*
CHAOSNET	= 0x0804
ARP	= 0x0806
VIP	= 0x0BAD*
VLOOP	= 0x0BAE
VECHO	= 0x0BAF
NETBIOS-3COM	= 0x3C00 -
	0x3C0D#
DEC-MOP	= 0x6001
DEC-RC	= 0x6002
DEC-DRP	= 0x6003*
DEC-LAT	= 0x6004
DEC-DIAG	= 0x6005
DEC-LAVC	= 0x6007
RARP	= 0x8035
ATALK	= 0x809B*
VLOOP	= 0x80C4
VECHO	= 0x80C5
SNA-TH	= 0x80D5*
ATALKARP	= 0x80F3
IPX	= 0x8137*
SNMP	= 0x814C#
IPv6	= 0x86DD*
LOOPBACK	= 0x9000
Apple	= 0x080007
* L3 Decoding	
# L5 Decoding	

1712

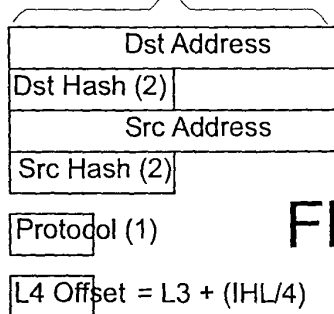
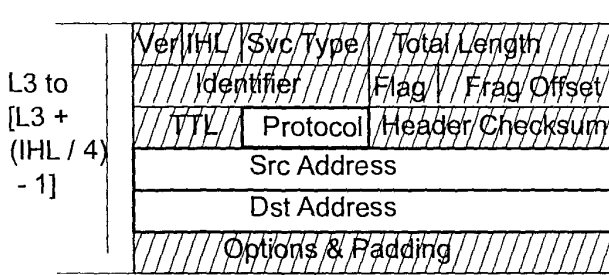


FIG. 17B

ICMP	= 1
IGMP	= 2
GGP	= 3
TCP	= 6*
EGP	= 8
IGRP	= 9
PUP	= 12
CHAOS	= 16
UDP	= 17*
IDP	= 22#
ISO-TP4	= 29
DDP	= 37#
ISO-IP	= 80
VIP	= 83#
EIGRP	= 88
OSPF	= 89
* L4 Decoding	
# L3 Re-Decoding	

18/21

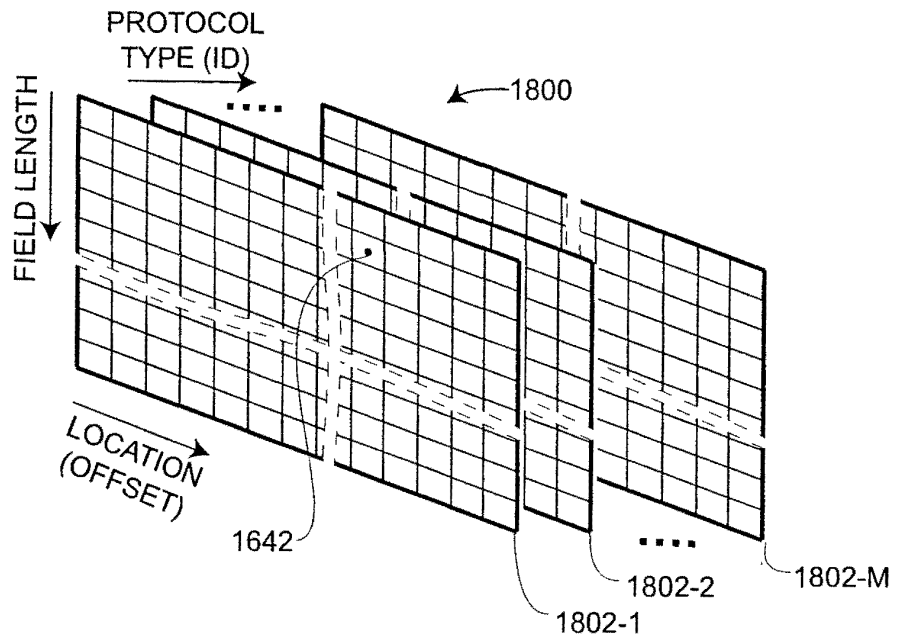


FIG. 18A

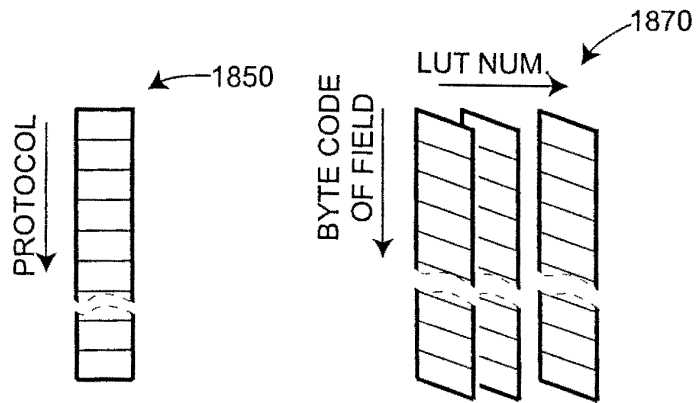
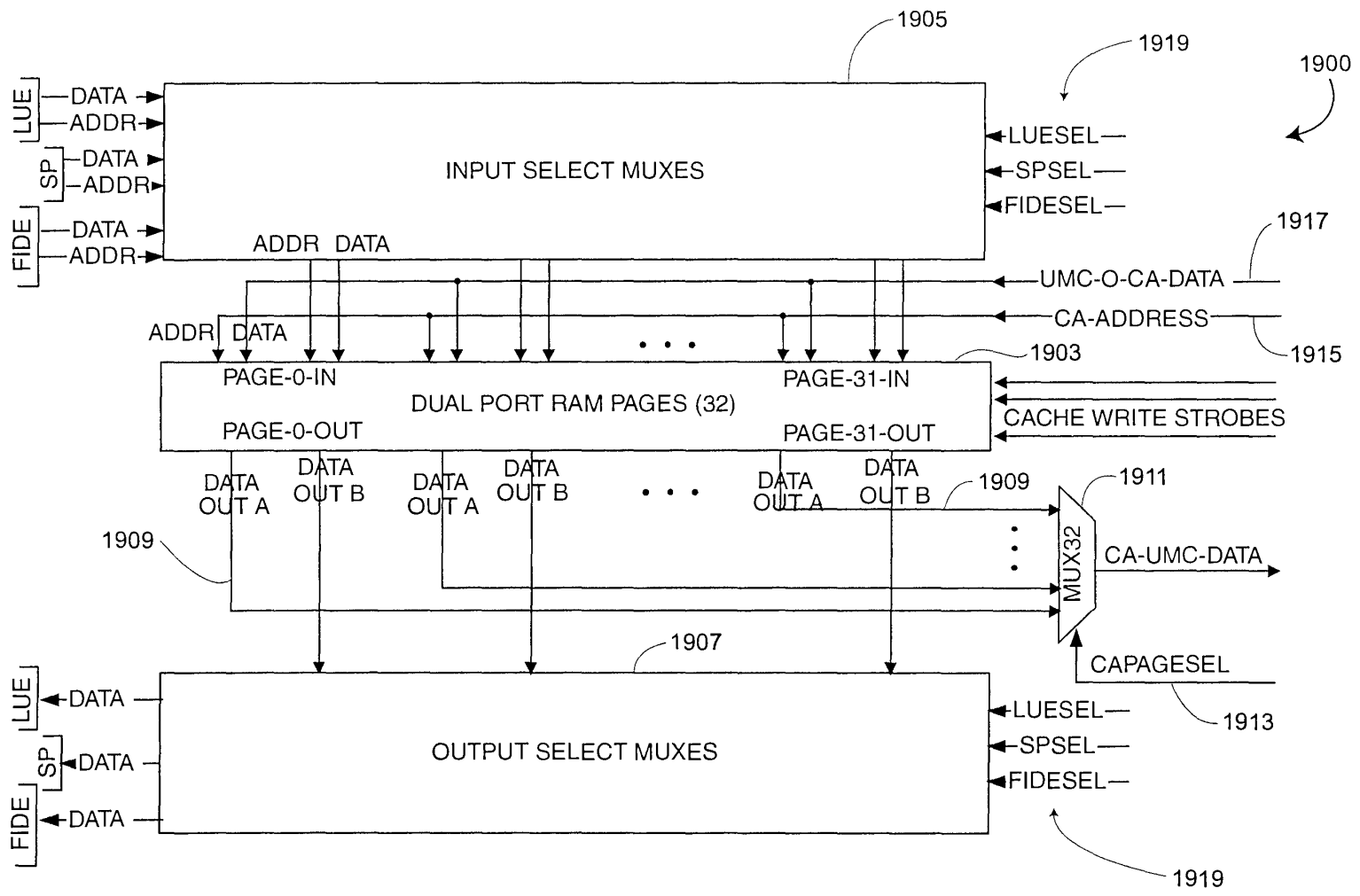


FIG. 18B



19/21

FIG. 19

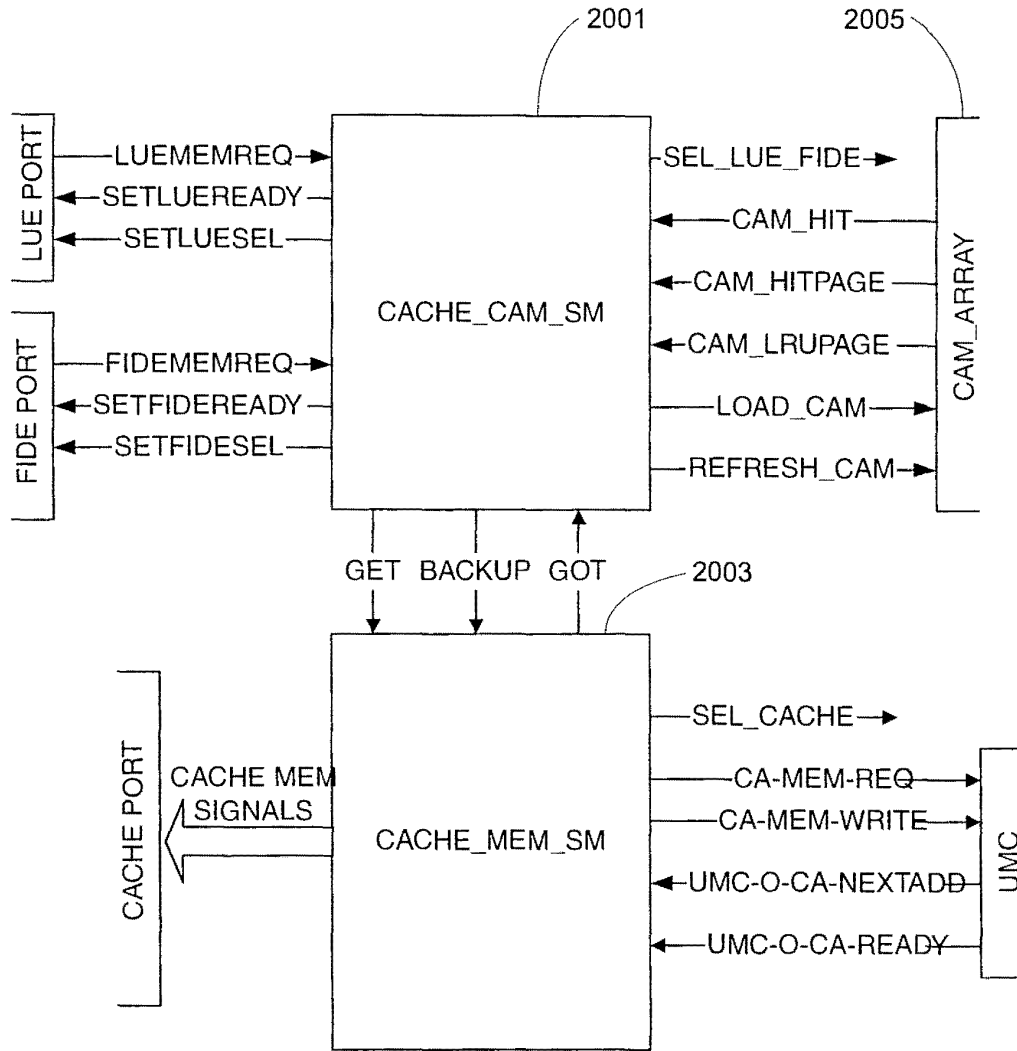


FIG. 20

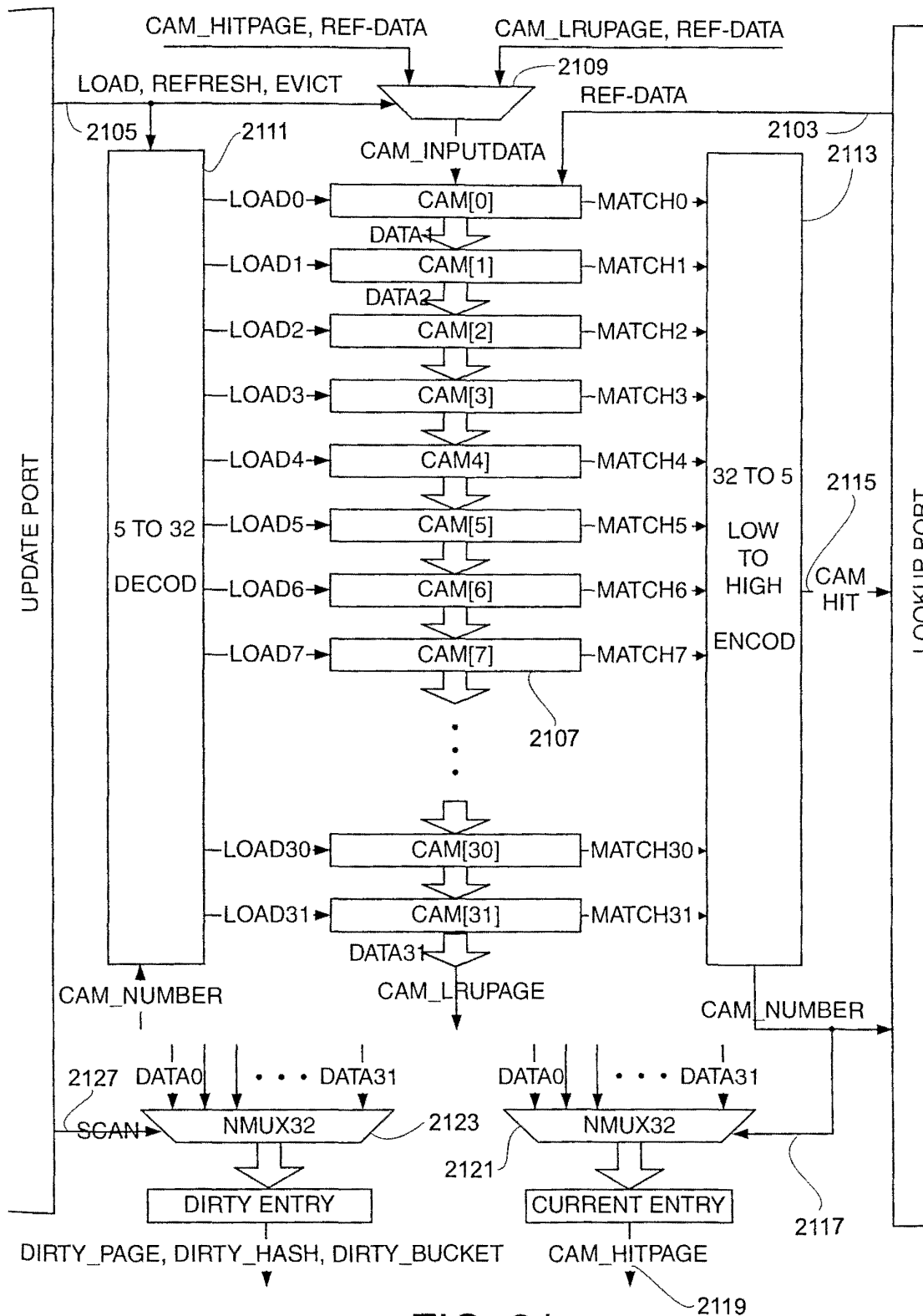


FIG. 21


UNITED STATES PATENT AND TRADEMARK OFFICE

 COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/608,266	06/30/2000	Haig A. Sarkissian	APPT-001-4

 Dov Rosenfeld
 5507 College Avenue
 Suite 2
 Oakland, CA 94618

FORMALITIES LETTER


OC000000005373402

Date Mailed: 09/05/2000

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION
FILED UNDER 37 CFR 1.53(b)
Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The statutory basic filing fee is missing.
Applicant must submit \$ 690 to complete the basic filing fee and/or file a small entity statement claiming such status (37 CFR 1.27).
- The oath or declaration is missing.
A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of \$130 for a non-small entity, must be submitted with the missing items identified in this letter

- **The balance due by applicant is \$ 820.**

*A copy of this notice **MUST** be returned with the reply.*

 Customer Service Center
 Initial Patent Examination Division (703) 308-1202
 PART 3 - OFFICE COPY



Office Ref./Docket No: APPT-091-4

Patent

113

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

<p>Applicant(s): Sarkissian, <i>et al.</i></p> <p>Application No.: 09/608266</p> <p>Filed: June 30, 2000</p> <p>Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR</p>	<p>Group Art Unit: 2731</p> <p>Examiner: (Unassigned)</p>
---	---

RESPONSE TO NOTICE TO FILE MISSING PARTS OF APPLICATION

Assistant Commissioner for Patents
Washington, D.C. 20231
Attn: Box Missing Parts

Dear Assistant Commissioner:

This is in response to a Notice to File Missing Parts of Application under 37 CFR 1.53(f). Enclosed is a copy of said Notice and the following documents and fees to complete the filing requirements of the above-identified application:

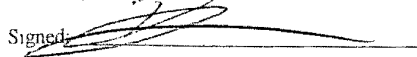
- Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration;
- Executed Assignment with assignment cover sheet.
- A credit card payment form in the amount of \$ 860.00 is attached, being for:
 - Statutory basic filing fee: \$ 690
 - Additional claim fee of \$ 0
 - Assignment recordation fee of \$ 40
 - Missing Parts Surcharge \$ 130

Applicant(s) believe(s) that no Extension of Time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition for an extension of time.

_____ Applicant(s) hereby petition(s) for an Extension of Time under 37 CFR 1.136(a) of:

- _____ one months (\$110) _____ two months (\$380)
- _____ two months (\$870) _____ four months (\$1360)

If an additional extension of time is required, please consider this as a petition therefor.

Certificate of Mailing under 37 CFR 1.8	
I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231 on	
Date: <u>Oct 20, 2008</u>	Signed: 
Name: Dov Rosenfeld, Reg. No 38687	

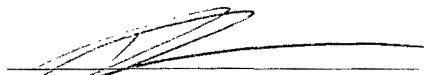
Application 09/608266, Page 2

X The Commissioner is hereby authorized to charge payment of any missing fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292

(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

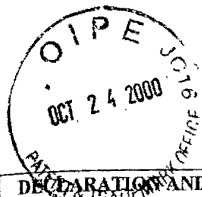
Respectfully Submitted,

Oct 20, 2008
Date


Dov Rosenfeld, Reg. No. 38687

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 653-7992



113

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. APPT-001-4

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

the specification of which is attached hereto unless the following box is checked:

(X) was filed on June 30, 2000 as US Application Serial No 09/608266 or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

Table with 4 columns: COUNTRY, APPLICATION NUMBER, DATE FILED, PRIORITY CLAIMED UNDER 35 (YES/NO).

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

Table with 2 columns: APPLICATION SERIAL NUMBER, FILING DATE.

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Table with 3 columns: APPLICATION SERIAL NUMBER, FILING DATE, STATUS(patented/pending/abandoned).

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Dov Rosenfeld, Reg No 38,687

Table with 2 columns: Send Correspondence to, Direct Telephone Calls To.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of First Inventor: Haig A. Sarkissian

Citizenship: USA

Residence: 8701 Mountain Top, San Antonio, Texas 78255

Post Office Address: Same

Handwritten signature: Haig A. Sarkissian

Handwritten date: Sept 21, 2000

First Inventor's Signature

Date

Declaration and Power of Attorney (Continued)

Case No; «Case CaseNumber»

Page 2

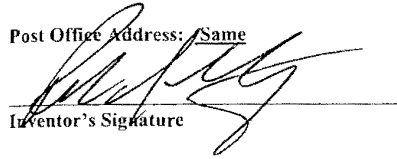
ADDITIONAL INVENTOR SIGNATURES:

Name of Second Inventor: Russell S. Dietz

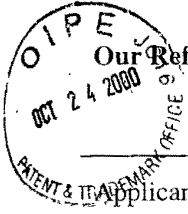
Citizenship: USA

Residence: 6146 Ostenberg Drive, San Jose, CA 95120-2736

Post Office Address: Same


Inventor's Signature

10/7/00
Date



Our Ref./Docket No: APP1-001-4

Patent

113

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.*

Group Art Unit: 2731

Application No.: 09/608266

Examiner: (Unassigned)

Filed: June 30, 2000

Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

REQUEST FOR RECORDATION OF ASSIGNMENT

Assistant Commissioner for Patents
Washington, D.C. 20231
Attn: Box Assignment

Dear Assistant Commissioner:

Enclosed herewith for recordation in the records of the U.S. Patent and Trademark Office is an original Assignment, an Assignment Cover Sheet, and \$40.00. Please record and return the Assignment.

Respectfully Submitted,

Oct 20, 2000
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 653-7992

Certificate of Mailing under 37 CFR 1.8	
I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231 on.	
Date: <u>Oct 20, 2000</u>	Signed:
	Name: Dov Rosenfeld, Reg. No. 38687



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

113

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/608,266	06/30/2000	Haig A. Sarkissian	APPT-001-4

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

FORMALITIES LETTER



OC000000005373402

Date Mailed: 09/05/2000

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1 136(a).

- The statutory basic filing fee is missing.
Applicant must submit \$ 690 to complete the basic filing fee and/or file a small entity statement claiming such status (37 CFR 1.27).
- The oath or declaration is missing
A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1 16(e) of \$130 for a non-small entity, must be submitted with the missing items identified in this letter.

- The balance due by applicant is \$ 820.

A copy of this notice MUST be returned with the reply.

Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 2 - COPY TO BE RETURNED WITH RESPONSE

SEARCHED

SERIALIZED

INDEXED

FILED

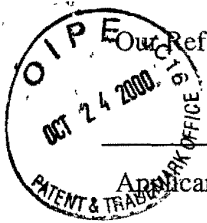
OCT 11 2000

PATENT & TRADEMARK OFFICE

WASHINGTON, DC

9/1/00

file://C:\APPS\PreExam\correspondence\2_B.xml



Our Ref./Docket No: APPT-6-1-4

Sector/#
Patent
#3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, et al.
Application No.: 09/608266
Filed: June 30, 2000
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

Group Art Unit: 2731
Examiner: (Unassigned)

RESPONSE TO NOTICE TO FILE MISSING PARTS OF APPLICATION

Assistant Commissioner for Patents
Washington, D.C. 20231
Attn: Box Missing Parts

Dear Assistant Commissioner:

This is in response to a Notice to File Missing Parts of Application under 37 CFR 1.53(f). Enclosed is a copy of said Notice and the following documents and fees to complete the filing requirements of the above-identified application:

- Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration;
- Executed Assignment with assignment cover sheet.
- A credit card payment form in the amount of \$ 860.00 is attached, being for:
 - Statutory basic filing fee: \$ 690
 - Additional claim fee of \$ 0
 - Assignment recordation fee of \$ 40
 - Missing Parts Surcharge \$ 130

Applicant(s) believe(s) that no Extension of Time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition for an extension of time.

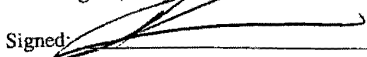
Applicant(s) hereby petition(s) for an Extension of Time under 37 CFR 1.136(a) of:

- one months (\$110) two months (\$380)
- two months (\$870) four months (\$1360)

If an additional extension of time is required, please consider this as a petition therefor.

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: Oct 20, 2000 Signed: 

Name: Dov Rosenfeld, Reg. No. 38687

Application 09/608266, Page 2

X The Commissioner is hereby authorized to charge payment of any missing fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292

(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

Oct 20, 2000
Date


Dov Rosenfeld, Reg. No. 38687

Address for correspondence:


Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 653-7992

Our Docket/Ref. No.: APPT-09-4

Patent

2664
RS

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

<p>Applicant(s): Sarkissian et al. Serial No.: 09/608266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR</p>	<p>Group Art Unit: 2731 Examiner: </p>	<p><i>#</i> <i>4</i> <i>4-12-01</i> RECEIVED APR 12 2001 TC 2600 MAIL ROOM</p>
---	--	---

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

- An Information Disclosure Statement for the above referenced patent application, together with PTO form 1449 and a copy of each reference cited in form 1449.
- Return postcard.
- The commissioner is hereby authorized to charge payment of any missing fee associated with this communication or credit any overpayment to Deposit Account 50-0292.
A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED

Respectfully submitted,

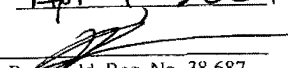
Date: Apr 9, 2001



Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:

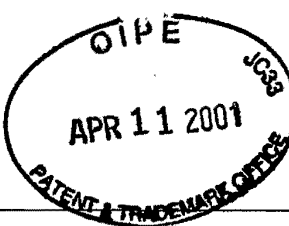
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

Certificate of Mailing under 37 CFR 1.18	
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.	
Date of Deposit:	<u>Apr 9 2001</u>
Signature:	 Dov Rosenfeld, Reg. No. 38,687

Our Docket/Ref. No.: APPT-001-4

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

<p>Applicant(s): Sarkissian et al. Serial No.: 09/608266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR</p>	<p>Group Art Unit: 2731 Examiner:  RECEIVED APR 12 2001 TTC 2600 HALLROOM</p>
---	---

Commissioner for Patents
Washington, D.C. 20231

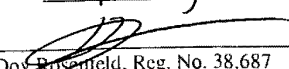
INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

This Information Disclosure Statement is submitted:

- under 37 CFR 1.97(b), or
(Within three months of filing national application; or date of entry of international application; or before mailing date of first office action on the merits; whichever occurs last)
- under 37 CFR 1.97(c) together with either a:
 - Certification under 37 CFR 1.97(e), or
 - a \$180.00 fee under 37 CFR 1.17(p)
(After the CFR 1.97(b) time period, but before final action or notice of allowance, whichever occurs first)
- under 37 CFR 1.97(d) together with a:
 - Certification under 37 CFR 1.97(e), and
 - a petition under 37 CFR 1.97(d)(2)(ii), and
 - a \$130.00 petition fee set forth in 37 CFR 1.17(i)(1).
(Filed after final action or notice of allowance, whichever occurs first, but before payment of the issue fee)

Applicant(s) submit herewith Form PTO 1449-Information Disclosure Citation together with copies, of patents, publications or other information of which applicant(s) are aware, which applicant(s) believe(s) may be material to the examination of this application and for which there may be a duty to disclose in accordance with 37 CFR 1.56.

Certificate of Mailing under 37 CFR 1.18	
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.	
Date of Deposit:	<u>Apr 9, 2001</u>
Signature:	
Dov Rosenfeld, Reg. No. 38,687	

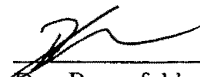
X Some of the references were cited in a search report from a foreign patent office in a counterpart foreign application. In particular, references AD, AF, AH, CI, EA, EB, EC, and ED were cited in a search report from a foreign patent office in a counterpart foreign application.

It is expressly requested that the cited information be made of record in the application and appear among the "references cited" on any patent to issue therefrom.

As provided for by 37 CFR 1.97(g) and (h), no inference should be made that the information and references cited are prior art merely because they are in this statement and no representation is being made that a search has been conducted or that this statement encompasses all the possible relevant information.

Date: Apr 9, 2001

Respectfully submitted,



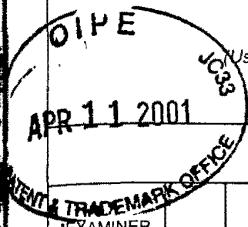
Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

INFORMATION DISCLOSURE STATEMENT

ATTY. DOCKET NO. APPT-001-4	SERIAL NO. 09/608266
APPLICANT Sarkissian et al.	
FILING DATE 6/30/2000	GROUP 2331 <i>2662</i>



(Use several sheets if necessary)

U.S. PATENT DOCUMENTS

EXAMINER INITIAL	DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
<i>AA</i>	4736320	Apr. 5, 1988	Bristol	364	300	Oct. 8, 1985
<i>AB</i>	4891639	Jan. 2, 1990	Nakamura	340	825.500	Jun. 23, 1988
<i>AC</i>	5101402	Mar. 31, 1992	Chui et al.	370	47	May 24, 1988
<i>AD</i>	5247517	Sep. 21, 1993	Ross et al.	370	85.5	Sep. 2, 1992
<i>AE</i>	5247693	Sep. 21, 1993	Bristol	395	800	Nov. 17, 1992
<i>AF</i>	5315580	May 24, 1994	Phaal	370	43	Aug. 26, 1991
<i>AG</i>	5339268	Aug. 16, 1994	Machida	365	49	Nov. 24, 1992
<i>AH</i>	5351243	Sep. 27, 1994	Kalkunte et. al.	370	92	Dec. 27, 1991
<i>AI</i>	5365514	Nov. 15, 1994	Hershey et al.	370	17	Mar. 1, 1993
<i>AJ</i>	5375070	Dec. 20, 1994	Hershey at al.	364	550	Mar. 1, 1993
<i>AK</i>	5394394	Feb. 28, 1995	Crowther et al.	370	60	Jun. 24, 1993

FOREIGN PATENT DOCUMENTS

DOCUMENT NUMBER	PUBLICATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES NO
<i>AM</i>					
<i>AN</i>					

OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

<i>AR</i>	"Technical Note: the Narus System," Downloaded April 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.
<i>AS</i>	

RECEIVED
 APR 12 2001
 TFC 2500 MAIL ROOM

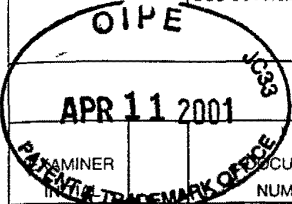
EXAMINER <i>Ala J...</i>	DATE CONSIDERED 4/2/05
-----------------------------	---------------------------

*EXAMINER: initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

INFORMATION DISCLOSURE STATEMENT

ATTY. DOCKET NO. APPT-001-4	SERIAL NO 09/608266
APPLICANT Sarkissian et al.	
FILING DATE 6/30/2000	GROUP 2731 <i>2662</i>

(Use several sheets if necessary)



U.S. PATENT DOCUMENTS

EXAMINER	DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
<i>Ar</i>	BA 5414650	May 9, 1995	Hekhuis	364	715.02	Mar. 24, 1993
<i>Ar</i>	BB 5430709	Jul. 4, 1995	Galloway	370	13	Jun. 17, 1992
<i>Ar</i>	BC 5432776	Jul. 11, 1995	Harper	370	17	Sep. 30, 1993
<i>Ar</i>	BD 5493689	Feb. 20, 1996	Waclawsky et al.	395	821	Mar. 1, 1993
<i>Ar</i>	BE 5500855	Mar. 19, 1996	Hershey et al.	370	17	Jan. 26, 1994
<i>Ar</i>	BF 5568471	Oct. 22, 1996	Hershey et al.	370	17	Sep. 6, 1995
<i>Ar</i>	BG 5574875	Nov. 12, 1996	Stansfield et al.	395	403	Mar. 12, 1993
<i>Ar</i>	BH 5586266	Dec. 17, 1996	Hershey et al.	395	200.11	Oct. 15, 1993
<i>Ar</i>	BI 5606668	Feb. 25, 1997	Shwed	395	200.11	Dec. 15, 1993
<i>Ar</i>	BJ 5608662	Mar. 4, 1997	Large et al.	364	724.01	Jan. 12, 1995
<i>Ar</i>	BK 5634009	May 27, 1997	Iddon et al.	395	200.11	Oct. 27, 1995

FOREIGN PATENT DOCUMENTS

DOCUMENT NUMBER	PUBLICATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES NO
BM					
BN					

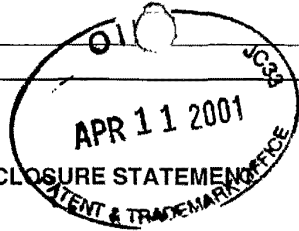
OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

BR	
BS	

RECEIVED
 APR 12 2001
 TC 2600 MAILROOM

EXAMINER <i>al. yz</i>	DATE CONSIDERED 9/2/03
---------------------------	---------------------------

*EXAMINER: initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant



INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO. APPT-001-4	SERIAL NO. 09/608266
APPLICANT Sarkissian et al.	
FILING DATE 6/30/2000	GROUP 2731 H62

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIATE
AV	CA	5651002	Jul. 22, 1997	Van Seters et al.	370	392	Jul. 12, 1995
AV	CB	5684954	Nov. 4, 1997	Kaiserswerth et al.	395	200.2	Mar. 20, 1993
AV	CC	5732213	Mar. 24, 1998	Gessel et al.	395	200.11	Mar. 22, 1996
AV	CD	5740355	Apr. 14, 1998	Watanabe et al.	395	183.21	Jun. 4, 1996
AV	CE	5761424	Jun. 2, 1998	Adams et al.	395	200.47	Dec. 29, 1995
AV	CF	5764638	Jun. 9, 1998	Ketchum	370	401	Sep. 14, 1995
AV	CG	5781735	Jul. 14, 1998	Southard	395	200.54	Sep. 4, 1997
AV	CH	5784298	Jul. 21, 1998	Hershey et al.	364	557	Jul. 11, 1996
AV	CI	5787253	Jul. 28, 1998	McCreery et al.	395	200.61	May 28, 1996
AV	CJ	5805808	Sep. 8, 1998	Hansani et al.	395	200.2	Apr. 9, 1997
AV	CK	5812529	Sep. 22, 1998	Czarnik et al.	370	245	Nov. 12, 1996

FOREIGN PATENT DOCUMENTS

	DOCUMENT NUMBER	PUBLICATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES NO
CM						
CN						

OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

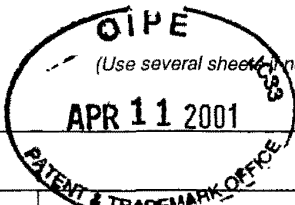
CR	
CS	

EXAMINER <i>Ala Jyr</i>	DATE CONSIDERED 9/21.03
----------------------------	----------------------------

*EXAMINER initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

RECEIVED
 APR 12 2001
 TC 2600 MAIL ROOM

INFORMATION DISCLOSURE STATEMENT



ATTY. DOCKET NO. APPT-001-4	SERIAL NO 09/608266
APPLICANT Sarkissian et al.	
FILING DATE 6/30/2000	GROUP 2131 <i>262</i>

U.S. PATENT DOCUMENTS

EXAMINER INITIAL	DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
<i>AR</i>	DA 5819028	Oct. 6, 1998	Manghirmalani et al.	395	185.1	Apr. 16, 1997
<i>AN</i>	DB 5825774	Oct. 20, 1998	Ready et al.	370	401	Jul. 12, 1995
<i>AV</i>	DC 5835726	Nov. 10, 1998	Shwed et al.	395	200.59	Jun. 17, 1996
<i>AR</i>	DD 5838919	Nov. 17, 1998	Schwaller et al.	395	200.54	Sep. 10, 1996
<i>AN</i>	DE 5841895	Nov. 24, 1998	Huffman	382	155	Oct. 25, 1996
<i>AV</i>	DF 5850386	Dec. 15, 1998	Anderson et al.	370	241	Nov. 1, 1996
<i>AN</i>	DG 5850388	Dec. 15, 1998	Anderson et al.	370	252	Oct. 31, 1996
<i>AR</i>	DH 5862335	Jan. 19, 1999	Welch, Jr. et al.	395	200.54	Apr. 1, 1993
<i>AN</i>	DI 5878420	Mar. 2, 1999	de la Salle	707	10	Oct. 29, 1997
<i>AR</i>	DJ 5893155	Apr. 6, 1999	Cheriton	711	144	Dec. 3, 1996
<i>AN</i>	DK 5903754	May 11, 1999	Pearson	395	680	Nov. 14, 1997

FOREIGN PATENT DOCUMENTS

DOCUMENT NUMBER	PUBLICATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES NO
<i>DM</i>					
<i>DN</i>					

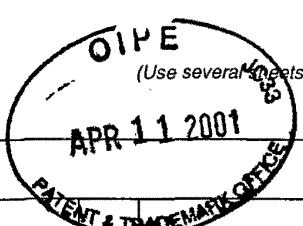
OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

<i>DR</i>	
<i>DS</i>	

RECEIVED
 APR 2 2001
 TC 2600 MAILROOM

EXAMINER <i>Ali G</i>	DATE CONSIDERED <i>9/2/03</i>
--------------------------	----------------------------------

*EXAMINER initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

INFORMATION DISCLOSURE STATEMENT 	ATTY. DOCKET NO. APPT-001-4	SERIAL NO. 09/608266
	APPLICANT Sarkissian et al.	
	FILING DATE 6/30/2000	GROUP 2731 <i>[Signature]</i>

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL	DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIATE
<i>AV</i>	EA 5917821	Jun. 29, 1999	Gobuyan et al.	370	392	Aug. 16, 1996
<i>AV</i>	EB 5414704	May 9, 1995	Spinney	370	60	Apr. 5, 1994
<i>AV</i>	EC 6014380	Jan 11, 2000	Hendel et al.	370	392	Jun. 30, 1997
<i>AV</i>	ED 5511215	Apr. 23, 1996	Terasaka et al.	395	800	Oct. 26, 1993
	EE					
	EF					
	EG					
	EH					
	EI					
	EJ					
	EK					

FOREIGN PATENT DOCUMENTS

DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES NO
DM					
DN					

OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

DR	
DS	

EXAMINER <i>Ale Jr</i>	DATE CONSIDERED 9/2/03
---------------------------	---------------------------

*EXAMINER: initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

1105



US005917821A

United States Patent [19]
Gobuyan et al.

[11] **Patent Number:** **5,917,821**
[45] **Date of Patent:** **Jun. 29, 1999**

[54] **LOOK-UP ENGINE FOR PACKET-BASED NETWORK**

[75] **Inventors:** Jerome Gobuyan, Kanata; Wayne Burwell, Ottawa; Nutan Behki, Nepean, all of Canada

[73] **Assignee:** Newbridge Networks Corporation, Kanata, Canada

[21] **Appl. No.:** 08/663,263

[22] **PCT Filed:** Dec. 21, 1994

[86] **PCT No.:** PCT/CA94/00695

§ 371 Date: Aug. 16, 1996

§ 102(e) Date: Aug. 16, 1996

[87] **PCT Pub. No.:** WO95/18497

PCT Pub. Date: Jul. 6, 1995

[30] **Foreign Application Priority Data**

Dec. 24, 1993 [GB] United Kingdom 9326476

[51] **Int. Cl.⁶** H04L 12/46

[52] **U.S. Cl.** 370/392; 370/401

[58] **Field of Search** 370/392, 395, 370/400, 401-405, 465, 466, 351, 389, 396, 397, 474; 395/200.68

[56] **References Cited**

U.S. PATENT DOCUMENTS

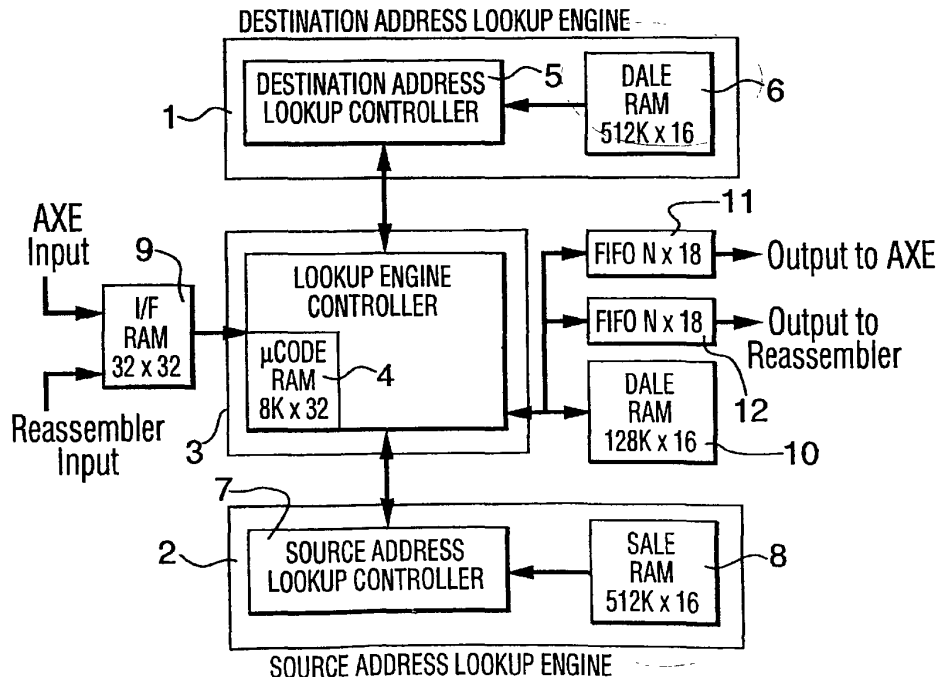
5,095,480 3/1992 Fenner 370/238
5,463,777 10/1995 Bialkowski et al. 370/256

Primary Examiner—Chau Nguyen
Attorney, Agent, or Firm—Marks & Clerk

[57] **ABSTRACT**

An arrangement is disclosed for parsing packets in a packet-based data transmission network. The packets include packet headers divided into fields having values representing information pertaining to the packet. The arrangement includes an input receiving fields from the packet headers of incoming packets, a memory for storing information related to possible values of said fields, and a device for retrieving the stored information appropriate to a received field value. The retrieving device comprises a look-up engine including at least one memory organized in a hierarchical tree structure, and a controller for controlling the operation of the memory. The arrangement is capable of performing fast look-up operations at a low cost of implementation.

29 Claims, 11 Drawing Sheets



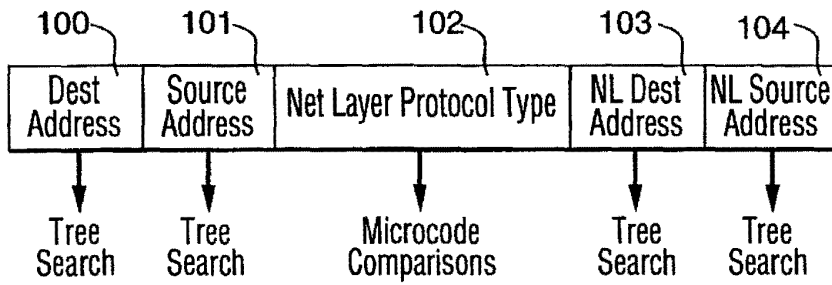


FIG. 1

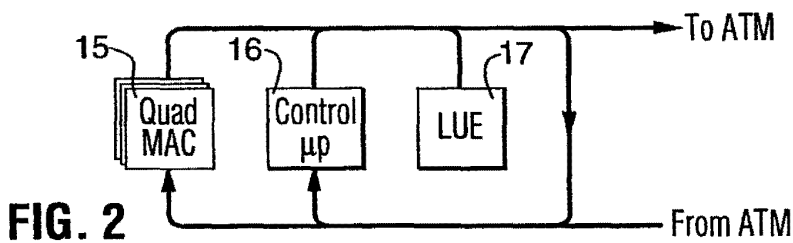


FIG. 2

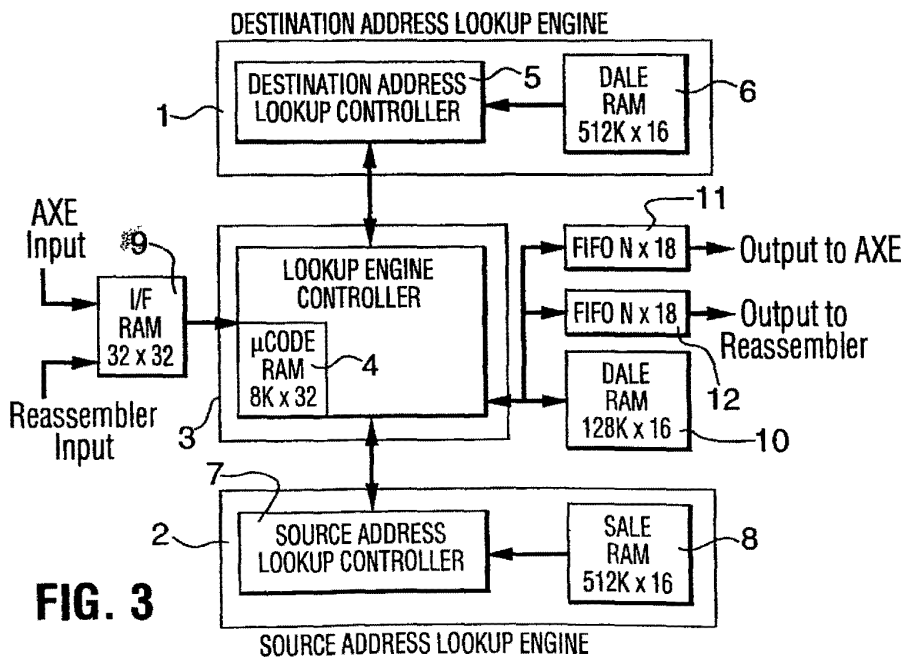


FIG. 3

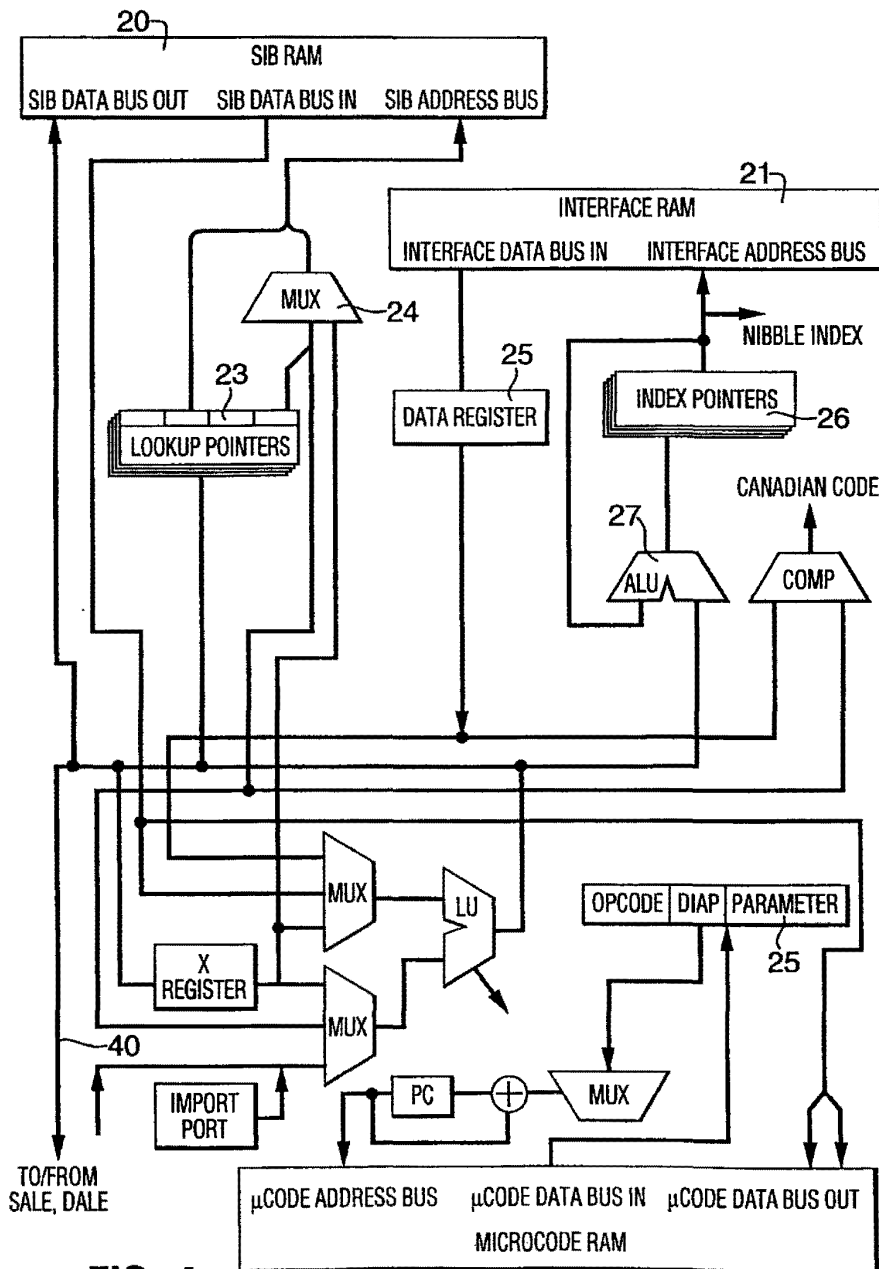


FIG. 4

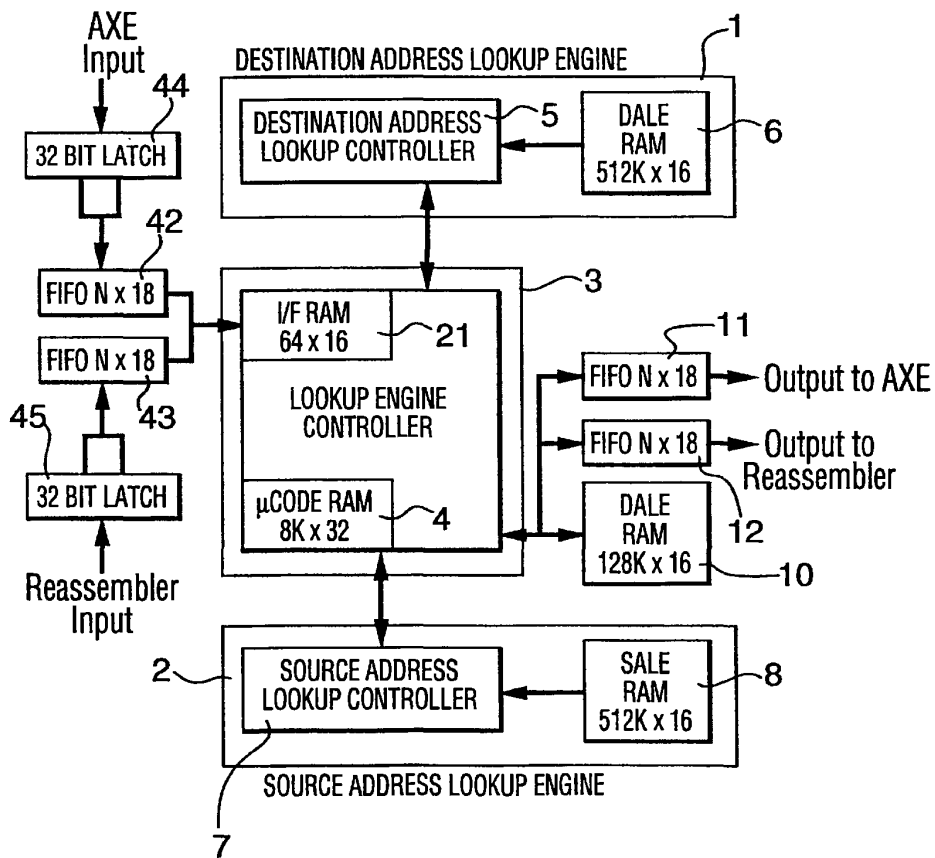


FIG. 5

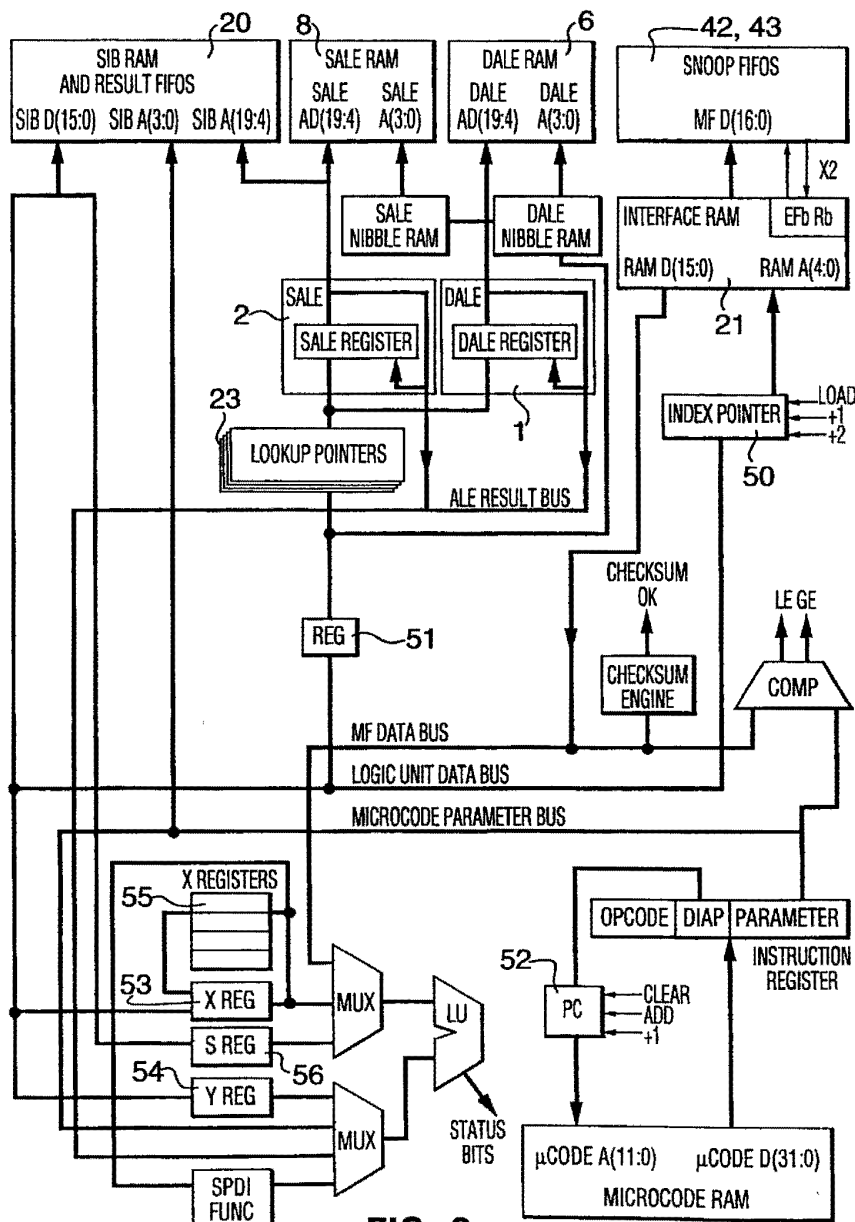


FIG. 6

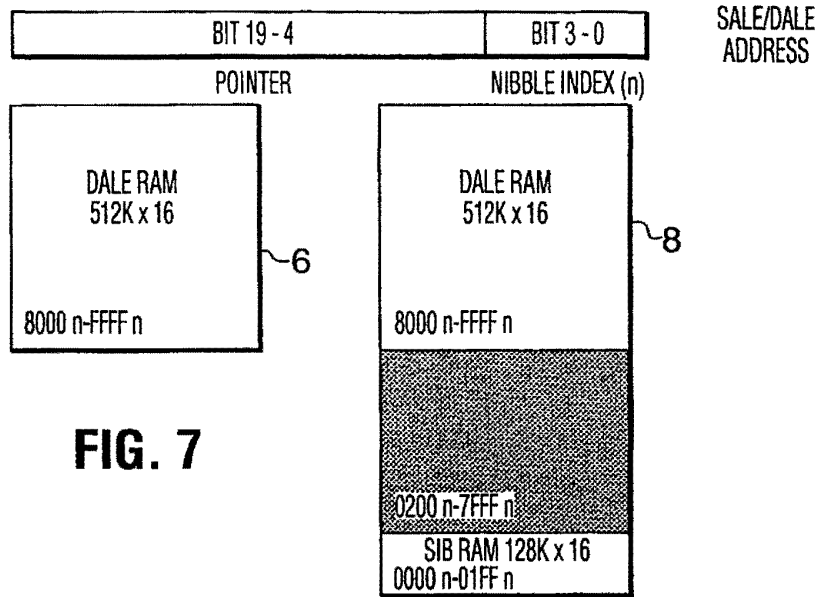


FIG. 7

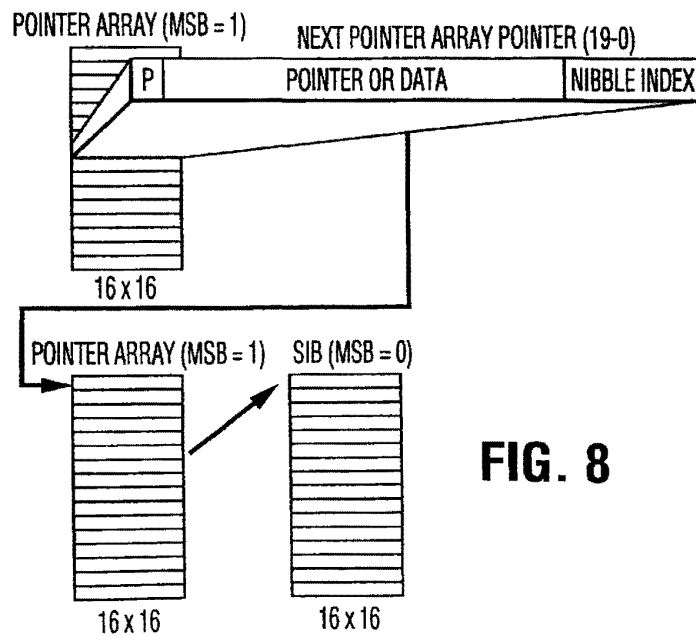


FIG. 8

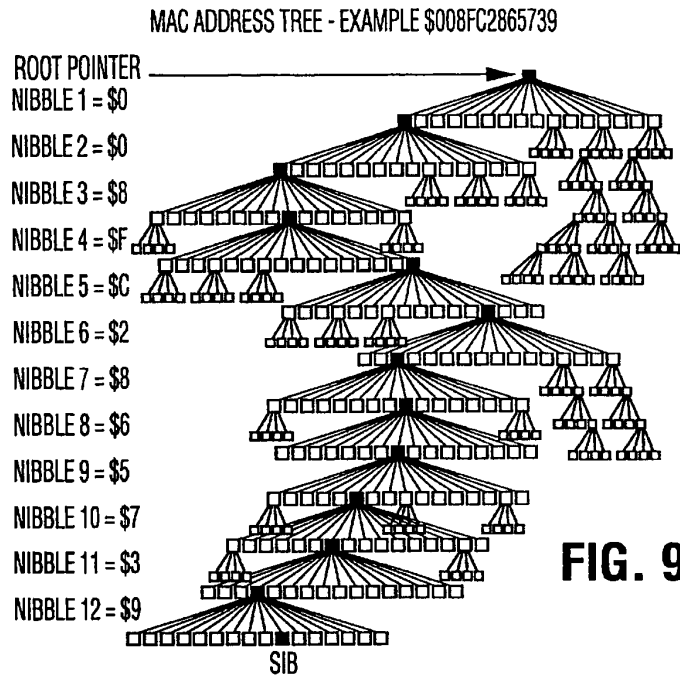


FIG. 9

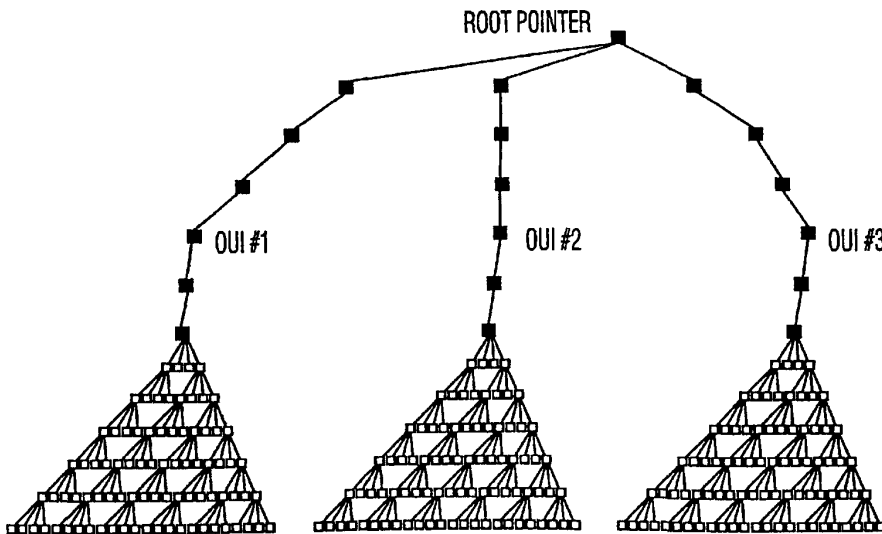


FIG. 10

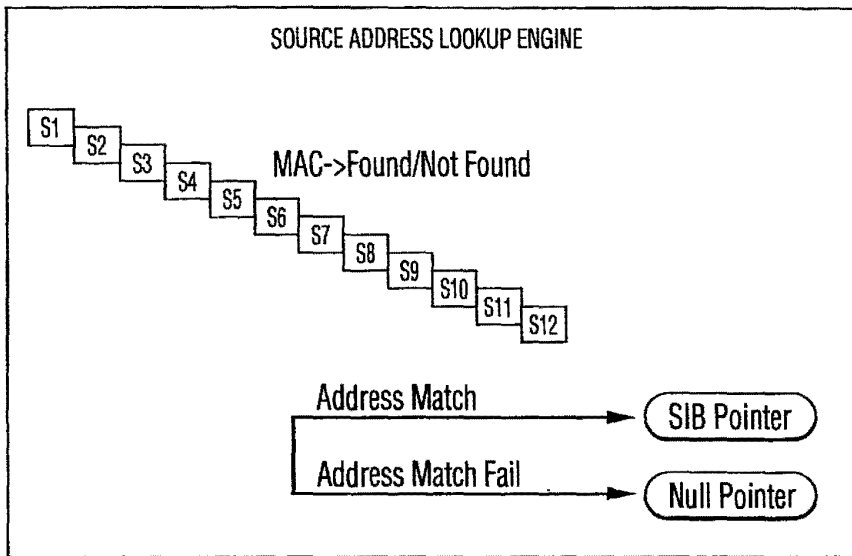


FIG. 11

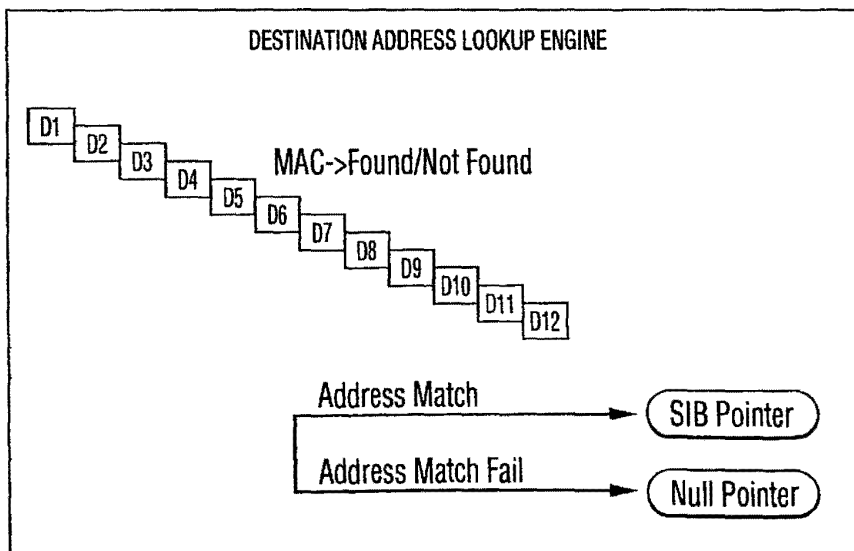


FIG. 12

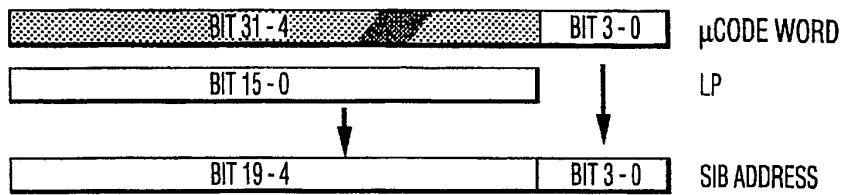


FIG. 13

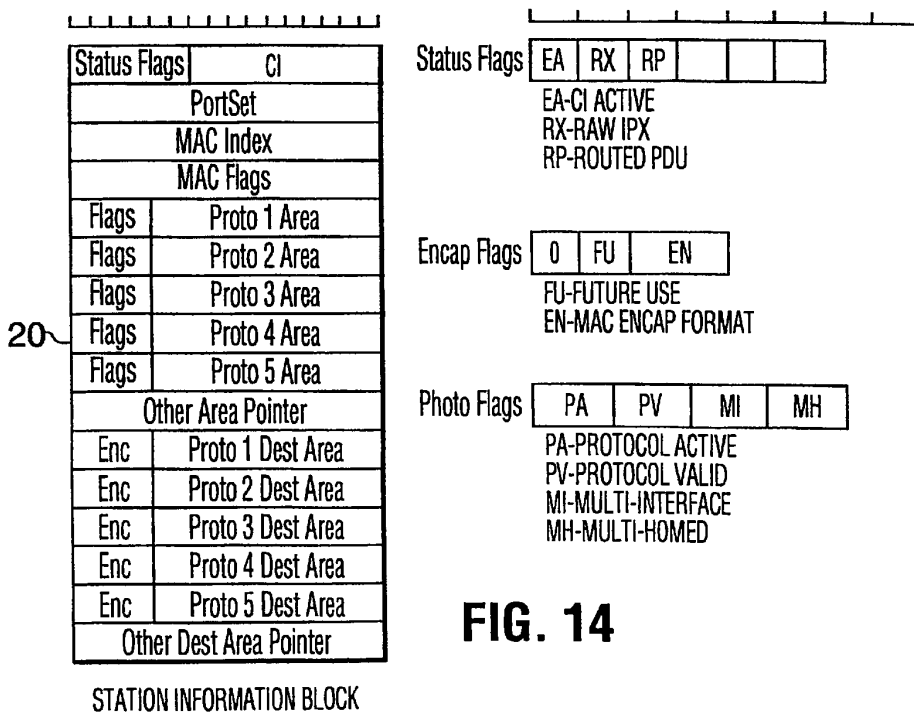


FIG. 14

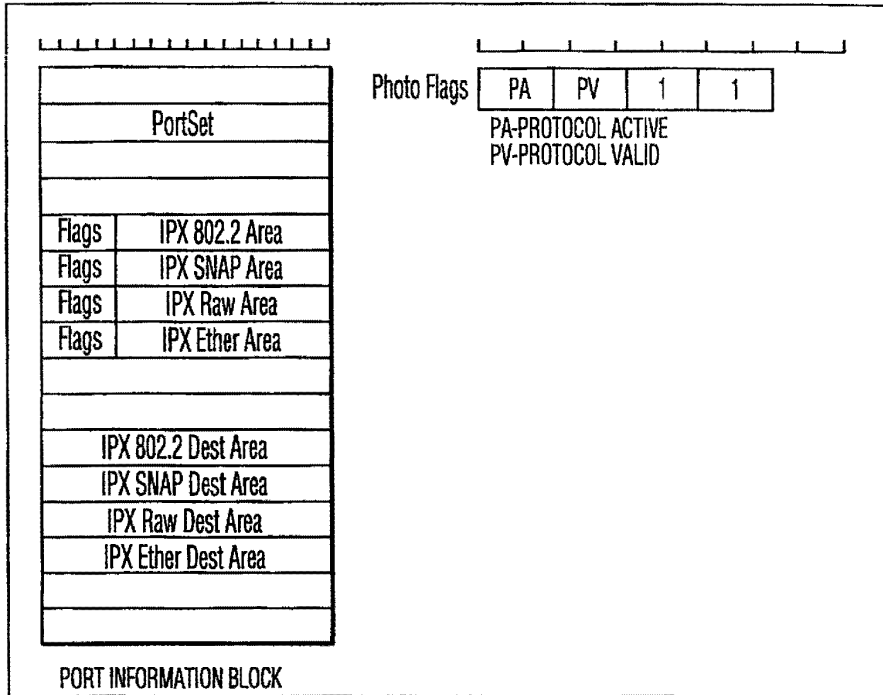


FIG. 15

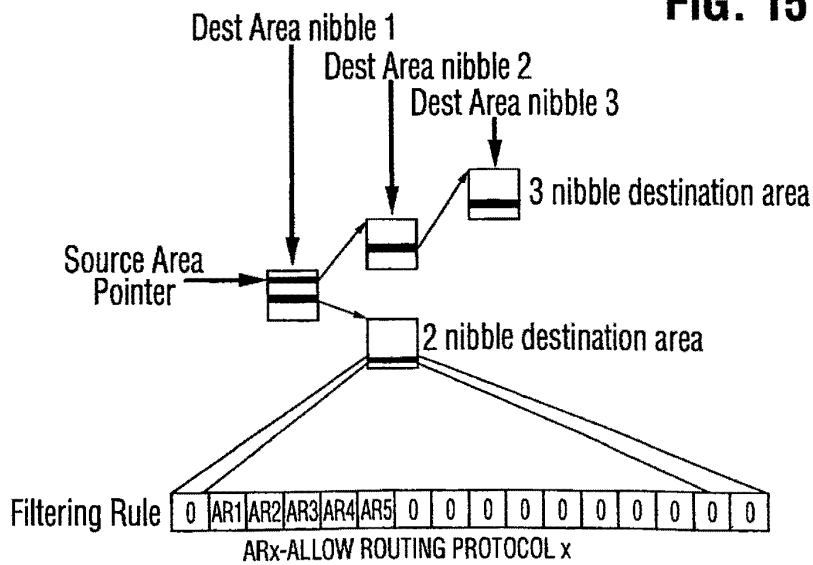


FIG. 16

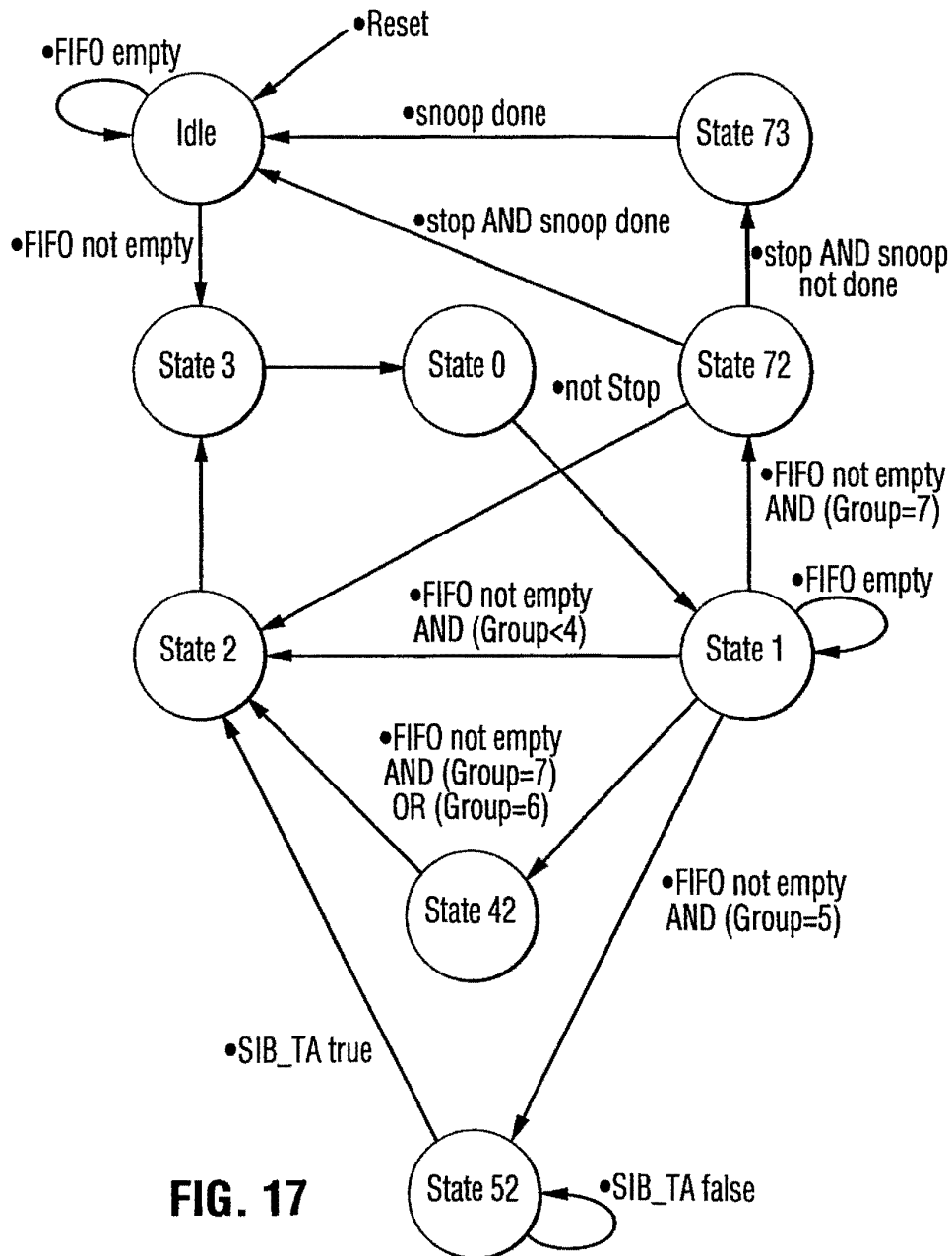


FIG. 17

Increment Branch Instructions (Group 2, no wait states)

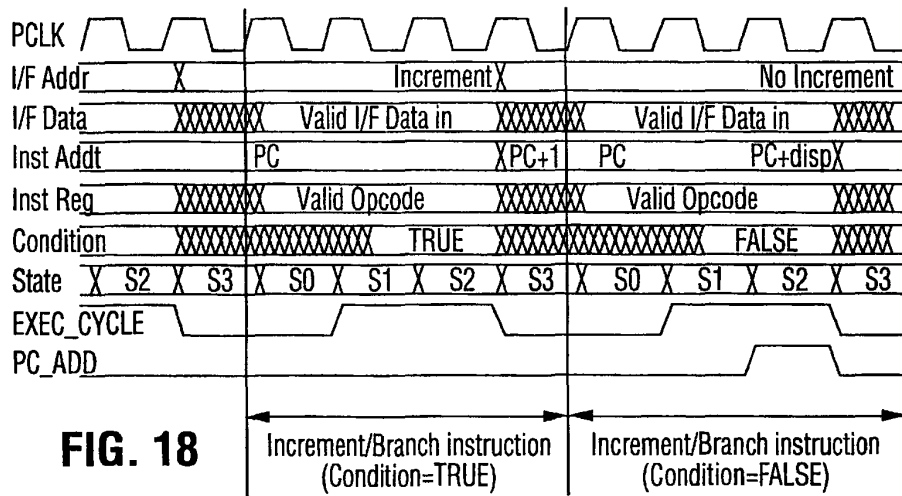


FIG. 18

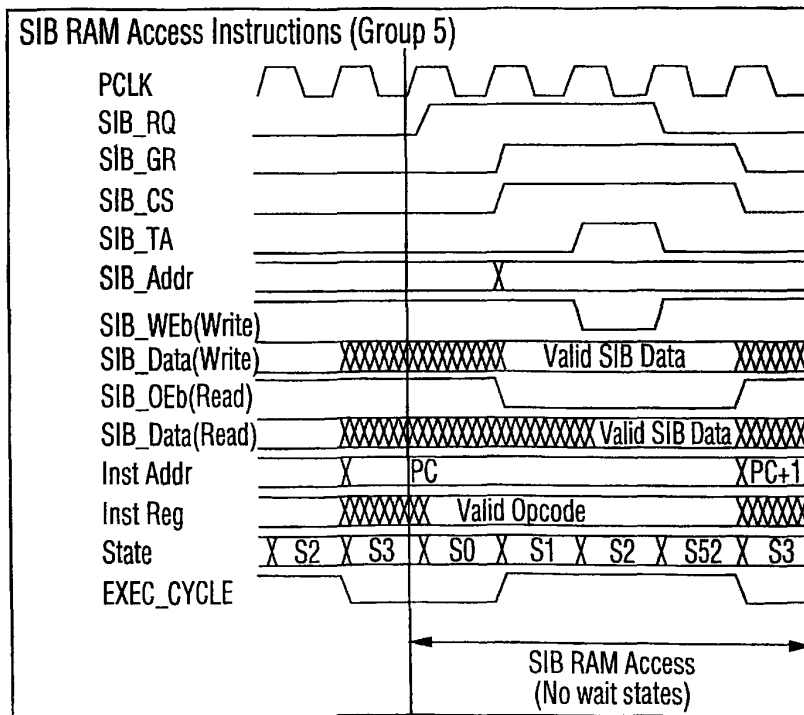


FIG. 19

LOOK-UP ENGINE FOR PACKET-BASED NETWORK

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

This invention relates to the field of data communications, and more particularly to packet-based digital communications networks.

There are two broad classes of network: circuit-based and packet-based. Conventional telephone networks are circuit based. When a call is established in a circuit-based network, a hard-wired connection is set up between the calling parties and remains in place for the duration of the call. Circuit-based networks are wasteful of available bandwidth and lack flexibility.

Packet-based networks overcome many of the disadvantages of circuit-based networks. In a packet-based network, the data are assembled into packets containing one or more address fields which define the context of a packet, such as protocol type and relative positions of other fields embedded in the packet. LAN bridges and routers use the information in the packet to forward it to the destination.

In a packet-based network, a packet must be parsed as it flows through the network. Parsing is the process of extracting and analyzing the information, such as source and destination address and net layer protocol, contained in the packets.

In known networks, packet parsing is generally performed with a microprocessor, which provides flexibility in handling different packet types and can be upgraded to handle new packet types as they are defined. Content Addressable Memory (CAM) is commonly used for hardware assistance to speed up searches through a list of known addresses. This is a tedious task. CAMs are also relatively expensive and limited in size and availability.

General purpose processor architectures are not specifically directed toward the types of operations required in packet parsing and so they tend to be inefficient. To meet performance requirements, a fast but expensive processor based solution can be implemented. In the highest performance systems, hardware solutions are implemented to increase speed, but at the cost of flexibility.

SUMMARY OF THE INVENTION

An object of the invention is to provide a fast, but inexpensive solution to the problem of packet-parsing in packet-based networks.

According to the present invention there is provided an arrangement for parsing packets in a packet-based digital communications network, said packets including packet headers divided into fields having values representing information pertaining to the packet, said arrangement comprising an input memory for receiving fields from said packet headers of incoming packets; and a look-up engine for retrieving stored information appropriate to a received field value. The look-up engine includes at least one memory storing information related to possible values of said fields in a hierarchical tree structure and associated with a respective field of packet headers; a memory controller associated with each said memory storing information related to possible values of said fields for controlling the operation thereof to retrieve said stored information therefrom; and a microcode controller for parsing a remaining portion of the packet header while said stored information is retrieved and controlling the overall operation of said look-up engine.

The memory and retrieving means constitute a look-up engine, which is the central resource containing all infor-

mation necessary for forwarding decisions. In a preferred embodiment the look-up engine includes a source address look-up engine and a destination address look-up engine.

In a packetized data transmission conforming to IEEE802 standards, the packets have a MAC (medium access control) header containing information about the destination and source addresses and the net layer protocol. The invention permits packet switching to be achieved in a bridge-router, for example an Ethernet to ATM bridge-router, at a rate of about 178,000 packets per second using 64 byte minimum Ethernet packets. This means that the MAC headers are interpreted once every 5.6 micro seconds.

The look-up engine preferably employs table look-ups using nibble indexing on variable portions of the packet, such as MAC and network layer addresses, and bit pattern recognition on fixed portions for network layer protocol determination.

Each look-up table is organized into a hexadecimal search tree. Each search tree begins with a 16 word root table. The search key (e.g. MAC address) is divided into nibbles which are used as indices to subsequent tables. The 16 bit entry in the table is concatenated with the next 4 bit nibble to form the 20 bit address of the next 16 word table. The final leaf entries point to the desired information.

Bit pattern recognition is achieved by a microcode instruction set. The microcode engine has the ability to compare fields in a packet to preprogrammed constants and perform branches and index increments in a single instruction cycle typically. The microcode engine has complete control over the search procedure, so it can be tailored to specific look-up functions. New microcode is downloaded as new functions are required.

The look-up engine can perform up to two tree searches in parallel with microcode execution. Look-up time is quick because the microcode determines the packet's network layer format while the source and destination addresses are being searched in parallel. The results of the source and destination look-ups and the protocol determination arrive at roughly the same time, at which point the next level of decisions is made.

The look-up engine also performs protocol filtering between areas. The system allows devices to be grouped arbitrarily into areas on a per protocol basis and defines filtering rules among these areas. The look-up engine keeps track of each station's area for each of its protocols. The source and destination areas are cross-indexed in a search tree, which is used to find the filtering rule between the two areas. Separate filtering rules are defined for bridging and network layer forwarding; bridging is normally allowed within an area while network layer forwarding is selectively allowed between areas.

The parsing controller typically has a pointer to the current field in the packet being examined. The controller moves this pointer to the next field in the packet after all decisions based on the current field are made.

At each decision point on a tree, the current field is compared to a known value or range. If the comparison yields a true condition, the controller moves to the next decision point by moving the current field pointer. Otherwise the field pointer is left alone and controller branches to new code to compare the current field to a different value or range. This process is repeated until a final decision is made.

Moving to the next decision point requires several discrete steps in a general purpose processor. Unlike a general purpose processor, which has the disadvantage that it only has a single memory bus for both instruction and data fetches, the Look-up engine controller has separate buses for instruction and data and typically performs one decision per step. Fast decisions are made possible by a special set of

instructions which both conditionally move the pointer and conditionally branch to new code in a single step. The comparisons and pointer movements can be byte or word wide, according to the current field's size.

The look-up engine implements other optimized instructions which perform bit level logical comparisons and conditional branches within the same cycle as well as other instructions tailored to retrieving data from nibble-indexed data structures.

The look-up engine is preferably divided into the following sections:

- a) one or more nibble tree address look-up engines (ALE)
- b) one microcode engine

Each ALE is used to search for addresses in a tree structure in its own large bank of memory. The result of a search is a pointer to pertinent information about the address. An ALE is assigned to destination addresses (DALE) and source addresses (SALE). The ALEs operate independently of each other.

The microcode engine is used to coordinate the search. It invokes the SALE and DALE to search for the source and destination addresses respectively and continues on to parse the remainder of the packet using an application-specific instruction set to determine the protocol.

The SALE, DALE and microcode engine can execute in parallel and arrive at their corresponding results at roughly the same time. The microcode engine then uses the SALE and DALE results along with its own to arrive at the forwarding decision.

The advantage of using RAM over a CAM is expandability and cost. Increasing RAM is a trivial and inexpensive task compared to increasing CAM size.

The advantage of the microcode engine over a general purpose processor is that an ASIC implementation of the function is much less expensive and less complex than a processor-based design with all the overhead (RAM, ROM) associated with it.

The invention also related to a method of parsing packets in a packet-based data transmission network, said packets including packet headers divided into fields having values representing information pertaining to the packet, comprising storing information related to possible values of said fields, receiving fields from said packet headers of incoming packets, and retrieving said stored information appropriate to a received field value, characterized in that said information is stored in a memory organized in a hierarchical tree structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will now be described in more detail, by way of example only, with reference to the accompanying drawings, in which:

FIG. 1 is an example of a MAC layer header of a typical packet;

FIG. 2 shows the data paths in a typical bridge-router between Ethernet LAN and ATM networks;

FIG. 3 is a block diagram of a first embodiment of a look-up engine in accordance with the invention;

FIG. 4 is a block diagram of a look-up engine controller for the look-up engine shown in FIG. 3;

FIG. 5 is a block diagram of a second embodiment of a look-up engine in accordance with the invention;

FIG. 6 is a block diagram of a look-up engine controller for the look-up engine shown in FIG. 5;

FIG. 7 is a map of look-up engine Address Look-up engine (ALE) memories;

FIG. 8 is a diagram illustrating search tree operation in an ALE;

FIG. 9 shows one example of a MAC search tree;

FIG. 10 shows the effect of the organizationally unique identifier of the MAC addresses on the size of the search tree;

FIG. 11 shows the source address look-up engine table;

FIG. 12 shows the destination address look-up table;

FIG. 13 illustrates the look-up engine addressing modes;

FIG. 14 shows a station information block;

FIG. 15 shows a port information block;

FIG. 16 shows an example of protocol filtering;

FIG. 17 shows a look-up engine controller Instruction State Machine;

FIG. 18 shows a typical fast timing diagram; and

FIG. 19 shows a typical SIB RAM access instruction timing diagram.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

A typical look-up engine (LUE) in accordance with the invention is designed to be used in a twelve-port wire speed Ethernet to ATM bridge-router capable of switching about 178,000 packets per second using 64 byte minimum Ethernet packets. This packet rate corresponds to a look-up request occurring every 5.6 μ secs. The LUE is used each time a packet is received off the Ethernet or the ATM network. The type of information that the engine provides depends on the direction of packet flow and the type of packet.

The look-up engine provides all the information needed to find the path to each known destination, as well as default information in the case of unknown destinations.

FIG. 1 shows a typical MAC layer header format for a packet that can be parsed with the aid of the look-up engine in accordance with the invention. The header comprises destination and source address fields 100, 101, a network layer protocol type field 102, and network layer destination and source address fields 103, 104. FIG. 1 also illustrates how the header is parsed in accordance with the invention. All fields except 102 are parsed using a tree search. The Net Layer Protocol Type field 102 is parsed by using microcode comparisons in the microcode engine to be described.

On a bridge-router, each port is represented by a corresponding bit in a PortSet (Ports 0-11), which is a 16 bit value that has local significance only. The Control Processor and ATM are each assigned a port.

The following definitions are special cases of a PortSet:

SinglePortSet
 a PortSet with a single bit set.
 HostPortSet
 a SinglePortSet corresponding to the Control Processor
 MyPortSet
 a SinglePortSet corresponding to the source port of this packet.
 NullPortSet
 a PortSet of no parts.

A Connection Identifier (CI), which is a 16 bit value with local significance only, is used to map connections into VPI/VCI values.

The following definitions are special cases of CI:

Mesh_CI
 a CI corresponding to a path towards the destination endstation's Bridge-router.

-continued

Null_CI	a CI connected to nothing. It is returned when the destination is attached to the local Bridge-router or if the connection is not allowed.
RS_CI	a CI corresponding to a path to the Route Server.
ABS_CI	a CI corresponding to a path to the Address/Broadcast Server.

MAC layer addresses are globally unique 48 bit values, except in some protocols such as DECNet, where they may not be globally unique.

Unicast_DA	a MAC layer destination address of an end-station.
Router_DA	a MAC layer destination address of the Route Server. An end-station sends packets to the Route Server when it cannot send to the destination directly at the MAC layer.
Broadcast_DA	the broadcast MAC layer address (all ones) which is received by all end-stations. It cannot be a source address.
Multicast_DA	a multicast MAC layer address (group bit set) which is received by end-stations that recognize that multicast address.

Network layer (NL) addresses are network protocol dependent. They are generally divided into Network, Subnet, and Node portions, although not all protocols have all three present. The Network Layer Address Field Sizes (in bits) are summarized in the table below.

Protocol	Total Size	Network	Subnet	Node
IP	32	8/16/24	variable	variable
IPX	80	n/a	32	48 (MAC address)
AppleTalk	24	n/a	16	8
DECNet	64	16 (reserved)	38 (32 = 'HIORD') (6 = subnet)	10

The look-up engine handles unicast network layer addresses.

When the look-up engine is used in a bridge-router providing an interface between an Ethernet and ATM network, packets coming from the Ethernet side are fed into the Look-up Engine. The result of the look-up has the form:

Input	->	Command, CI, PortSet
-------	----	----------------------

where Input is derived from the first few bytes of the packet and Command is an opcode to the AXE (Transfer engine).

The Quad MAC status word distinguishes between router MAC, broadcast and multicast MACs.

Bridging occurs when the destination address is a unicast address other than the Route Server address. Bridging is allowed between two endstations in the same area for a given protocol.

Both source and destination MAC addresses must be known before automatic bridging/filtering is performed; otherwise, the packet is sent to the Route Server for:

- SA (Source Address) validation if the SA has never been seen speaking a given protocol
- DA (Destination Address) resolution if the DA was not found in the local MAC cache.

The Bridge command instructs the AXE (Transfer Engine) to use RFC-1483 bridge encapsulation. BridgeProp command instructs the AXE to use bridge-router encapsulation (include source PortSet in encapsulation)

Unknown_SA	-> BridgeProp, Null_CI, HostPortSet, MyPortSet
* Unknown SA	- send to HP for Spanning Tree processing
* HP	will decide whether to forward it to ABS for learning, depending on Spanning Tree state
Unicast_DA	-> Bridge, Mesh_CI, NullPortSet
* DA in the same area	on a different Bridge-router
Unicast_DA	-> Bridge, Null_CI, NullPortSet
* DA not in the same area	(reject)
* Protocol not allowed	to bridge-router
* DA on the same port	
Unicast_DA	-> Bridge, Null_CI, SinglePortSet
* DA in the same area	on the same Bridge-router but on a different port
Unknown_DA	-> BridgeProp, ABS_CI, NullPortSet, MyPortSet
* DA not found in the table	- send to ABS for flood processing
Broadcast_DA	-> BridgeProp, ABS_CI, NullPortSet, MyPortSet
* Broadcast DA	- Send to Control Processor for broadcast processing
Multicast_DA	-> BridgeProp, ABS_CI, NullPortSet, MyPortSet
* Multicast DA	- Send to ABS for multicast processing
Multicast_DA	-> Bridgeprop, Null_CI, HostPortSet, MyPortSet
* Multicast DA is of interest	to HP (eg Spanning Tree)
* HP	will decide whether to forward it to ABS for multicast processing

Routing occurs when the destination address is the unicast Route Server address. Filtering rules between areas are explicitly defined per protocol. The per protocol source area is an attribute of the source MAC address and the per protocol destination area is an attribute of the destination NL address.

Both source MAC and destination NL addresses must be known before network layer forwarding can occur.

The packet will be bridged to the Route Server if any of the following are true:

- IP options are present
- Protocol is unknown
- The packet will be dropped if any of the following are true:

- Source area is not allowed to send to Destination area for this protocol
- Source NL address is invalid (e.g. any IP broadcast address)
- Checksum is invalid
- Time-To-Live field expires

Unicast_NLDA	-> Route, Mesh_CI, NullPortSet
* NL node on a different	bridge-router
Unicast_NLDA	-> Route, Null_CI, SinglePortSet
* NL node on the same	bridge-router (could be same port)
Unknown_NLDA	-> Bridge, RS_CI, NullPortSet
* unknown NL node	- send to Route Server
Unknown_Protocol	-> Bridge, RS_CI, NullPortSet
* protocol unknown,	or packet with options

FIG. 2 shows the data paths in a typical bridge-router. Control processor 16 has control over the formatting of packets it sends and receives. If the control processor 16 wants look-up engine 17 to perform a look-up, it formats the packet in the same way as Quad Mac 15; otherwise it sends it as a raw packet, which does not require a lengthy look-up. The control processor predetermines the destination by providing a CI (Connection Identifier) and an output Portset as part of the data stream. A bit in the Quad MAC status word indicates a raw packet and the look-up engine simply retrieves the CI and Portset as part of the data stream. A bit

Cache

in the Quad MAC status word indicates a raw packet and the look-up engine simply retrieves the CI and Portset from the data stream and feeds it to the AXE (Transfer Engine) through the result FIFO. The Control processor is responsible for correctly formatting the required encapsulation.

As shown in FIG. 2, packets coming from the AIM side are fed into the look-up engine. The look-up engine accepts an RFC-1483 encapsulated packet and determines whether to look at a MAC or NL address. The result of the look-up will have the form:

Input	->	PortSet
-------	----	---------

Filtering is not performed in this direction. It is assumed that the all filtering is done at the ingress side. It is also assumed that the destination endstation is known to be attached to the receiving Bridge-router, so unicast packets with unknown destination addresses are dropped.

Flood and broadcast packets are encapsulated in a special format which includes an explicit output PortSet.

Unicast_DA	->	SinglePortSet
* DA on this Bridge-router		
Unknown_DA	->	NullPortSet
* DA not in the table (drop) - this situation should not occur.		
Unicast_NLDA	->	SinglePortSet
* NLDA on this Bridge-router		
Unknown_NLDA	->	NullPortSet
* NLDA not in the table (drop) - this situation should not occur.		
Broadcast_DA,PortSet	->	PortSet
* Proprietary Broadcast request received from RS		
Multicast_DA,PortSet	->	PortSet
* Proprietary Multicast request received from RS		
Unknown_DA,PortSet	->	PortSet
* Proprietary Flood request received from RS		

Turning now to FIG. 3, the look-up engine consists of three functional blocks, namely a destination address look-up engine (DALE) 1, a source address look-up engine (SALE) 2, and a look-up engine controller (LEC) 3, which includes a microcode ram 4. DALE 1 includes a destination address look-up controller 5 and DALE RAM 6. SALE 2 includes a source address look-up controller 7 and SALE RAM 8. The input to the look-up engine is through a fast 16-bit wide I/F RAM 9 receiving input from the AXE (Transfer Engine) and reassembler. The output from the look-up engine is through word-wide FIFOs 11, 12.

One embodiment of look-up engine controller (LEC) 3 is shown in more detail in FIG. 4. This comprises (Station Information Block) SIB ram 20, interface ram 21, and microcode ram 22. The SIB ram 20 is connected to look-up pointers 23. Interface ram 21 is connected to data register 25 and index pointers 26 connected to ALU (Arithmetic Logic Unit) 27. Microcode ram 22 is connected to instruction register 28.

The look-up Engine controller 3 is a microcoded engine tailored for efficient bit pattern comparisons through a packet. It communicates with the Source Address Look-up Engine 2 and the Destination Address Look-up Engine 1, which both act as co-processors to the LEC 3.

The look-up engine snoops on the receive and transmit data buses and deposits the header portion of the packet into the I/F RAM 9. The look-up response is sent to the appropriate FIFO 11, 12.

FIGS. 5 show an alternative embodiment of the loop-up engine and controller. In FIG. 5, the LEC 3 includes a 64x16 I/F (Interface) ram 41 connected to FIFO's 42, 43 (First-in, First-out memories) respectively connected to latches 44, 45 receiving AXE (Transfer Engine) and reassembler input.

Referring now to FIG. 6, the LEC 3 also contains several registers, which will now be described. Register select instructions are provided for the register banks (XP0-7, LP0-7).

Index Pointer register (IP) 50 is a byte index into the I/F RAM 21. Under normal operation, the index pointer register 50 points to the current packet field being examined in the I/F RAM 21 but it can be used whenever random access to the I/F RAM 21 is required.

The IP 50 can be modified in one of the following ways:

- 1) loaded by the LOADIP instruction (e.g. to point to the beginning of the packet)
- 2) incremented by 1 (byte compare) or 2 (word compare) if a branch condition is not met.
- 3) incremented by 2 by a MOVE (IP)+ type instruction.

Data Register 51 contains the 16 bit value read from I/F RAM 21 using the current IP. The DR 51 acts like a one word cache; the LEC keeps its contents valid at all times.

Program Counter 52 points to the current microcode instruction. It is incremented by one if a branch condition is true, otherwise the displacement field is added to it.

The Lookup Pointers (LP0-7) 23 are 16 bit registers which contain pointers to the SIB RAM 20. The LPs are used to store pointers whenever milestones are reached in a search. One LP will typically point to a source SIB and another will point to a destination SIB. The LP provides the upper 16 bits of the pointer; the lower 4 bits are provided by the microcode word for indexing into a given SIB.

The LPs are also used to prime the SALE and DALE with their respective root pointers.

X,Y Registers 53, 54 are general purpose registers where logic manipulations can be made (AND, OR, XOR). They are used for setting and clearing bits in certain words in the SIB RAM (e.g. Age bit) and to test for certain bits (e.g. status bits). The X Register 53 can be selected as Operand A to the Logic Unit while the Y Register can be selected as Operand B.

The BYZ and BYNZ instructions conditionally branch on Y=0 and Y<>0 respectively.

The Y Register 54 is the only register source for moves to the result FIFOs.

The X Register 53 can be saved to or restored from X' Registers (X'0-X'7) 55. The mnemonic symbol for the currently selected X' register is XP.

The S Register 56 is a pipelining stage between SIB RAM 20 and the Logic Unit. It simplifies read access from SIB RAM 20 by relaxing propagation delay requirements from SIB RAM 20 valid to register setup. It provides the added advantage of essentially caching the most recent SIB RAM access for repeated use. It is loaded by the GET Index(LP) instruction.

As in FIG. 3, the LEC 3 controls the operation of the look-up engine. All look-up requests pass through the LEC 3, which in turn activates the SALE 2 and the DALE 5 as required. The LEC 3 is microcode based, running from a 32-bit wide microcode RAM. The instruction set consists mainly of compare-and-branch instructions, which can be used to find specific bit patterns or to check for valid ranges in packet fields. Special I/O instructions give the LEC random read access to the interface RAM.

The LEC has access to 3 memory systems: the interface RAM 9, the SIB RAM 20 and the Microcode RAM 22.

The interface RAM 9 is used to feed packet data into the LEC 3. The look-up engine hosts dump packet headers into this RAM through snoop FIFOs 42, 43. This RAM is only accessible through the snooped buses.

The SIB RAM 20 is used to hold information for each known end-station. The LEC 3 can arbitrarily retrieve data from this RAM and transfer it to one of the response FIFOs 11, 12 or to internal registers for manipulation and checking. High speed RAM is also used to minimize the data retrieval time. The size of the SIB RAM 20 is dependent on the maximum number of reachable end-stations. For a limit of 8,000 end-stations, the SIB RAM size is 256K bytes. This RAM is accessible directly to the Control Processor for updates.

cache

The Microcode RAM 22 is dedicated to the LEC 3. It contains the 32 bit microcode instructions. The LEC 3 has read-only access to this high speed RAM normally, but it is mapped directly to the Control Processor's memory space at startup for microcode downloading.

Variable fields of a packet, such as addresses, are searched in one of many search trees in the ALEs 1, 2, (FIG. 5), which are nibble index machines. Each ALE 1, 2 has its own search tree RAM 6, 8 (FIG. 7), which is typically high density but low speed. This RAM is divided into 32 byte blocks which can either be Index Arrays or Information Blocks.

The searches in the ALEs 1, 2 are based strictly on the root pointer, the search key and search key length it is given. A look at the look-up engine memory map (FIG. 7) as viewed from the ALEs shows how the mechanism works.

All search trees in a given ALE 6, 8 reside in the upper half of its memory. The 16-bit root pointer given to the ALE will have the most significant bit set. The search key (e.g. MAC address) is divided into nibbles. The first nibble is concatenated with the root pointer to get an index into the root pointer array. The word at this location is retrieved. If the MSB (Most Significant Bit) (P Bit) is set, the next nibble is concatenated with the retrieved word to form the next pointer. If the P Bit is clear, the search is finished. The final result is given to the LEC, which uses it either as a pointer into the SIB RAM, or as data, depending on the context of the search. A zero value is reserved as a null pointer value. FIG. 8 illustrates search tree operation.

The search key length limits the number of iterations to a known maximum. The control processor manipulating the search tree structure may choose to shorten the search by putting data with a zero P bit at any point in the tree.

"Don't Care" fields are also achievable by duplicating appropriate pointers within the same pointer array. Search trees are maintained by the Control Processor, which has direct access to the SALE and DALE RAMs 6, 8.

FIG. 9 is a diagram illustrating a MAC search tree example. The main purpose of the ALE RAMs 6, 8 is to hold MAC layer addresses. The size of the RAM required for a MAC address tree depends on the statistical distribution of the addresses. The absolute worst case is given by the following formula:

$$N = \sum_{i=1}^L \min(16^{i-1}, X)$$

where

X is the number of addresses

L is the number of nibbles in the address

N is the number of pointer arrays

The amount of memory required, given 32-byte pointer arrays, is 32N. The number obtained from this formula can be quite huge, especially for MAC addresses, but some rationalizations can be made.

In the case of MAC addresses, the first 6 nibbles of the address is the Organizationally Unique Identifier (OUI), which is common to Ethernet cards from the same manufacturer. It can be assumed that a particular system will only have a small number of different OUIs.

The formula for MACs then becomes:

$$N = \sum_{i=1}^6 \min(16^{i-1}, M) + \sum_{j=1}^M \sum_{i=7}^{12} \min(16^{i-7}, X_j)$$

where

M is the number of different OUIs

X_j is the number of stations in OUI_j

Assuming that the addresses are distributed evenly over all OUIs,

$$N = \sum_{i=1}^6 \min(16^{i-1}, M) + M \sum_{i=7}^{12} \min(16^{i-7}, \frac{X}{M})$$

The effect of OUI on Search Tree Size is shown in FIG. 10.

Similar rationalizations can be made with IP and other network layer protocol addresses. An IP network will not have very many subnets and even fewer network numbers.

Although the SALE 2 typically holds locally attached source MAC addresses and the DALE 1 typically holds destination MAC addresses, either ALE 1, 2 is capable of holding any arbitrary search tree. Network layer addresses, intra-area filters, and user-defined MAC protocol types can all be stored in search trees. The decision to put a search tree in either SALE or DALE is implementation dependent; it relies on what searches can be done in parallel for maximum speed.

The principal function of the SALE 2 is to keep track of the MAC addresses of all stations that are locally attached to the bridge-router. Typically one station will be attached to a bridge-router port, but connections to traditional hubs, repeaters and bridge-routers are allowed, so more source addresses will be encountered.

Using the formula for RAM size above, typical RAM calculations for the source address trees are as follows:

Number of OUIs	Number of Stations	Total Bytes
20	400	65,440
2	500	65,184
20	500	77,984
20	800	116,284
5	1,000	131,552

The number of source stations is limited to some fraction of the total allowable stations. The limit is imposed here because the SALE will most likely hold many of the other search trees (e.g. per protocol NL address search trees, intra-area filters).

Whenever a new source address is encountered, the SALE 1 will not find it in the MAC source address search tree. The LEC 3 realizes the fact and sends it to the Control Processor. The new source address is inserted into the search tree once validation is received from the Route Server.

Whenever a previously learned address is re-encountered, the Age entry in the SIB 20 is refreshed by the LEC 3. The control processor clears the Age entry of all source addresses every aging period. The entry is removed when the age limit is exceeded.

The source address look-up engine table is shown in FIG. 11.

The DALE 1 keeps track of all stations that are directly reachable from the bridge-router, including those that are locally attached. The DALE search trees are considerably larger because they contain MAC addresses of up to 8,000 stations.

Typical memory sizes for MAC destination address search trees would be:

Number of OUIs	Number of Stations	Total Bytes
10	8,000	856,992
20	8,000	945,824
30	8,000	1,034,464

A station's MAC address will appear in the MAC search tree if the station is reachable through MAC bridging. A

station's network layer address will appear in the corresponding network layer search tree if it is reachable through routing.

The destination address look-up engine MAC table is shown in FIG. 12.

IP masking may be required if a particular port is known to have a router attached to it. Masking is achieved by configuring the IP network layer search tree in such a way that the node portion of the address is treated as Don't Care bits and the corresponding pointers point to the same Next Index Array.

The SALE and DALE RAMs 8, 6 are divided up into 16 word blocks. These RAMs are accessible only to the corresponding ALE and the Control Processor. These RAMs contain mostly pointer arrays organized in several search trees.

The SIB RAM 20 is divided into 16 word blocks which can be treated as records with 16 fields. Each block typically contains information about an endstation. This RAM is accessible only to the LEC and the CP.

The LEC 3 uses the lookup pointer (LP) as a base pointer into a SIB 20. The contents of the LP is obtained either from the result of a SALE 2 or DALE 1 search to access end-station information, or from a constant loaded in by the microcode to access miscellaneous information (e.g. port information). The LP provides the upper sixteen bits and the microcode word provides the lowest four bits of the SIB RAM address.

The lookup Engine addressing scheme is shown in FIG. 13.

The SIB RAM 20 (FIG. 14) generally contains information about the location of an endstation and how to reach it. For example, the PortSet field may keep track of the port that the endstation is attached to (if it is locally attached) and the connection index refers to a VPI/VCI pipe to the endstation (if it is remotely attached). Other fields are freely definable for other things such as protocol filters, source and destination encapsulation types and quality-of-service parameters, as the need arises.

A variant of the SIB is the Port Information Block (PIB) (FIG. 15). PIBs contain information about a particular port. Certain protocols have attributes attached to the port itself, rather than the endstations. An endstation inherits the characteristics assigned to the port to which it is attached.

The definition of the SIB is flexible; the only requirement is that the data be easily digestible by the LUE instruction set. The field type can be a single bit, a nibble, a byte, or a whole word.

In FIG. 14, the CI (Connection Identifier) field is a reference to an ATM connection to the endstation if it is remotely attached. This field is zero for a locally attached endstation.

The PortSet field is used both for determining the destination port of a locally attached endstation, and for determining whether a source endstation has moved. In one Newbridge-router Networks system, a moved endstation must go through a readmission procedure to preserve the integrity of the network. This field is zero for a remotely attached endstation.

The MAC Index is a reference to the 6-byte MAC layer address of the endstation. This field is used for network layer forwarded packets, which have the MAC layer encapsulation removed. The MAC layer address is re-attached when a packet is re-encapsulated before retransmission out an Ethernet port. The encapsulation flags determine the MAC re-encapsulation format.

The Proto Area and Proto Dest Area fields are used for filtering operations. Because the Newbridge-router system essentially removes the traditional physical constraints on a network topology, the area concept logically re-imposes the constraints to allow existing protocols to function properly.

Filtering rules defined between areas determine whether two endstations are logically allowed to communicate with each other using a specific protocol.

The Proto Area field is a pointer to a filtering rule tree, which is similar in structure to the address trees. The Dest Area field is a search key into the tree. The result of the search is a bitfield in which each protocol is assigned one bit. Communications is allowed if the corresponding bit is set.

FIG. 16 shows a filtering rule tree.

The microcode for the LEC 3 will now be described. The LEC microcode is divided into four main fields as shown in the table below. The usage of each field is dependent on the instruction group.

	31-29	28-24	23-16	15-0
Inst Group	Instruction	Displacement	Parameter	

The instruction group field consists of instructions grouped according to similarity of function. A maximum of eight instruction groups can be defined.

The Instruction field definition is dependent on Instruction Group.

In branch instructions, the Displacement field is added to the PC if the branch condition is true. This field is used by non-branch instructions for other purposes.

The Parameter field is a 16 bit value used for comparison, as an operand, or as an index, dependent on the instruction.

The functions of the groups are set out in the following table.

Group 0	Index Pointer/Bank Select Instructions These instructions manipulate the IP and the register bank select register.
Group 1	Fast Move Instructions These instructions move data between I/F RAM and internal registers.
Group 2	Conditional Branch Instructions These instructions branch when a given condition is met. They can optionally increment the IP.
Group 3	X Register Branch Instructions These instructions branch on an X Register logic comparison.
Group 4	Not Used
Group 5	Slow Move Instructions These instructions generally involve the SIB RAM bus. The access time to the SIB RAM is longer because of address setup time considerations and because the CP may be accessing it at the same time. Access to the Result FIFOs are included here.
Group 6	Not Used
Group 7	Misc Instructions These instructions invoke special functions.

The following table describes the use of each of the fields.

Grp	31-29	28-26	25-24	23-21	20-18	17-16 18-16*	15-0
0	0 0 0	0 0 0	Oper.	1 1 1	1 1 0	BSEL	Immediate Value (15-0) or Register Select (15-4)
1	0 0 1	Dest.	Size	LSel	ASel	BSEL	Immediate Value (15-0) Register Select (15-4) or Index (3-0)
2	0 1 0	Cond.	Size		Disp. (8)		Comparand
3	0 1 1	Cond.	0 0	LSel	Disp. (5)		Comparand
4	1 0 0						
5	1 0 1	Dest.	Size	LSel	ASel	BSEL	Immediate Value (15-0) Register Select (15-4) or Index (3-0)
6	1 1 0						
7	1 1 1	0 0 0	Size	0 0 0	0 0 0	0 0	codes

*when LSel = 110

20

-continued

Condition	
000 - (IP) = Comparand	
001 - (IP) < Comparand	
010 - (IP) > Comparand	
011 - True	
100 - Extended Condition = True	
101 - Extended Condition = False	
110 - Y = 0	
111 - Y <> 0	
Dest - Destination	
000 - currently active FIFO	
001 - X Register	
010 - Lookup Engine Address RAM	
011 - Group 5: S Register otherwise: None	
100 - Y Register	
101 - Index (LP) (SIB RAM)	
110 - XP Register	
111 - Lookup Pointer	
Operation - IP/Register Select operation	
00 - Register Select	
10 - Load	
Size - IP increment size	
00 - no increment	
01 - byte (+1)	
10 - word (+2)	
Displacement (8 bits)	
00000001 - next instruction	
00000000 - same instruction	
Displacement (5 bits)	
00001 - next instruction	
00000 - same instruction	
LSel - Logic Unit Select	
000 - A AND B	
001 - A OR B	
010 - A AND NOT B	
011 - A OR NOT B	
100 - A XOR B	
101 - Reserved	
010 - B	
111 - A	
ASel - Operand A Select	
000 - (IP), (IP)+ Indirect I/F Data	
001 - X X Register	
010 - S S Register	
011 - XP X' Register	
100 - XP X' Register	
101 -	
110 -	
111 -	
BSEL - Operand B Select	
00 - Y Y Register	
01 - #Value Immediate Value	
11 - Special Function	
When LSel = 110:	

Condition	
010 - DALE Lookup Result	
25 110 - SALE Lookup Result	
Immediate Value	
Word values fill the whole field	
Byte values must be repeated twice to fill the field	
When BSEL = 11 (Special Functions):	
30 Value	Function Mnemonic
\$0000 X rotate left 4 L4(X),R12(X)	
\$1000 X rotate 8 (byte swap) SWAP(X),L8(X),R8(X)	
\$2000 X rotate right 4 R4(X),L12(X)	
\$3000 portset(X) PSET(X)	
35 \$4000 X rotate left 1 L1(X)	
\$5000 X rotate right,1 R1(X)	
\$6000 flip X FLIP(X)	
\$7000 LUE Version number VER	
When Value = \$3000 (Portset Function):	
40 X(11:8) f(15:0)	
0 0000000000000001	
1 0000000000000010	
2 0000000000000100	
3 0000000000001000	
45 4 000000000010000	
5 000000000100000	
6 000000001000000	
7 000000010000000	
50 8 000000100000000	
9 000001000000000	
10 000010000000000	
11 000010000000000	
12 000100000000000	
55 13 001000000000000	
14 010000000000000	
15 100000000000000	
60 FIFO Write Instructions	
31-29 28-26 25-24 23-21 20-18 17-16 15-0	
1 0 1 0 0 0 0 0 1 1 0 Extra BSEL Immediate Value (15-0)	
65	

0cc 01	MOVEF #Value,Extra Move Immediate Value to FIFO with Extra bits
0cc 00	MOVEF Y,Extra Move Y Register to FIFO with Extra bits
1cc 00	MOVEF Index(LP),Extra Move Indexed Lookup Data to FIFO with Extra bits

The FIFO write instructions are used to write data into the currently active result FIFO. The Extra field control bits 16 and 17 in the FIFO data bus.

The third instruction in the list is a direct memory access from SIB RAM to the active FIFO. SIB RAM is enabled while the active FIFO is sent a write pulse. Doing so avoids having SIB data propagate through the LUE. Bit 20 differentiates between a DMA and a non-DMA instruction.

The X register cannot be used as a MOVEF source because what would normally be the ASel field conflicts with the Extra field.

Usage:

MOVEF #IPSnap,0 ; Packet is IP over SNAP
Interface RAM Data Read Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	Dest	Size	1 1 1	0 0 0	0 0	Unused

Dest/Size	
001 00	MOVE (IP),X Move IP indirect to X Register
001 10	MOVE (IP)+X Move IP indirect autoinc to X Register
100 00	MOVE (IP),Y Move IP indirect to Y Register
100 10	MOVE (IP)+Y Move IP indirect autoinc to Y Register
111 00	MOVE (IP),LP Move IP indirect to LP Register
111 10	MOVE (IP)+LP Move IP indirect autoinc to LP Register

Interface RAM Data Read instructions are used to read data from the Interface RAM 41 into the X, Y or LP Register. The LP used is preselected using the RSEL instruction.

Lookup Pointer Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
Group	Dest	0 0	LSel	ASel or Extra	BSEL	Immediate Value (15-0) Reg Sel (15-4) or Index (3-0)

Group/Dest/LSel/ASel/BSEL - Instruction Type
101 101 111 001 00 MOVE X,Index(LP) Move X Register to Indexed Lookup Data

-continued

Group/Dest/LSel/ASel/BSEL - Instruction Type
101 101 110 000 00 MOVE Y,Index(LP) Move X Register to Indexed Lookup Data
101 011 000 000 00 GET Index(LP) Load S Register with Indexed Lookup Data
001 111 110 000 00 MOVE Y,LP Move X Register to Lookup Pointer
001 111 110 000 01 MOVE #Value,LP Move Immediate Value to Lookup Pointer
001 111 111 001 00 MOVE X,LP Move X Register to Lookup Pointer

Lookup Pointer instructions are used to load the Lookup Pointers or to store and retrieve values in Lookup RAM.

Usage:

MOVE	Age(LP),X	; Get Age field
...		; check age
...		; reset age
MOVE	X,Age(LP)	; put it back in

Logic Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	Dest	0 0	LSel	ASel	BSEL	Immediate Value (15-0) or Index (3-0)

Logic instructions are used to perform logic manipulations on the X and Y Registers. Combinations of the selections above yield the following (useful) instructions:

Dest/LSel/ASel/BSEL	
001 110 000 00	MOVE Y,X
100 111 001 00	Y -> X
	MOVE X,Y
	X -> Y
001 111 010 00	MOVE S,X
	S -> X
100 111 010 00	MOVE S,Y
	S -> Y
001 110 000 01	MOVE #Value,X
	Immediate Value -> X
100 110 000 01	MOVE #Value,Y
	Immediate Value -> Y
001 000 001 00	AND X,Y,X
	X AND Y -> X
001 000 010 00	AND S,Y,X
	S AND Y -> X
001 000 001 01	AND X,#Value,X
	X AND Value -> X
001 000 010 01	AND S,#Value,X
	S AND Value -> X
100 000 001 00	AND X,Y,Y
	X AND Y -> Y

17

-continued

Dest/LSel/ASel/BSel	
100 000 010 00	AND S,Y,Y S AND Y -> Y
100 000 001 01	AND X,#Value,Y X AND Value -> Y
100 000 010 01	AND S,#Value,Y S AND Value -> Y
OR, ANDN, ORN and XOR are similar to AND: dst 001 aaa bb OR aaa,bb,dst dst 010 aaa bb ANDN aaa,bb,dst dst 011 aaa bb ORN aaa,bb,dst dst 100 aaa bb XOR aaa,bb,dst aaa OR bb -> dst	

Conditional Branch Instructions

31-29	28-26	25-24	23-16	15-0
0 1 0	Cond.	Size	Displacement	Comparand

Cond/Size	
000 01	ESCNE.b #Comparand,Label Escape if Byte Not Equal
000 10	ESCNE.w #Comparand,Label Escape if Word Not Equal
001 01	ESCGE.b #Comparand,Label Escape if Byte Greater or Equal
001 10	ESCGE.w #Comparand,Label Escape if Word Greater or Equal
010 01	ESCLE.b #Comparand,Label Escape if Byte Less or Equal
010 10	ESCLE.w #Comparand,Label Escape if Word Less or Equal
110 00	BYZ Label Branch if Y Register is zero
111 00	BYNZ Label Branch if Y Register is not zero

Increment Branch instructions are used to compare the current packet field with an immediate value. If the condition is met, the branch is taken; otherwise IP is incremented by the Increment Size.
Usage:

Label1: ESCNE.w #AAAA,Label12 ; check if SNAP header
ESCNE.w #0003,OtherLabel ; compare to SNAP value
Label12:

X Register Branch Instructions

31-29	28-26	25-24	23-21	20-16	15-0
0 1 1	Cond	0 0	LSel	Disp	Value

Cond/LSel	
110 100	BXEQ #Value,Label Branch if X is equal to value

18

-continued

Cond/LSel	
111 100	BXNE #Value,Label Branch if X is not equal to value
110 000	ANDBZ #Value,Label Branch if X AND Value is equal to zero
111 000	ANDBNZ #Value,Label Branch if X AND Value is not equal to zero
110 010	ANDNBZ #Value,Label Branch if X AND NOT Value is equal to zero
111 010	ANDNBNZ #Value,Label Branch if X AND NOT Value is not equal to zero

X Register Branch instructions are derived from the X Register Logic instructions with Operand A always set to the X Register and Operand B always set to the Immediate value. The X Register is not affected by any of these instructions. The displacement field is reduced to 5 bits (+/-32 instructions)

Usage:

See Destination Lookup Instruction example
SKIP.w ; ignore the next word field

Other Branch Instructions

31-29	28-26	25-24	23-16	15-4	3-0
0 1 0	Cond	Size	Disp	ExtCond	ExtDisp

Cond/Size/Disp/ExtCond/ExtDisp	
100 00 \$00 \$000 0 DWAIT	Wait for DALE
100 00 \$00 \$800 0 SWAIT	Wait for SALE
101 00 \$00 \$C00 0 FWAIT	Wait for Snoop FIFO done
101 00 ddd \$400 0 BCSERR ddd	Branch on checksum error
011 01 \$01 \$000 0 SKIP.b	Skip Byte (same as IBRA.b +1)
011 10 \$01 \$000 0 SKIP.w	Skip Word (same as IBRA.w +1)
011 01 ddd \$000 0 IBRA.b	Increment Byte and Branch Always
011 10 ddd \$000 d IBRA.w	Increment Word and Branch Always
011 00 000 \$800 0 SWITCH	Switch on X (add X to PC)
011 00 ddd \$000 d BRA.u	Branch Always

These instructions are derived from the conditional branch instructions. Wait instructions loop until the extended condition is false. Skip instructions move to the next instruction and increment the IP appropriately.

More branch instructions can be defined easily by using Cond=100 or 101 and picking an unused ExtCond pattern.

When Cond=011 (True), the displacement field is extended to 12 bits.

The SWITCH instruction adds the least significant nibble of X to the PC. If X(3:0)=0, 16 is added to the PC.

Usage:

SKIP.w ; ignore the next word field
Index Pointer/Register Select Instructions

Index Pointer/Register Select Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
Group	Dest	Oper	LSel	ASel	BSel	Immediate Value (15-0) or Register Select (15-4)

Group/Dest./Oper/LSel/ASel/BSel

001 110 00 111 000 00	ST	X[,XPn,LPn]				
		X -> XP, optionally switch to XPn,LPn				
001 001 00 111 100 00	LD	X[,XPn,LPn]				
		XP -> X, optionally switch to XPn,LPn				
001 011 00 111 000 00	RSEL	XPn,LPn				
		switch to XPn,LPn				
000 011 10 111 000 01	LOADIP	# Value				
		Load IP immediate				
000 011 10 111 001 00	LOADIP	X				
		Load IP with X				

Index Pointer instructions are used to perform manipulations on the index pointer.

Transfers from the X registers are not normally used in a lookup function but may be useful for general purpose transfers from interface RAM.

The Register Select instruction selects a register from each of the register banks. The format of the Bank Select Bits field is:

15-12	11	10-8	7	6-4	3-0
X X X X	XEn	XSel	LPEn	LPSEL	X X X X

The En bits determine whether the corresponding select bits are valid. If En is zero, the corresponding register selection remains unchanged. If En is one, the corresponding select bits are used. This mechanism allows register selections to be made independent of each other.

Destination Lookup Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	0 1 0	Size	1 1 1	ASel	0 0	Command/Address

Size/ASel

00 001	DLOAD X,Address [,Command]
	Load X into DALE
00 000	DLOAD (IP),Address [,Command]
	Load IP indirect into DALE / load Command Reg
10 000	DLOAD (IP)+,Address [,Command]
	Load IP indirect autoinc into DALE / load Command Reg

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	Dest	0 0	1 1 0	0 0 0	1 0	not used

Dest

111	DMOVE LP	Move DALE result pointer into Lookup Pointer
001	DMOVE X	Move DALE result pointer into X Register
100	DMOVE Y	Move DALE result pointer into Y Register

The destination lookup instructions set up the DALE and read results from it. The currently selected lookup pointer is used as the root pointer.

The DLOAD instruction loads words into the 16 by 16 bit DALE Nibble RAM and loads the Command Register. The DMOVE instruction returns the DALE result.

Command Register

15	14	13-12	11-4	3-0
Start	0	Nibble Offset	0 0 0 0 0 0 0	Address

The Start bit signals the DALE to start the lookup.

The Nibble Offset field points to the first valid nibble in the first word loaded into the Address RAM.

The Address field points to the word being written in Nibble RAM.

The DMOVE instruction gets the 16 bit DALE result pointer. DMOVE should be preceded by DWAIT, otherwise the result may be invalid.

Usage:

LOADIP	#StartOfPacket	; point to start of packet
DLOAD	(IP)+, Word1	; load DA word 1
DLOAD	(IP)+, Word2	; load DA word 2
DLOAD	(IP)+, Word3, Start	; load DA word 3 and start lookup
...		; do other stuff
DMOVE	X	; get result
BXNE	#Null,DAFound	; address found in table

Source Lookup Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	0 1 0	Size	1 1 1	ASel	0 1	Command/Address

Size/ASel

00 001	SLOAD X,Address [,Command]
	Load X into SALE
00 000	SLOAD (IP),Address [,Command]
	Load IP indirect into SALE / load Command Word
10 000	SLOAD (IP)+,Address [,Command]
	Load IP indirect autoinc into SALE / load Command Word

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	Dest	0 0	1 1 0	0 0 1	1 0	Immediate Value (15-0)

Dest	
111	SMOVE LP Move SALE result pointer into Lookup Pointer
001	SMOVE X Move SALE result pointer into X Register
100	SMOVE Y Move SALE result pointer into Y Register

The destination lookup instructions set up the SALE and read results from it. The currently selected lookup pointer is used as the root pointer.

The SLOAD instruction loads words into the 16 by 16 bit SALE Nibble RAM and loads the Command Word. The SMOVE instruction returns the SALE result. Command Word

15	14	13-12	11-4	3-0
Start	0	Nibble Offset	0 0 0 0 0 0 0	Address

The Start bit signals the SALE to start the lookup. The Nibble Offset field points to the first valid nibble in the first word loaded into the Address RAM.

The Address field points to the word being written in Address RAM.

The SMOVE instruction gets the 16 bit SALE result pointer. The SMOVE instruction should be preceded by SWAIT, otherwise the result may be invalid. Usage:

SLOAD	(IP)+,Word1	; load DA word 1
SLOAD	(IP)+,Word2	; load DA word 2
SLOAD	(IP)+,Word3,Start	; load DA word 3 and start lookup
...		; do other stuff
SWAIT		; wait for SALE to finish
SMOVE	X	; get result
BXNE	#Null,SAFound	; address found in table

Checksum Engine Instructions

31-29	28-26	25-24	23-21	20-18	17-16	15-0
0 0 1	0 1 0	Size	1 1 1	ASel	1 0	\$8000

Size/ASel	
00 001	CLOAD X Load X into Checksum Engine and start
00 000	CLOAD (IP) Load IP indirect into Checksum Engine and start
10 000	CLOAD (IP)+ Load IP indirect autoinc into Checksum Engine and start

The CLOAD instruction loads a word count into the checksum engine, clears the checksum and starts the engine. The word currently indexed by IP is subsequently added to

the checksum each time the IP crosses a word boundary until the count is exhausted.

Miscellaneous Instructions

31-29	28-16	15-0
1 1 1	0 0 0 0 0 0 0	Code (2-0)

These instructions invoke special functions

Code	
001	STOP Stop execution until next lookup request

The lookup engine operation will now be described in more detail. The instruction State Machine (ISM) is shown in FIG. 17.

A lookup engine microcode will typically take four clock cycles. At 50 MHz, the instruction cycle takes 80 ns to execute. Instructions that require access to SIB RAM, which require arbitration with the Control Processor, and any future extensions that require more time to execute will require one or more additional cycles to complete.

After reset, the 3 LEC is in the idle state. As soon as one of the snoop FIFOs 42, 43 is non-empty, the ISM enters the main instruction cycle loop.

A microcode instruction cycle is typically divided into four main states. State 3 and State 0 allow the microcode contents to propagate through the LEC. The instruction group is determined in State 1. If a fast instruction is being executed (Groups 0-3), State 2 is entered immediately. Otherwise the appropriate next state is entered according to the Group field.

FIG. 18 shows a typical fast instruction. By the time State 2 is reached, all signals will have settled. New values for the PC and if necessary, the IP and/or the selected destination, are loaded at the end of this state.

State 42 is a dummy state for currently undefined groups. State 52 is a wait state for external accesses to SIB RAM. The ISM exits this state when the SIB RAM has been granted to the LEC long enough for an access to complete.

FIG. 19 shows a typical SIB RAM access instruction. States 72 and 73 are executed during the STOP instruction. State 73 flushes the snoop FIFOs in case.

The LEC cycles through States 0 to 3 indefinitely until a STOP instruction is encountered, which brings the LEC back to the idle state.

The lookup request mechanism for a MAC layer lookup is as follows:

The requester (e.g. the AXE) places information, generally a packet header, into the snoop FIFO.

The empty flag of the FIFO kickstarts the LEC.

The LEC instructs the DALE to look up the destination address.

The LEC instructs the SALE to look up the source address.

The LEC looks into the packet to determine the network layer protocol in case it needs to be forwarded.

The LEC waits for the SALE and reads the Source Address SIB pointer.

The source port is compared against the previously stored portset to see if the source endstation has moved. The LEC waits for the DALE and reads the Destination Address SIB pointer.

PROCESS

The destination area is compared to the source area to see if the endstations are in the same area.

The source port is compared against the destination port to see if the endstations are on the same port.

Packets are discarded if they serve no other useful purpose (e.g. SA and DA on the same port or in different areas, errored packets). Otherwise they are sent to the Control Processor for further processing.
Sample Program

```

; File: BDG.a
; Unicast Bridging Case
; Release 1.1 Functionality
;-----
BDG_Start:
; XO = Packet Status Word
; IP = Points to 2nd byte of PSW
; DR = Contains Packet Status Word
; XO, LPO are default XP, LP
MOVE    $8000,LP    ;Look up Destination MAC
DLOAD  (IP)+,0     ;Load Dst Addr bits 0-15
DLOAD  (IP)+,1     ;Load Dst Addr bits 16-31
                ;Load Dst Addr bits 32-47
DLOAD  (IP)+,2,$8000 ;and start lookup
MOVE    $8000,LP    ;Look up Source MAC
SLOAD  (IP)+,0     ;Load Src Addr bits 0-15
SLOAD  (IP)+,1     ;Load Src Addr bits 16-31
                ;Load Src Addr bits 32-47
SLOAD  (IP)+,2,$8000 ;and start lookup
; determine protocol here
ESCGE.w    1500,                ;check if 802.3 format
                CheckEncType
ESCNE.w    $AAAA,                ;check DSAP/SSAP
                $AAAA
                UnknownType
ESCNE.w    $0300,                ;check CTL field
                SNAPUnknown-
                Type
ESCNE.w    $0000,                ;check protocol type field
                SNAPUnknown-
                Type
ESCNE.w    $0800,                ;check protocol type field
                SNAPUnknown-
                Protocol
; It's IP over SNAP
BdgSNAPIP:
CLOAD 5      ;assume IP
                header length is
                5
ESCNE.w    $4500,                ;check IP header
                BdgSNAPIP...
                withOpt
SKIP.w    ;skip length
SKIP.w    ;skip identifica-
                tion
SKIP.w    ;skip offset
ESCLE.b    $01,                ;check TTL
                BdgSNAPIP...
                TTLExpired
SKIP.b    ;skip protocol
SKIP.w    ;skip checksum
MOVE (IP)+,X ;read NLSA
MOVE R12(X),X ;shift first nibble
                to bottom
SWTCH      ;check IP Class
BRA.u     BdgSNAPIP-
                ClassA ;0xxx = Class A address
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA
BRA.u     BdgSNAPIP-
                ClassA ;10xx = Class B address
BRA.u     BdgSNAPIP-
                ClassB
BRA.u     BdgSNAPIP-
                ClassB
BRA.u     BdgSNAPIP-
                ClassB
BRA.u     BdgSNAPIP-
                ClassB

```

-continued

BRA.u	BdgSNAPIP- ClassC	;110x = Class C address
BRA.u	BdgSNAPIP- ClassC	
BRA.u	BdgSNAPIP- ClassD	;1110 = Class D address
BRA.u	BdgSNAPIP- ClassE	;1111 = Class E address (future)
BRA.u	BdgSNAPIP- ClassA	;0xxx = Class A Address
BdgSNAPIPClassA:		
OR	X,\$FF00,X	;check if broadcast
BXNE	\$FFFF, BdgSNAPIP_ NLSARealign	
MOVE	(IP)+,X	;check lower address word
BXEQ	\$FFFF, BdgSNAPIP_ NLSAInvalid	;all ones host address
BRA.u	BdgSNAPIP_ NLSAInvalid	;broadcast SA is not allowed
BdgSNAPIP_ NLSAInvalid	BdgSNAPIP_ NLSAValid	
BdgSNAPIPClassB:		
MOVE	(IP)+,X	;check lower address word
BXNE	\$FFFF, BdgSNAPIP_ NLSAInvalid	
BRA.u	BdgSNAPIP_ NLSAInvalid	
BdgSNAPIPClassC:		
MOVE	(IP)+,X	;check lower address byte
OR	X,\$FF00,X	;check if broadcast
BXEQ	\$FFFF, BdgSNAPIP_ NLSAInvalid	
BRA.u	BdgSNAPIP_ NLSAInvalid	
BdgSNAPIPClassD:		
SKIP.w		
BRA.u	BdgSNAPIP_ NLSAInvalid	
BdgSNAPIP_ NLSAInvalid		
SWAIT		;clean up after SALE and DALE
DWAIT		
OR	XP,CMD_ DISCARD CMD_ UNICAST,Y	;Load command Word
MOVEF	Y, FIRST	;Send Command Word
MOVEF	NULL,_CI	;Send CI Index
MOVEF	PORT,_CP	;Dest Port is CP
MOVEF	RSN,_FRC_ MAC,_SRC_ INVALID	;Send Reason
STOP		
BdgSNAPIP_ NLSAInvalid:		
SKIP.w		;skip NLDA
BCSERR BDG_ SNAPIP_CSError		
RSEL	LP1	;Store source SIB pointer in LP1
SWAIT		
SMOVE	Y	;Y contains SALE result
MOVE	Y,LP,LP2	;LP1 points to Source Addr SIB ;Store dest SIB pointer in LP2
BYNZ	BDG_SrcHit	
BDG_SrcMiss:		;*** Source Cache Miss ***
OR	XP,CMD_ FWDCP CMD_ UNICAST,Y	;Load command Word
MOVEF	Y, FIRST	;Default MAC Ethernet Type ;Default Low priority ;Send Command Word

-continued

```

MOVEF      NULL_CI      ;Send CI Index
MOVEF      PORT_CP      ;Dest Port is CP
MOVEF      RSN_FRC_     ;Send Reason
           MAC_SRC_
           MISS
STOP       ;Done!!!
BDG_SNAPIP_
CSError:
OR         XP,CMD_      ;Load command Word
           DISCARD |
           CMD_
           UNICAST,Y
MOVEF      Y, FIRST     ;Send Command Word
MOVEF      NULL_CI      ;Send CI Index
MOVEF      PORT_CP      ;Dest Port is CP
MOVEF      RSN_FRC_     ;Send Reason
           MAC_CSERR
STOP
BDG_SrcHit:
DWAIT
DMOVE     Y             ;Get DALE result
MOVE     Y,LP,LP1      ;point to source SIB
BYNZ     BDG_           ;and check source port
BDG_DeatMiss:
           CheckSrcPort
           ;*** Destination
           Cache Miss ***
OR         XP,CMD_      ;Load command Word
           FWDCP |
           CMD_
           UNICAST,Y
           ;Default MAC Ethernet Type
           ;Default Low priority
MOVEF      Y, FIRST     ;Send Command Word
MOVEF      NULL_CI      ;Send CI Index
MOVEF      PORT_CP      ;Dest Port is CP
MOVEF      RSN_FRC_     ;Send Reason
           MAC_DST_
           MISS
STOP       ;Done!!!
BDG_CheckSrcPort:
GET       SIB_MAC_      ;Compare portsets in LP => Src SIB
           PORTSET(LP)
AND       S,PSET(X),Y   ;Y = src addr bit AND src port bit
BYNZ     BDG_           ;source moved if bits don't match
BDG_SrcMove:
           CheckDestArea
           ;*** Source
           Moved ***
OR         XP,CMD_      ;Load command Word
           FWDCP |
           CMD_
           UNICAST,Y
           ;Default MAC Ethernet Type
           ;Default Low priority
MOVEF      Y, FIRST     ;Send Command Word
MOVEF      NULL_CI      ;Send CI Index
MOVEF      PORT_CP      ;Dest Port is CP
MOVEF      RSN_FRC_     ;Send Reason
           SRC_MOVED
STOP       ;Done!!!
BDG_CheckDestArea:
RSEL     LP2            ;point to dest SIB
GET       SIB_PROTO_    ;get IP Dest Area
           AREA_1(LP)
AND       S,MASK_
           AREA,Y;Mask
           off top 4 bits
BYNZ     BDG_
           CheckSrcArea
BDG_DeatAreaInvalid: ;*** Destination
           Area Invalid ***
LD        X
OR         X,CMD_      ;Load command Word
           DISCARD |
           CMD_
           UNICAST,Y
;Default MAC Ethernet
Type
;Default Low priority
;Default Multicast
MOVEF      Y, FIRST     ;Send Command Word
MOVEF      NULL_CI      ;Send CI Index
MOVEF      PORT_CP      ;Dest Port is CP

```

-continued

```

MOVEF      RSN_DRC_   ;Send Reason
           DST_AREA_
           INV
STOP
BDG_CheckSrcArea:
RSEL       LP1        ;get ready for Source Addr check
GET        SIB_PROTO_
           AREA_1(LP)
OR         S,SIB_AREA_ ;set PA bit in SIB_IPAREA
           PROTO_
           ACTIVE,X
MOVE      X,SIB_      ;modify
           PROTO_
           AREA_1(LP)
AND       X,MASK_     ;Mask off top 4 bits
           AREA, X
XOR       X,Y,Y,LP2  ;check against Dest Area
;switch to LP2 (Dest
SIB)
BYZ       BDG_
           CheckDestPort
BDG_SrcAreaInvalid: ;*** Source
           Area Invalid ***
OR        XP,CMD_     ;Load command Word
           DISCARD |
           CMD_
           UNICAST,Y
;Default MAC Ethernet
Type
;Default Low priority
;Default Multicast
MOVEF     Y, FIRST   ;Send Command Word
MOVEF     NULL_CI    ;Send CI Index
MOVEF     PORT_CP    ;Dest Port is CP
MOVEF     RSN_DRC_   ;Send Reason
           SRC_AREA_
           INV
STOP
BDG_CheckDestPort:
;X0, LP2 are
default XP, LP
LD        X           ;restore PSW
GET       SIB_MAC_    ;S = dest address portset
           PORTSET(LP)
AND       S, PSET(X),Y ;compare against source port portset
BYZ      BDG_OK
BDG_SamePort:
;*** Src Port =
Dest Port ***
OR        XP,CMD_     ;Load command Word
           DISCARD |
           CMD_
           UNICAST,Y
;Default MAC Ethernet
Type
;Default Low priority
MOVEF     Y, FIRST   ;Send Command Word
MOVEF     NULL_CI    ;Send CI Index
MOVEF     PORT_NULL  ;Dest Port is NULL
MOVEF     RSN_DRC_   ;Send Reason
           DST_SAME
STOP
BDG_OK:
;*** Bridge-
router ***
OR        XP,CMD_     ;Load command Word
           BRIDGE-
           ROUTER |
           CMD_
           UNICAST,Y
;Default MAC Ethernet
Type
;Default Low priority
MOVEF     Y, FIRST   ;Send Command Word
MOVEF     SIB_MAC_CI ;Send CI Index from dst SIB
           (LP)
MOVEF     SIB_MAC_   ;Dest Port is determined from dst SIB
           PORTSET(LP)
MOVEF     SIB_MAC_   ;Get MAC Index from dst SIB
           MACINDEX
           (LP)
STOP
;Done!!!

```

The described look-up engine is capable of performing bridge-router and most network layer look-ups in less than 5.6 μ s (1/178,000) with to minimum RAM requirements and cost and maximizes flexibility for future additions/corrections without hardware changes.

The intended application of the look-up engine is high performance LAN systems and other packet-based devices.

GLOSSARY

BRIDGE-ROUTER	A LAN bridging-routing device, with 12 ethernet ports and 1 ATM port.
ATM	Asynchronous Transfer Mode. A cell relay standard.
ABS	Address/Broadcast Server A component of a Route Server that handles address resolution and broadcast traffic.
AXE	A Transfer Engine
DA	Destination Address. The MAC address of the intended destination of a MAC frame.
DALE	Destination Address Look-up Engine. The LUE component that generally searches through a table of MAC layer destination addresses.
CI	Connection Identifier. A number internally used to indicate a particular connection.
IP	Internet Protocol A popular network layer protocol used by the Internet community.
IPX	Internet Packet Exchange A Novell developed network layer protocol.
LEC	Look-up Engine Controller. The LUE component that executes microcode.
LUE	Look-up Engine.
MAC	Medium Access Control. A term commonly encountered in IEEE 802 standards generally referring to how a particular medium (ie. Ethernet) is used. "MAC address" is commonly used to refer to the globally unique 48 bit address given to all interface cards adhering somewhat to the IEEE 802 standards.
RS	Route Server.
SA	Source Address. The MAC address of the originator of a MAC frame.
SALE	Source Address Look-up Engine. The LUE component that generally searches through a table of MAC layer source addresses.
SIB	Station Information Block. The data structure in the LUE that holds relevant information about an endstation.
CAM	Content Addressable Memory.
VPI	Virtual Path Identifier
VCI	Virtual Channel Identifier
Control Processor	The processor in the Bridge-router that handles management functions

We claim:

1. An arrangement for parsing packets in a packet-based digital communications network, said packets including packet headers divided into fields having values representing information pertaining to the packet, said arrangement comprising:

- a) an input memory for receiving fields from said packet headers of incoming packets; and
- (b) a look-up engine for retrieving stored information appropriate to a received field value, said look-up engine including:
 - (i) at least one memory storing information related to possible values of said fields in a hierarchical tree structure and associated with a respective field of packet headers;
 - (ii) a memory controller associated with each said memory storing information related to possible values of said fields for controlling the operation thereof to retrieve said stored information therefrom; and
 - (iii) a microcode controller for parsing a remaining portion of the packet header while said stored information

is retrieved and controlling the overall operation of said look-up engine.

2. An arrangement as claimed in claim 1, wherein said memory controller associated with each said memory compares, at each decision point on the tree structure, the current field with a stored value or range, and moves to the next decision point by moving a pointer for the current field and branching to new code if said comparison results in a first logical condition, and if said comparison results in a second logical condition the current field is compared to a different value or range, and so on until said comparison results in said first logical condition.

3. An arrangement as claimed in claim 1, wherein said controller associated with each said memory compares values based on successive nibbles of a field value in said memory with stored values to locate the related information.

4. An arrangement as claimed in claim 3, wherein said memory controller associated with each said memory concatenates a first nibble of an incoming field value with a root pointer to obtain an index to a root pointer array, retrieves a word at a location identified by said index, concatenates the next nibble with the retrieved word to form the next pointer and so on until said related information is retrieved.

5. An arrangement as claimed in claim 1, wherein said at least one memory is a random access memory (RAM).

6. An arrangement as claimed in claim 1, wherein one of said fields comprises a destination address and said related information comprises the path data associated with said respective destination addresses.

7. An arrangement as claimed in claim 1, wherein a plurality of said memories storing information related to possible values of said fields in a hierarchical tree structure operate in parallel and are associated with respective fields of said packet headers.

8. An arrangement as claimed in claim 7, wherein each said memory is a random access memory (RAM).

9. An arrangement as claimed in claim 7, wherein one of said fields comprises a destination address and said related information comprises the path data associated with said destination address, and another of said fields comprises a source address, and said look-up engine also locates path data associated with the source in parallel with the location of the path data associated with the destination address.

10. An arrangement for parsing packets in a packet-based digital communications network, said packets including packet headers divided into fields having values representing information pertaining to the packet, said arrangement comprising:

- (a) an input memory for receiving fields from said packet headers of incoming packets; and
- (b) a look-up engine for retrieving stored information appropriate to a received field value, said look-up engine including:
 - (i) a plurality of memories storing information related to possible values of said fields in a hierarchical tree structure and operating in parallel, said memories being associated with respective fields of said packet headers;
 - (ii) a main controller controlling overall operation of the look-up engine; and
 - (iii) a memory controller associated with each said respective memory for controlling the operation thereof to retrieve said stored information therefrom.

11. An arrangement as claimed in claim 10, wherein said main controller is a microcode.

12. An arrangement as claimed in claim 11, wherein said microcode controller comprises an interface memory for

receiving headers of incoming packets, a station information block memory for storing information pertaining to endstations, a microcode memory storing microcode instructions, and logic circuitry for implementing said microcode instructions.

13. An arrangement as claimed in claim 11, wherein said microcode controller parses the remainder of the packet header using a specific instruction set while said information is retrieved from said plurality of memories.

14. An arrangement as claimed in claim 13, wherein said microcode controller comprises separate buses for instructions and data.

15. An arrangement as claimed in claim 14, wherein said microcode controller is arranged to implement optimized instructions that perform bit level logical comparisons and conditional branches within the same cycle and other instructions tailored to retrieving data from nibble-indexed data structures.

16. An arrangement as claimed in claim 15, wherein said microcode controller is implemented as an ASIC processor.

17. An arrangement for parsing packets in a packet-based digital communications network, said packets including packet headers including destination and source address fields, said arrangement comprising:

- (a) an input memory for receiving fields from said packet headers of incoming packets; and
- (b) a look-up engine for retrieving stored information appropriate to a received field value, said look-up engine including:
 - (i) a source address look-up engine including a memory storing information related to possible values of said source address field in a hierarchical tree structure;
 - (ii) a memory controller associated with said source look-up engine for controlling the operation thereof to retrieve stored information therefrom;
 - (iii) a destination address look-up engine including a memory storing information related to possible values of said destination address field in a hierarchical tree structure;
 - (iv) a memory controller associated with said destination look-up engine for controlling the operation thereof to retrieve stored information therefrom;
 - (v) a processor controlling overall operation of said source and destination address look-up engines, said source and destination address look-up engines and said processor operating in parallel.

18. An arrangement as claimed in claim 17, wherein said processor is a microcode controller.

19. An arrangement as claimed in claim 18, wherein said memory controllers compare, at each decision point on the tree structure, the current field with a stored value or range, and move to the next decision point by moving a pointer for the current field and branching to new code if said comparison results in a first logical condition, and if said comparison results in a second logical condition, the current field is compared to a different value or range, and so on until said comparison results in said first logical condition.

20. An arrangement for parsing packets in a packet-based digital communications network, said packets including packet headers including destination and source address fields, said arrangement comprising:

- (a) an input memory for receiving fields from said packet headers of incoming packets; and
- (b) a look-up engine for retrieving stored information appropriate to a received field value, said look-up engine including:

(i) a source address look-up engine including a memory storing information related to possible values of said source field in a hierarchical tree structure;

(ii) a memory controller associated with said source look-up engine for controlling the operation thereof to retrieve stored information therefrom;

(iii) a destination address look-up engine including a memory storing information related to possible values of said destination field in a hierarchical tree structure and an associated memory controller;

(iv) a memory controller associated with said destination look-up engine for controlling the operation thereof to retrieve stored information therefrom; and

iii) a microcode processor controlling overall operation of said source and destination address look-up engine, said source and destination address look-up engines and said processor operating in parallel, and said microcode processor being arranged to parse additional fields in said packet header while said source and destination address look-up engines retrieve said related information.

21. An arrangement as claimed in claim 20 wherein said microcode processor comprises an interface memory for receiving said incoming packets, a station information block memory for storing information pertaining to endstations, a microcode memory storing microcode instructions, and logic circuitry for implementing said instructions.

22. A method of parsing packets in a packet-based digital communications network, said packets including packet headers divided into fields having values representing information pertaining to the packet, comprising the steps of:

- (a) receiving fields of packet headers from incoming packets in an input memory;
- (b) retrieving stored information appropriate to a received field value by performing a tree search in a look-up engine having at least one memory storing information related to possible values of said fields in a hierarchical tree structure and associated with a respective field of packet headers, said at least one memory being controlled by a memory controller associated therewith to retrieve said stored information therefrom; and
- (c) parsing a remaining portion of the packet header while said stored information is being retrieved from said at least one memory with a main controller, which main controller also controls the overall operation of said look-up engine.

23. A method as claimed in claim 22, wherein at each decision point in the tree search, in retrieving said information the current field is compared with a stored value or range, a pointer for the current field is moved and branched to new code if said comparison results in a first logical condition, and if said comparison results in a second logical condition, the current field is compared to a different value or range, and so on until said comparison results in said first logical condition.

24. A method as claimed in claim 22, wherein values based on successive nibbles of a field value are compared with stored values to locate the related information.

25. A method as claimed in claim 24, wherein a first nibble of an incoming field value is concatenated with a root pointer to obtain an index to a root pointer array, a word at a location identified by said index is retrieved, the next nibble is concatenated with the retrieved word to form the next pointer and so on until said related information is retrieved.

26. A method as claimed in claim 22, wherein information related to a plurality of fields is retrieved in parallel.

35

27. A method as claimed in claim 26, wherein one of said fields comprises a destination address and said related information comprises the path data associated with said respective destination address, and another of said fields comprises a source address and said related information comprises the path data associated with said source address. 5

28. A method of parsing packets in a packet-based digital communications network, said packets including packet headers divided into fields having values representing information pertaining to a packet, comprising the steps of: 10

- (a) storing in memory information related to possible values of said fields in a hierarchical tree structure;
- (b) receiving a plurality fields from said packet headers of incoming packets, one of said fields being a destination address and said related information therefor comprising path data associated with said respective destination address, and another of said fields being a source

36

address and said related information therefor comprising path data associated with said source address;

- (c) retrieving in parallel said stored information appropriate to received field values by performing a tree search under the control of a microcode controller; and
- (d) parsing a remaining portion of the packet header using a specific instruction set while said related information is retrieved.


29. An arrangement as claimed in claim 1, wherein said at least one memory provides table look-ups using nibble indexing for variable portions of the packet header and said microcode controller employs bit pattern recognition on fixed portions of the packet header for network layer protocol determination. 15

* * * * *

Our Docket/Ref. No.: APPT-0004

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al. Serial No.: 09/608266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Group Art Unit: 2731 Examiner:  TC 2600 MAILROOM APR 12 2001 RECEIVED
--	--

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:


Transmitted herewith are:

- An Information Disclosure Statement for the above referenced patent application, together with PTO form 1449 and a copy of each reference cited in form 1449.
- Return postcard.
- The commissioner is hereby authorized to charge payment of any missing fee associated with this communication or credit any overpayment to Deposit Account 50-0292.

A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED

Date: Apr 9, 2001


Respectfully submitted,



Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

Certificate of Mailing under 37 CFR 1.18	
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.	
Date of Deposit:	<u>Apr. 9, 2001</u>
Signature:	 Dov Rosenfeld, Reg. No. 38,687

Our Docket/Ref. No.: APP1-001-4

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al.
Serial No.: 09/608266
Filed: June 30, 2000
Title: ASSOCIATIVE CACHE
STRUCTURE FOR LOOKUPS AND
UPDATES OF FLOW RECORDS IN
A NETWORK MONITOR

Group Art Unit:

Examiner:



#5
4-22-02
RECEIVED

APR 17 2002

Technology Center 2600

Commissioner for Patents
Washington, D.C. 20231

INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

This Information Disclosure Statement is submitted:

under 37 CFR 1.97(b), or
(Within three months of filing national application; or date of entry of international application; or before mailing date of first office action on the merits; whichever occurs last)

Applicant(s) submit herewith Form PTO 1449-Information Disclosure Citation together with copies, of patents, publications or other information of which applicant(s) are aware, which applicant(s) believe(s) may be material to the examination of this application and for which there may be a duty to disclose in accordance with 37 CFR 1.56.

(Certification) Each item of information contained in this information disclosure statement was first cited in a formal communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this information disclosure statement (written opinion from PCT mailed Jan 11, 2002).

It is expressly requested that the cited information be made of record in the application and appear among the "references cited" on any patent to issue therefrom.

As provided for by 37 CFR 1.97(g) and (h), no inference should be made that the information and references cited are prior art merely because they are in this statement and no representation is

Certificate of Mailing under 37 CFR 1.18

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: 30 Mar 2002

Signature: 

Dov Rosenfeld, Reg. No. 38,687

S/N: 09/608266


Page 2

IDS

being made that a search has been conducted or that this statement encompasses all the possible relevant information.

Date: 30 Mar 2002

Respectfully submitted,

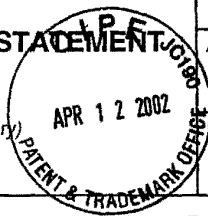


Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

<p>INFORMATION DISCLOSURE STATEMENT</p> <p><i>(Use several sheets if necessary)</i></p>	<p>ATTY. DOCKET NO. APPT-001-4</p>	<p>SERIAL NO. 09/608266</p>
<p>APPLICANT Sarkissian et al.</p>		<p>RECEIVED APR 17 2002 Technology Center 2600</p>
<p>FILING DATE 6/30/2000</p>		<p>GROUP 2667</p>



U.S. PATENT DOCUMENTS

EXAMINER INITIAL	DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
AV	AA	5,703,877	Dec. 30, 1997, Nuber et al.	370	395	Nov. 22, 1995
AV	AB	5,835,963	Nov. 10, 1998, Yoshioka et al.	711	207	Sep. 7, 1995
L	AC	5,860,114	Jan. 12, 1999, Sell	711	146	Oct. 1, 1997
	AD					
	AE					
	AF					
	AG					
	AH					
	AI					
	AJ					
	AK					
	AL					
	AM					
	AN					

FOREIGN PATENT DOCUMENTS

DOCUMENT NUMBER	PUBLICATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES I NO
AO					

OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

AP	
----	--

EXAMINER <i>[Signature]</i>	DATE CONSIDERED 9/2/03
--------------------------------	---------------------------

*EXAMINER: initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

Oür Docket/Ref. No.: APP-001-4

Patent

2731
2664

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al.

Serial No.: 09/608266

Filed: June 30, 2000

Title: ASSOCIATIVE CACHE
STRUCTURE FOR LOOKUPS AND
UPDATES OF FLOW RECORDS IN
A NETWORK MONITOR

Group Art Unit: 2731

Examiner:



RECEIVED

APR 17 2002

Technology Center 2600

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

- An Information Disclosure Statement for the above referenced patent application, together with PTO form 1449 and a copy of each reference cited in form 1449.
- A check for petition fees.
- Return postcard.
- The commissioner is hereby authorized to charge payment of any missing fee associated with this communication or credit any overpayment to Deposit Account 50-0292.

A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED

Date: 30 Mar 2002

Respectfully submitted,

Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

Certificate of Mailing under 37 CFR 1.18

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.


Date of Deposit: 30 Mar 2002

Signature:
Dov Rosenfeld, Reg. No. 38,687

Our Docket/Ref. No.: APP 001-4

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al. Serial No.: 09/608266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Group Art Unit: 2731 Examiner:  RECEIVED APR 17 2002 Technology Center 2600
--	---

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:


Transmitted herewith are:

- An Information Disclosure Statement for the above referenced patent application, together with PTO form 1449 and a copy of each reference cited in form 1449.
- A check for petition fees.
- Return postcard.
- The commissioner is hereby authorized to charge payment of any missing fee associated with this communication or credit any overpayment to Deposit Account 50-0292.

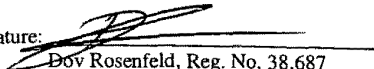
A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED

Respectfully submitted,

Date: 30 Mar 2002


Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

Certificate of Mailing under 37 CFR 1.18	
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.	
Date of Deposit: <u>30 Mar 2002</u>	Signature:  Dov Rosenfeld, Reg. No. 38,687



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
PO Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO	CONFIRMATION NO.
09/608,266	06/30/2000	Haig A. Sarkissian	APPT-001-4	9867

7590 09/10/2003
Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER

NGUYEN, ALAN V

ART UNIT PAPER NUMBER

2662

DATE MAILED: 09/10/2003

6

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No. 09/608,266	Applicant(s) SARKISSIAN ET AL	
Examiner Alan Nguyen	Art Unit 2662	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on _____.
- 2a) This action is FINAL. 2b) This action is non-final.
- 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1-20 is/are pending in the application.
4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) Claim(s) _____ is/are allowed.
- 6) Claim(s) 1-20 is/are rejected.
- 7) Claim(s) _____ is/are objected to.
- 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
- 10) The drawing(s) filed on 06/30/2000 is/are: a) accepted or b) objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- 11) The proposed drawing correction filed on _____ is: a) approved b) disapproved by the Examiner.
If approved, corrected drawings are required in reply to this Office action.
- 12) The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

- 13) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
a) All b) Some * c) None of:
1. Certified copies of the priority documents have been received.
2. Certified copies of the priority documents have been received in Application No. _____.
3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
* See the attached detailed Office action for a list of the certified copies not received.
- 14) Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).
a) The translation of the foreign language provisional application has been received.
- 15) Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

- 1) Notice of References Cited (PTO-892)
- 2) Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) Information Disclosure Statement(s) (PTO-1449) Paper No(s) 4 & 5.
- 4) Interview Summary (PTO-413) Paper No(s). _____.
- 5) Notice of Informal Patent Application (PTO-152)
- 6) Other:

DETAILED ACTION

Specification

1. The disclosure is objected to because of the following informalities: The serial numbers of related applications are missing on pages 1 and 2 of the specifications.

Appropriate correction is required.

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

(e) the invention was described in a patent granted on an application for patent by another filed in the United States before the invention thereof by the applicant for patent, or on an international application by another who has fulfilled the requirements of paragraphs (1), (2), and (4) of section 371(c) of this title before the invention thereof by the applicant for patent.

The changes made to 35 U.S.C. 102(e) by the American Inventors Protection Act of 1999 (AIPA) and the Intellectual Property and High Technology Technical Amendments Act of 2002 do not apply when the reference is a U.S. patent resulting directly or indirectly from an international application filed before November 29, 2000. Therefore, the prior art date of the reference is determined under 35 U.S.C. 102(e) prior to the amendment by the AIPA (pre-AIPA 35 U.S.C. 102(e)).

3. Claims 7-11, 19, and 20 rejected under 35 U.S.C. 102(b) as being anticipated by Chang (US 4,458,310).

Regarding claims 7 and 19, Chang clearly describes a cache memory system shown in figure 1 element 100 that utilizes a number of content addressable memory (CAMs). The cache system is coupled to a processor and main memory as, clearly shown in Figure 1 elements 101 and 102 of Chang. Figure 1 further shows the use of LRU (least recently used) circuits (elements 104-106), each coupled to cache data memory (elements 107-109). Figure 2 shows the use of a CAM in each LRU circuit (a CAM controller coupled to the CAM set). Reverting to figure 1, elements 104-106 clearly show a top LRU circuit connected to a middle LRU circuit, which is connected to a bottom LRU circuit. Chang shows in figure 1 a control and sequencer device (element 103) that is coupled to the LRU circuit controlling the CAM, main memory, and the cache data memory. Chang further explains the function of the LRU circuit/CAM and its corresponding cache data memory in column 4 lines 13-20 and column 5 lines 26-33. The CAM responds to the input of the address being received and compares that address to the contents stored in the CAM. If there is a match, indicating a hit, the LRU circuit uses that address to point to the cache data memory for accessing. In addition to checking if the associated cache data has the desired word, the LRU circuit maintains the priority of each word in the associated cache data memory, this priority information is automatically updated by the LRU circuit for each access to the associated cache data memory and defines which word in the cache memory is the least recently used word. Chang also discloses repeatedly how the address of each new, least recently used word is written into the CAM. Since each CAM will contain addresses that are

constantly changing being written into it, the CAM will therefore point to a different address in the cache memory element.

In regards to claim 8, with the features in parent claim 7 addressed above, Chang further discloses a deletion of the least recently used word in column 4 lines 48-51. It is stated that the least recently used word of cache data memory 109 no longer exists in cache memory 100 at the completion of the previous operation after the values have been shifted down from data memory 107.

In regards to claim 9, with the features in parent claim 7 addressed above, Chang further discloses an example of a hit, shown in column 9 lines 50-62 and figure 1. LRU circuit 104 and data memory 107 are the priority CAM and cache memory, respectively. LRU circuit 105 and memory 108 are the next highest priority. The contents of the match/hit are transmitted and stored within LRU circuit 104 and data memory 107. The least recently used words from LRU circuit 104 and memory 107 are transmitted to LRU circuit 105 and data memory 108. The steps above explain the shifting-down process of the least recently used value. The bottom CAM (LRU circuit 106) will always point to the least recently used value in the device.

In regards to claim 10, with the features in parent claim 7 addressed above, Chang discloses a deletion of the least recently used word in column 4 lines 48-51. It is stated that the least recently used word of cache data memory 109 no longer exists in cache memory 100 at the completion of the previous operation after the values have been shifted down from data memory 107. As the replacement process keeps going,

shifting of values also continues. This deducts to the replacing of values at the bottom of the list, which is according to an inverse order of recentness of use.

In regards to claim 11, with the features in parent claim 7 addressed above, it is understood that cache data memory (figure 1 elements 107-109) contains cells of words and can be a page of memory.

In regards to claim 20, with the features in parent claim 19 addressed above, Chang further discloses the use of least recently used (LRU) cache memory element. Chang discloses in column 4 lines 42-48 an example of a new word placed in cache data memory (element 107). The LRU word of memory 107 is then shifted down to cache memory (element 108) and the LRU word of memory 108 is written to cache memory 109. The address of that LRU word is then written to the CAM (element 106) associated with memory 109, as described in column 5 lines 49-51, and shown in Figure 1. Therefore LRU circuit 106 is understood to be the bottom CAM of figure 1 and points to the least recently used value stored in cache memory 109.

4. Claims 1 and 2 rejected under 35 U.S.C. 102(e) as being anticipated by Gobuyan et al (US 5,917,821), herein Gobuyan.

Regarding claim 1, Gobuyan discloses an apparatus that examines packets through a connection point on a network. This indicates that the apparatus has a device for acquiring packets. Gobuyan shows in figure 3 a device with a lookup engine (element 3), memory for storage of the entries (elements 6, 8), and a subsystem accessing the memory (elements 5 and 7). In column 7 lines 41-43 and 56-59, Gobuyan

discloses that the lookup engine receives portions of packets containing identifying information through a 16-bit I/F RAM (element 9). Regarding claim 2, the apparatus of Gobuyan inherently includes a parser that extracts packets identifying information because this operation is necessary for the lookup engine to operate.

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

4. Claim 3-6 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gobuyan in view of Chang (US 4,458,310).

(a) Regarding claims 3 and 4, Gobuyan discloses the use of a subsystem that accesses the database memory to search for the stored information. The lookup engine invokes the address lookup engines (ALE) to search for the specified address in its bank of memory.

(b) Gobuyan fails to teach the use and function of content addressable memory (CAM) as a method to search for specified data fields.

(c) Chang teaches the use of a cache memory system that utilizes a set of CAMs. The cache system is coupled to a processor and main memory as, clearly shown in Figure 1 of Chang. Figure 1 further shows the use of LRU (least recently used) circuits (elements 104-106), each coupled to cache data

memory (elements 107-109). Figure 1 further shows a control and sequencer device (element 103) that is coupled to the LRU circuits. Figure 2 shows the use of a CAM in each LRU circuit (a CAM controller coupled to the CAM set). Claim 3 is therefore rejected since Chang indicates the use of CAMs for the cache subsystem. Reverting to figure 1, elements 104-106 clearly show a top LRU circuit connected to a middle LRU circuit, which is connected to a bottom LRU circuit. Chang shows in figure 1 a control and sequencer device (element 103) that is coupled to the LRU circuit controlling the CAM, main memory, and the cache data memory. Chang further explains the function of the LRU circuit/CAM and its corresponding cache data memory in column 4 lines 13-20 and column 5 lines 26-33. The CAM responds to the input of the address being received and compares that address to the contents stored in the CAM. If there is a match, indicating a hit, the LRU circuit uses that address to point to the cache data memory for accessing. In addition to checking if the associated cache data has the desired word, the LRU circuit maintains the priority of each word in the associated cache data memory, this priority information is automatically updated by the LRU circuit for each access to the associated cache data memory and defines which word in the cache memory is the least recently used word. Chang also discloses repeatedly how the address of each new, least recently used word is written into the CAM. Since each CAM will contain addresses that are constantly

changing being written into it, the CAM will therefore point to a different address in the cache memory element.

- (d) It would have been obvious to one having ordinary skill in the art at the time the invention was made for Gobuyan's arrangement to have a cache memory subsystem utilizing a stack of CAMs for looking up address fields, the motivation being improved performance through quicker execution and accessing, as taught by Chang.

In regards to claim 5, with the features in parent claim 4 addressed above, Gobuyan fails to disclose the use of CAMs utilizing a least recently used scheme. Chang teaches the use of least recently used (LRU) cache memory element. Chang discloses in column 4 lines 42-48 an example of a new word placed in cache data memory (element 107). The LRU word of memory 107 is then shifted down to cache memory (element 108) and the LRU word of memory 108 is written to cache memory 109. The address of that LRU word is then written to the CAM (element 106) associated with memory 109, as described in column 5 lines 49-51, and shown in Figure 1. Therefore LRU circuit 106 is understood to be the bottom CAM of figure 1 and points to the least recently used value stored in cache memory 109. It would have been obvious to one having ordinary skill in the art at the time the invention was made for Gobuyan to use a cache subsystem having CAMs to utilize a lowest priority word scheme, the motivation being a much faster lookup time of data fields, as taught by Chang.

In regards to claims 6, with the features in parent claim 4 addressed above, Gobuyan fails to disclose a CAM scheme that shifts down content due to a more

recently used value. Chang teaches an example of a cache hit, shown in column 9 lines 50-62 and figure 1. LRU circuit 104 and data memory 107 are the priority CAM and cache memory, respectively. LRU circuit 105 and memory 108 are the next highest priority. The contents of the match/hit are transmitted and stored within LRU circuit 104 and data memory 107. The least recently used words from LRU circuit 104 and memory 107 are transmitted to LRU circuit 105 and data memory 108. The steps above explain the shifting-down process of the least recently used value. The bottom CAM (LRU circuit 106) will always point to the least recently used value in the device.

It would have been obvious to one having ordinary skill in the art at the time the invention was made for Gobuyan to use a cache subsystem having CAMs utilizing a LRU element pointed to by the bottom CAM for faster accessing of data fields, as taught by Chang

5. Claims 12-18 rejected under 35 U.S.C. 103(a) as being unpatentable over Chang in view of Carter et al (US 6,003,123), herein Carter.

- (a) Regarding claims 12, 13, 14, 15, 16, and 17, Chang discloses the use of a cache system having content addressable memory as a way of looking up specified addresses quickly.
- (b) Chang fails to disclose a method to indicate dirty entries in the cache. A dirty entry is one that has not been updated by an external memory.
- (c) Carter teaches the use of labeling elements as being dirty or not dirty. Carter discloses in column 15 lines 12-17 the use setting bits as "dirty" to allow

hardware to determine if the block has been modified. Carter further explains that the dirty bit of a block status in the cache is always set to zero when the block is brought into the cache to reflect the fact that the block has not been modified since it was brought into the cache. Carter also discloses that if the block is cleaned, the status remains at zero. When a block is evicted from the cache, its dirty bit is examined, and the status of the block changed to dirty if the cache dirty bit is set to one. When an entry is evicted, its block status bits are copied to the local page table. This is analogous to the address being written to the main memory in Chang's apparatus.

- (d) It would have been obvious to one having ordinary skill in the art at the time the invention was made for Chang to modify the arrangement such that the use of setting dirty flags to determine if the cache has been modified or not, the motivation being the prevention of contamination of data. Each cache memory element would have an indication of whether or not it is dirty. If the cache element is cleaned the status remains at zero.

In regards to claims 18, with the features in parent claim 17 addressed above, For Chang's apparatus, it inherently cleans the least recently used cache data first because the apparatus does use the LRU scheme. The concept of lowest word priority is to flush out the least used word.

Conclusion

6. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

The following patents are cited to further show the state of the art with respect to associative cache memory and content addressable memory:

Colloff et al (US 5,530,834)
Hoover et al (US 5,749,087)
Churchill (US 3,949,369)
Houseman et al (US 4,559,618)
Okamoto et al (US 4,910,668)
Agarwal et al (US 5,530,958)
Inoshita et al (JP 2003044510A)

7. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Alan Nguyen whose telephone number is 703-305-0369. The examiner can normally be reached on 8am-5pm ET.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hassan Kizou can be reached on 703-305-4744. The fax phone numbers for the organization where this application or proceeding is assigned are 703-872-9314 for regular communications and 703-872-9314 for After Final communications.

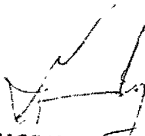
Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-4700.

Application/Control Number: 09/608,266
Art Unit: 2662

Page 12

an

September 3, 2003



HASSAN KIZOU
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2600

Notice of References Cited

Application/Control No. 09/608,266	Applicant(s)/Patent Under Reexamination SARKISSIAN ET AL.	
Examiner Alan Nguyen	Art Unit 2662	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-5,530,958	06-1996	Agarwal et al.	711/3
*	B	US-4,458,310	07-1984	Chang, Shih-Jeh	711/119
*	C	US-6,003,123	12-1999	Carter et al.	711/207
*	D	US-5,530,834	06-1996	Colloff et al.	711/136
*	E	US-5,749,087	05-1998	Hoover et al.	711/108
*	F	US-3,949,369	04-1976	Churchill, Jr., William Philip	711/128
*	G	US-4,559,618	12-1985	Houseman et al.	365/49
*	H	US-4,910,668	03-1990	Okamoto et al.	711/207
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
*	N	JP02003044510A	02-2003	JP	Inoshita et al	G06F017/30
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



US005530958A

United States Patent [19]
Agarwal et al.

[11] **Patent Number:** **5,530,958**
[45] **Date of Patent:** **Jun. 25, 1996**

- [54] **CACHE MEMORY SYSTEM AND METHOD WITH MULTIPLE HASHING FUNCTIONS AND HASH CONTROL STORAGE**
- [75] Inventors: **Anant Agarwal**, Framingham, Mass.;
Steven D. Pudar, Rancho Cordova, Calif.
- [73] Assignee: **Massachusetts Institute of Technology**, Cambridge, Mass.
- [21] Appl. No.: **363,542**
- [22] Filed: **Dec. 23, 1994**

Related U.S. Application Data

- [63] Continuation of Ser. No. 926,613, Aug. 7, 1992, abandoned.
- [51] Int. Cl.⁶ **G06F 12/10; G06F 12/08**
- [52] U.S. Cl. **395/403; 395/421.06; 395/435; 395/460; 364/DIG. 1; 364/243.41; 364/244.7; 364/255.8; 364/259.8**
- [58] Field of Search **395/421.06, 403, 395/435, 460**

References Cited

U.S. PATENT DOCUMENTS

5,235,697 8/1993 Steely, Jr. et al. 395/425

FOREIGN PATENT DOCUMENTS

2154106 5/1972 Germany .

OTHER PUBLICATIONS

- Agarwal, "Analysis of Cache Performance for Operating Systems and Multiprogramming," Technical Report No. CSL-TR-87-332, Computer Systems Laboratory, Stanford University (May 1987).
- Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," Proc. of the IEEE (1990).
- Agarwal, Anant, "Analysis of Cache Performance for Operating Systems and Multiprogramming," Kluwer Academic Publishers, Boston, MA, Title page, Contents pp. vi-ix, pp. 120-124, see p. 122, line 14-p. 124, line 2.

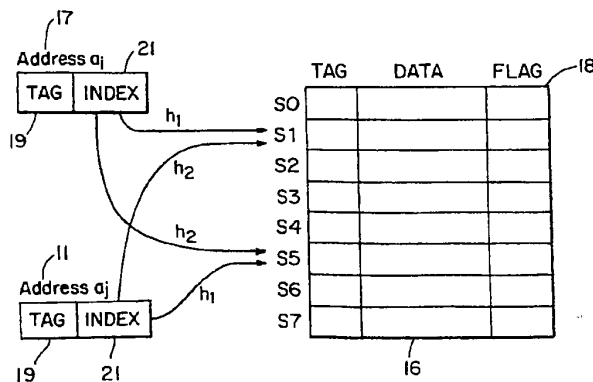
- Kessler, et al., "Inexpensive Implementations of Set-Associativity," *Computer Architecture News* 17(3): 131-139 (Jun. 1989).
- da Silva, et al., "Pseudo-associative Store with Hardware Hashing," *IEE Proceedings E. Computers & Digital Techniques* 130(1): 19-24 (Jan. 1983).
- Anant Agarwal and Steven D. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches." In *Proceeding ISCA 1993* (Abstract).
- Anant Agarwal et al., "Cache Performance of Operating System and Multiprogramming Workloads," *ACM Transactions on Computer Systems*, 6(4): 393-431, Nov., 1988.
- Anant Agarwal et al., "An Analytical Cache Model," *ACM Transactions on Computer Systems*, 7(2): 184-215, May, 1989.
- Kimming So and Rudolph N. Rechtschaffen, "Cache Operations by MRU Change," (Research Report #RC11613 (#51667) Computer Science, pp. 1-19, (Nov. 13, 1985). Yorktown Heights, NY: IBM T. J. Watson Research Center.
- "A High Performance Memory Management Scheme"; Thakkar, Shreekant S. and Knowles, Alan E.; Computer; May 1986; IEEE Computer Society; pp. 8-20.

Primary Examiner—Eddie P. Chan
Assistant Examiner—Reginald G. Bragdon
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds

[57] **ABSTRACT**

A column-associative cache that reduces conflict misses, increases the hit rate and maintains a minimum hit access time. The column-associative cache indexes data from a main memory into a plurality of cache lines according to a tag and index field through hash and rehash functions. The cache lines represent a column of sets. Each cache line contains a rehash block indicating whether the set is a rehash location. To increase the performance of the column-associative cache, a content addressable memory (CAM) is used to predict future conflict misses.

25 Claims, 7 Drawing Sheets



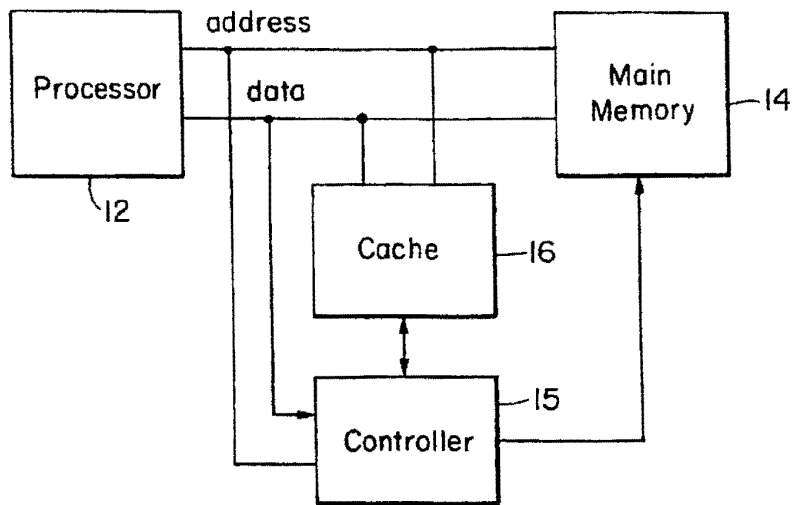


Fig. 1

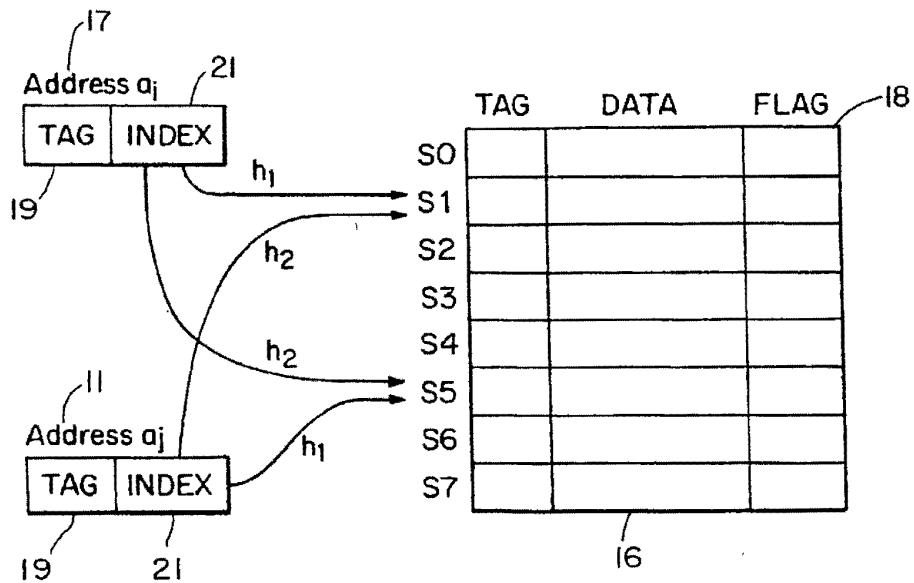


Fig. 2A

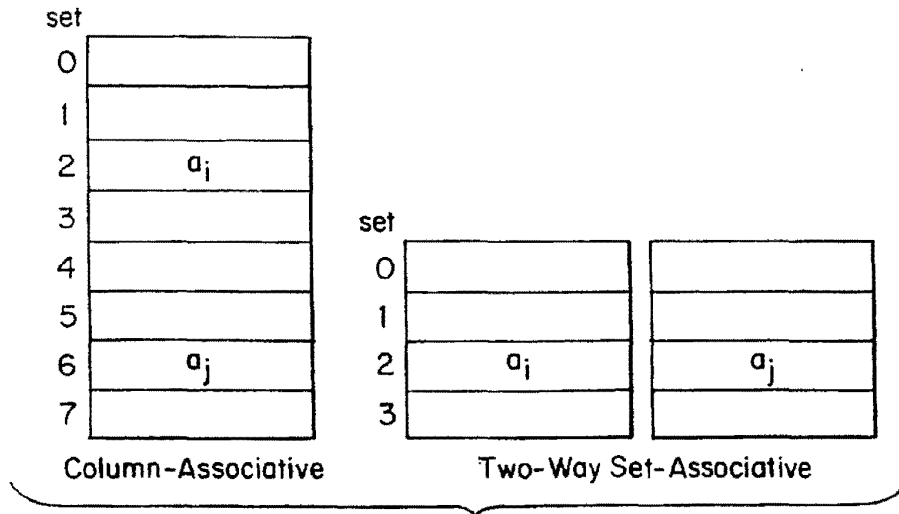


Fig. 2B

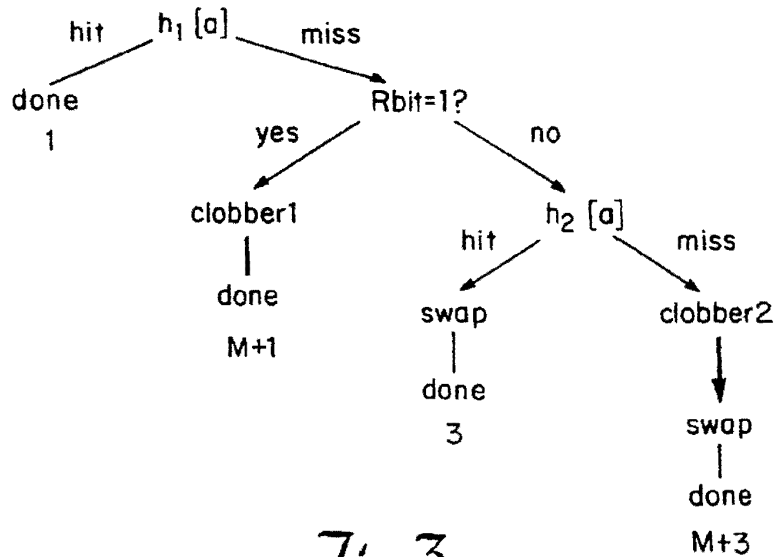


Fig. 3

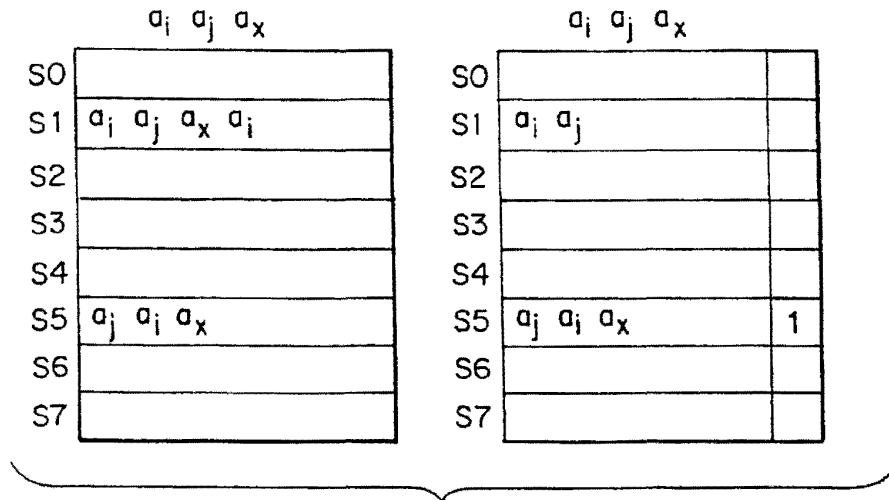


Fig. 4

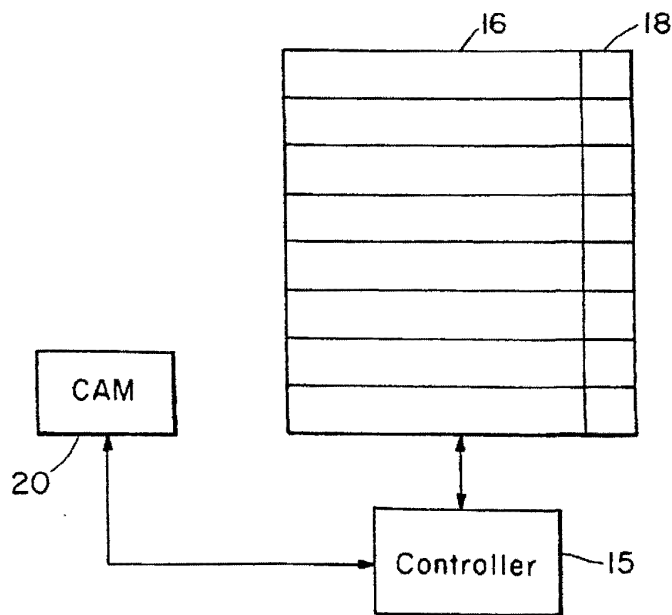


Fig. 5

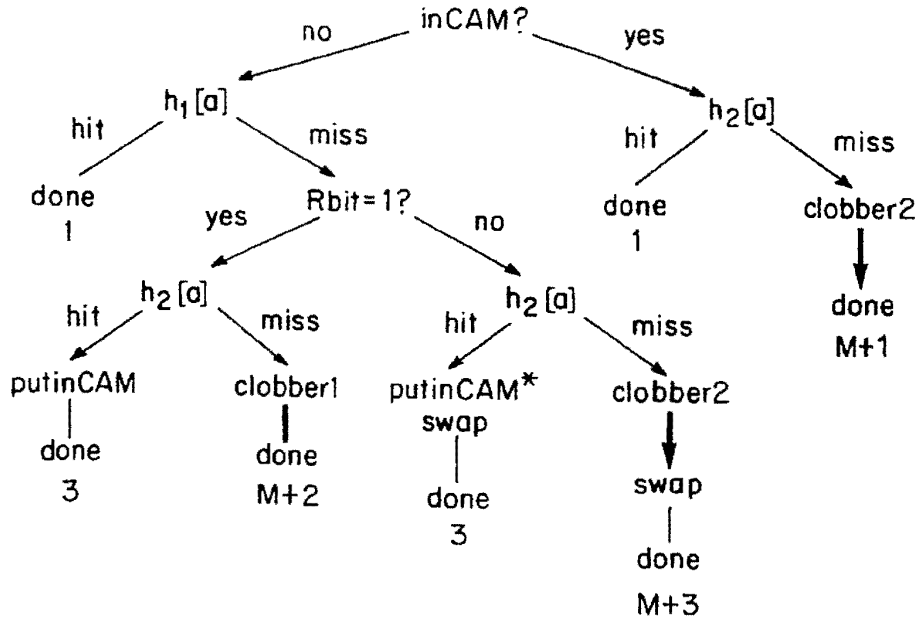


Fig. 6

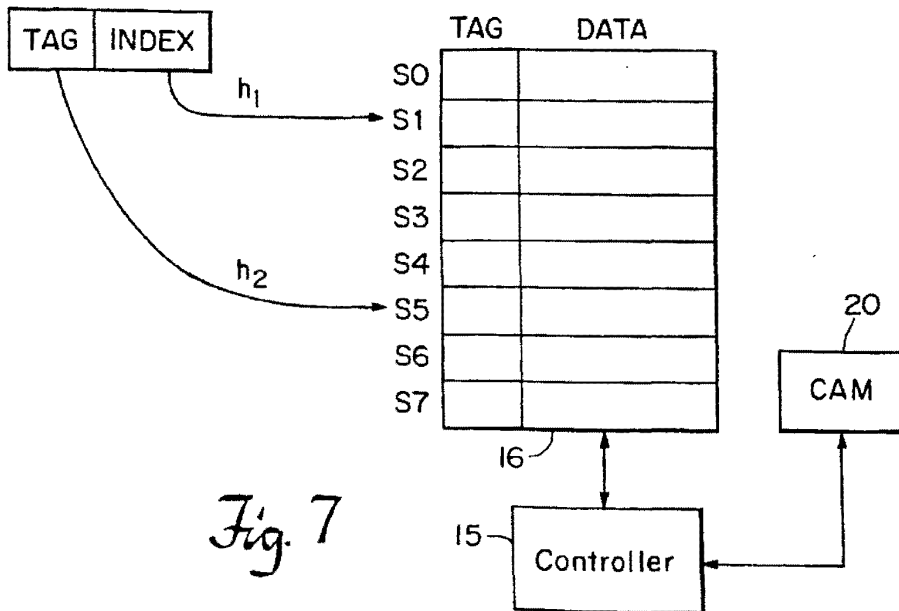


Fig. 7

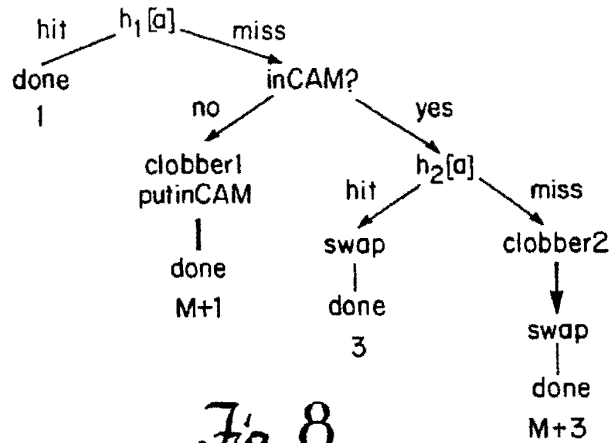


Fig. 8

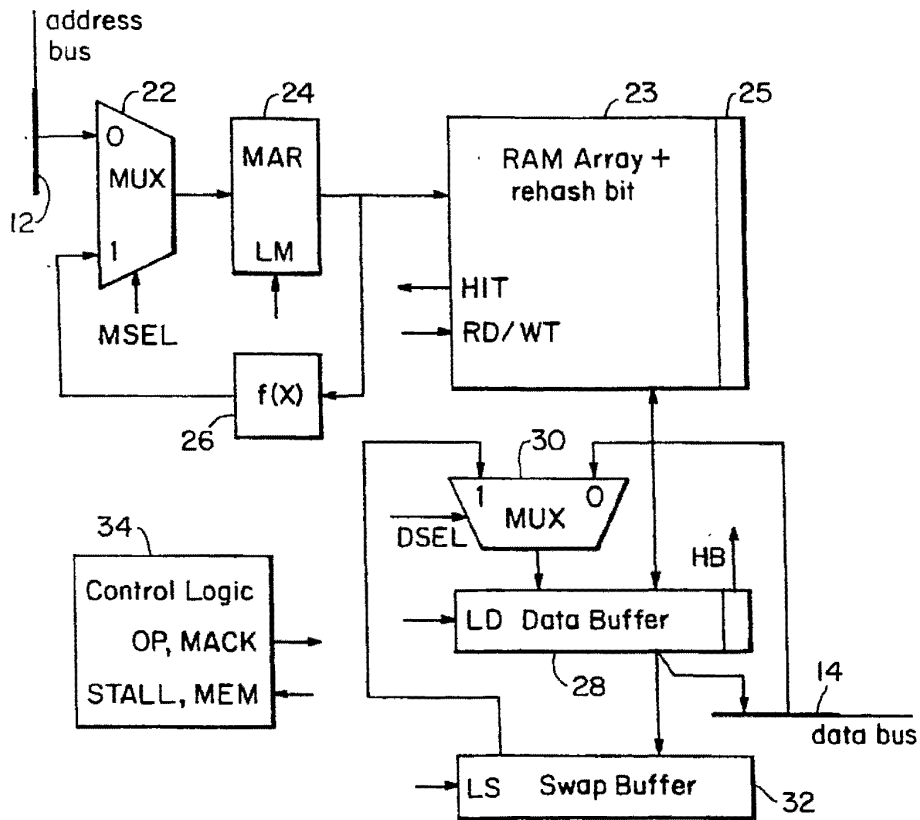


Fig. 9

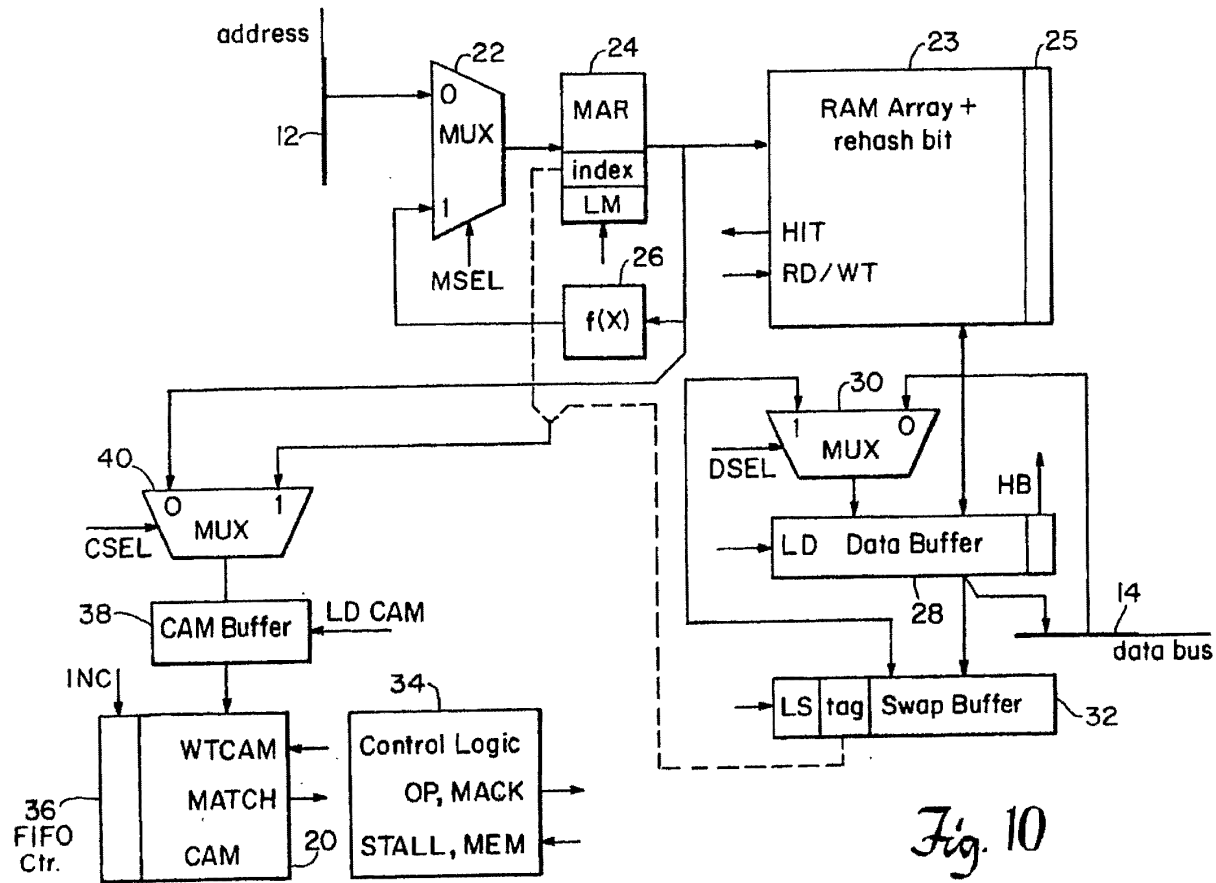


Fig. 10

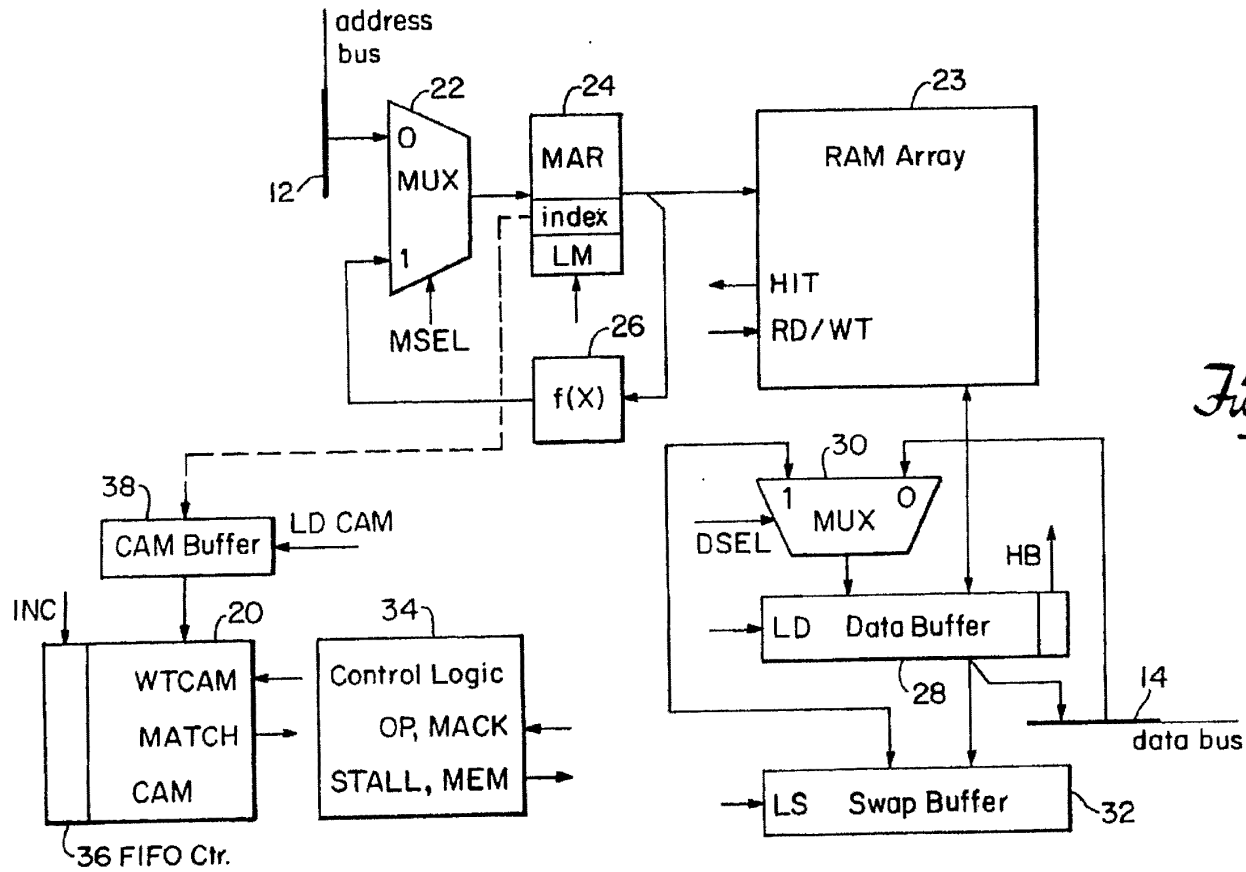


Fig. 11

**CACHE MEMORY SYSTEM AND METHOD
WITH MULTIPLE HASHING FUNCTIONS
AND HASH CONTROL STORAGE**

This application is a continuation of No. 07/926,613 filed 5
Aug. 7, 1992, now abandoned.

BACKGROUND OF THE INVENTION

This invention relates generally to the field of high 10
performance processors that require a large bandwidth to
communicate with a main memory system. To effectively
increase the memory bandwidth, a cache memory system is
typically placed between the processor and the main
memory. The cache memory system stores frequently used 15
instructions and data in order to provide fast access from the
main memory.

In order for a processor to access memory, it checks the
cache first. If the desired data is in the cache, a cache hit
occurs, and the processor receives the data without further 20
delay. If the data is not in the cache, a cache miss occurs, and
the data must be retrieved from the main memory to be
stored in the cache for future use. Main memory accesses
take longer than cache accesses, so the processor is stalled
in a cache miss, wasting a number of cycles. Thus, the goal 25
for nearly all modern computer systems is to service all
memory references from the cache and to minimize refer-
ences which require accesses from the main memory.

In a typical cache system, a portion of a main memory
address is used to index a location or a set of locations in
cache memory. In addition to storing a block (or line) of data
at that indexed location, cache memory stores one or more
tags, taken from another portion of the main memory
address, which identify the location in main memory from 30
which the block of data held in cache was taken.

Caches are typically characterized by their size (i.e.,
amount of memory available for storage), their replacement
algorithm (i.e., method of inserting and discarding blocks of
data into a set), their degree of associativity or set size (i.e., 40
number of tags associated with an index and thus the number
of cache locations where data may be located), and their
block or line size (i.e., number of data words associated with
a tag). These characteristics influence many performance
parameters such as the amount of silicon required to imple- 45
ment the cache, the cache access time, and the cache miss
rate.

One type of a cache that is frequently used with modern
processors is a direct-mapped cache. In a direct-mapped
cache, each set contains only one data block and tag. Thus, 50
only one address comparison is needed to determine whether
the requested data is in the cache. The direct-mapped cache
is simple, easy to design, and requires less chip area.
However, the direct-mapped cache is not without draw-
backs. Because the direct-mapped cache allows only one
data block to reside in the cache set, its miss rate tends to be
very high. However, the higher miss rate of the direct-
mapped cache is mitigated by a small hit access time.

Another type of a cache that is frequently used is a d-way,
set associative cache. A d-way, set associative cache con- 60
tains S sets of d distinct blocks of data that are accessed by
addresses with common index fields that have different tag
fields. For each cache index, there are several block loca-
tions allowed, one in each set. Thus, a block of data arriving
from the main memory can go into a particular block 65
location of any set. The d-way set associative cache has a
higher hit rate than the direct-mapped cache. However, its

hit access time is also higher because an associative search
is required during each reference, followed by a multiplex-
ing of the data block to the processor.

Currently, the trend among computer designers is to use
direct-mapped caches rather than d-way set associative
caches. However, as mentioned previously, a major problem
associated with direct-mapped caches is the large number of
misses that occur. One particular type of miss that occurs is
a conflict miss. A conflict miss occurs when two addresses
map into the same cache set. This situation occurs when the
addresses have identical index fields but different tags.
Therefore, the addresses reference the same set. A d-way set
associative cache typically does not suffer from conflict
misses because the data can co-reside in a set. Although
other types of misses, such as compulsory (misses that occur
when loading a working set into a cache) and capacity
(misses that occur when the cache is full and when the
working set is larger than the cache size) do occur, they tend
to be minimal as compared to conflict misses.

The problem of conflict misses has caused designers to
reconsider using a direct-mapped cache and to begin design-
ing cache memory systems that can incorporate the advan-
tages of both the direct-mapped cache and the d-way asso-
ciative cache. One approach has been to use a victim cache.
A victim cache is a small, fully associative cache that
provides some extra cache lines for data removed from the
direct-mapped cache due to misses. Thus, for a reference
stream of conflicting addresses $a_1, a_2, a_3, a_4, \dots$, the second
reference a_2 misses and forces the data i indexed by a_1 out of
the set. The data i that is forced out is placed in the victim
cache. Thus, the third reference address, a_3 , does not require
accessing main memory because the data is in the victim
cache and can be accessed therefrom.

However, there are several drawbacks to the victim cache.
For example, the victim cache must be very large to attain
adequate performance because it must store all conflicting
data blocks. Another problem with the victim cache is that
it requires at least two access times to fetch a conflicting
datum (i.e., one to check the primary cache, the second to
check the victim cache, and maybe a possible third to store
the datum in the primary cache). Still another drawback to
the victim cache is that performance is degraded as the size
of the cache memory is increased because the victim cache
becomes smaller relative to the cache memory, thereby
reducing the probability of resolving conflicts.

Consequently, there is a need for an improved cache
memory system that incorporates the low conflict miss rate
of the d-way set-associative cache, maintains the critical
access path of the direct-mapped cache, and has better
performance than the victim cache.

SUMMARY OF THE INVENTION

To provide a cache memory system with a high hit rate
and a low hit access time, the present invention has set forth
a column associative cache that uses an area-efficient cache
control algorithm. A column associative cache removes
substantially more conflict misses introduced by a direct-
mapped access for small caches and virtually all of those
misses for large caches. Also, there is a substantial improve-
ment in the hit access time.

In accordance with the present invention, there is a cache
memory having a plurality of cache sets representing a
column of sets for storing data. Each cache set is indexed by
memory addresses having a tag field and an index field. A
controller indexes memory addresses to the cache data

memory by applying at least one hashing function. A hashing function is an operation that maps the addresses of the data from a main memory to the cache sets of the cache data memory. A rehashed location stores data that is referenced by an alternate hashing function. The use of alternative hash functions (i.e., hash and rehash) allows cache sets associated with a common index to be stored within the single cache column rather than in separate columns, each of which requires its own memory space. For example, in a direct-mapped cache, the two hash functions allow two blocks with the same index to reside in different cache locations. In accordance with the present invention, hash control data is stored in the cache memory to direct the cache system to a hashed location or a rehashed location based on past cache operations. The hash control data may be a hash/rehash block associated with each cache location which indicates whether the hash or rehash function was used to store the data in that location. Alternatively, or in combination with the hash/rehash block, a memory may identify recent cache indexes or groups of indexes which have required rehash.

The cache memory system of the present invention resolves conflict misses that arise in direct-mapped cache access by allowing conflicting addresses to dynamically choose alternate hashing functions, so that most conflicting data can reside in the cache. In the cache memory system of the present invention, data is accessed from the cache by applying a first hashing function to the indexed memory address. If the data is valid, it is a hit and is subsequently retrieved. For a miss at a rehashed location, as indicated by a rehash block, the controller removes that data and replaces it with new data from the main memory. If the cache location is not a rehashed location, then a second hashing function is applied in order to place or locate the data in a different location. With a second miss, valid data is accessed and the controller swaps the data in the cache locations indexed by the first and second hashing functions.

The preferred first type of hashing function used by the present invention is a bit selection operation. The bit selection operation indexes the data in the cache lines according to the index field. If there is a conflict miss, then the second hashing function is applied. The preferred second hashing function of the present invention is a bit flipping operation. The bit flipping operation inverts the highest order bit of the index field of the address and accesses the data in that particular location. The present invention is not limited to two hashing functions and may use more.

In another preferred embodiment of the present invention, there is provided a content addressable memory (CAM) coupled to the cache memory system for storing portions of addresses that are expected to indicate future conflict misses in the cache. The CAM, preferably a tag memory, improves the efficiency of the cache by increasing the first time hit rate. The CAM stores the indexes of cache blocks that are present in rehashed locations. If the index of an address matches an index stored in the CAM, then the cache controller uses the rehash function (instead of the hash function) for the first time access. Thus, second time accesses are reduced.

While the present invention will hereinafter be described in connection with a preferred embodiment and method of use, it will be understood that it is not intended to limit the invention to this embodiment. Instead, it is intended to cover all alternatives, modifications, and equivalents as may be included in the spirit and scope of the present invention as defined by the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a block diagram of a cache memory system of the present invention.

FIG. 2A illustrates a column associative cache with rehash blocks.

FIG. 2B illustrates a comparison of a column associated cache and two-way set associative cache.

FIG. 3 shows a decision tree for the column associative cache with rehash blocks.

FIG. 4 shows a comparison between a single column associative cache and the column associative cache with rehash blocks.

FIG. 5 shows a column associative cache with a content addressable memory (CAM) and rehash blocks.

FIG. 6 shows a decision tree for a column associative cache with rehash blocks and a CAM.

FIG. 7 shows a column associative cache with a CAM.

FIG. 8 shows a decision tree for a column associative cache with a CAM.

FIG. 9 shows the circuitry for a column associative cache with rehash blocks.

FIG. 10 shows the circuitry for a column associative cache with rehash blocks and a CAM.

FIG. 11 shows the circuitry for a column associative cache with a CAM.

DETAILED DESCRIPTION OF THE INVENTION

Referring to FIG. 1 of the present invention, there is shown a cache memory system 10 placed between a processor 12 and a main memory 14. The speed of the cache is compatible with the processor, whereas the main memory is lower in speed. The cache anticipates the processor's likely use of data in the main memory based on previously used instructions and data in the cache. Based on an assumption that a program will sequence through successive instructions or data addresses, a block or line of several words from the main memory is transferred to the cache even though only one word is needed. When the processor needs to read from main memory the cache is checked first. If the data is in the cache, there is a hit and retrieval from cache. If the data is not in the cache, there is a miss and retrieval is from main memory.

To provide a cache memory system with a high hit rate and a low access time, the present invention has set forth a cache that incorporates the characteristics of a direct-mapped cache and a d-way set associative cache. The cache of the present invention is a column associative cache 16 and is shown in FIG. 2A. The column associative cache contains a plurality of cache lines that represent a column of sets each of one line. In FIG. 2A, eight sets, S0-S7 of the cache are shown. It is noted that the column associative cache would likely have hundreds or thousands of sets.

To access the cache 16, a memory address 17 is divided into at least two fields, a tag field 19 (typically the high-order bits) and an index field 21. As in a conventional direct mapped cache, the index field is used through a hash function h_1 to reference one of the cache sets S0-S7 and the tag field is compared to the tag of the data within that set. A tag memory is coupled to the plurality of cache sets for storing the tags of the data blocks. If the tag field of the address matches the tag field of the referenced set, then there is a hit and the data can be obtained from the block that exhibited the hit. If the tag field of the address does not match the tag field of the referenced set, there is a miss.

Data addresses are indexed from the main memory 14 to the column associative cache 16 according to two hashing

functions, h_1 and h_2 , which are applied by controller 15. The hashing functions are operations that map the data addresses from the main memory to the cache sets based on spatial and temporal locality. Spatial locality suggests that future addresses are likely to be near the locations of current addresses. Temporal locality indicates that future addresses are more likely to reference the most recently accessed locations again.

The first hashing function, h_1 , is preferably a bit selection operation. In a bit selection operation, data is indexed to the sets of the column associative cache according to its index field. Since some data may contain the same index field, there is high probability that there will be conflict miss between the data. The column associative cache of the present invention resolves the conflict by then applying a second hashing function, h_2 . The second hashing function dynamically chooses a different location in which the conflicting data can reside. The second hashing function, h_2 , is preferably a bit flipping operation that flips the highest ordered bit of the referenced by the index address and accesses the conflicting data at the set indexed by the inverted address. As shown in FIG. 2A, the first hashing function, h_1 , indexes address a , 17 to set S1. Address 11 then attempts to access S1 but there is a miss because address 17 is already there. To resolve the conflict, the second hashing, h_2 , function is applied to address 11. This hashing function flips the highest ordered bit of the index field so that address 11 can be indexed to S5. Thus, S1 and S5 share locations through h_1 and h_2 so that conflicts are resolved not within a set but within the column of sets of the entire cache.

A comparison of a column associative cache with a conventional two way set associative cache is illustrated in FIG. 2B. In the conventional cache, a set, such as set 2, stores two lines of data. Thus, if the requested data is stored in either line of a set, there is a hit. Drawbacks of such a cache are the high hit access time and hardware complexity. The column associate cache performs as a direct mapped cache unless there is a miss. With a miss it accesses another location within the same memory column. Thus, two sets share two locations.

Also, shown in FIG. 2A is a rehash block 18 coupled to each cache set for indicating whether the set has been rehashed. A rehashed location is a set that has already been indexed through the second hashing function to store data. The purpose of the rehash block is to indicate whether a location stores data through a rehashed index so the data should be replaced in preference for a non-rehashed index. Temporal locality suggests that rehashed locations should be preferentially replaced.

FIG. 3 discloses a controller decision tree for indexing the cache. Table 1 provides the decision tree mnemonics and cycle times for each cycle. First, the first hashing function, h_1 , is applied to the memory address a . If the first-time access is a hit, then the data is accessed to the processor. However, if the first-time access is a miss, then the rehashed location block of that set is checked (Rbit=1?). If the rehash block has been set to one, then the data is removed from that cache set indexed by $h_1[a]$ and data from the main memory is retrieved and substituted therefor (Clobber 1). Next, the rehash block is reset to zero to indicate that the data in this set is to be indexed by the first hashing function h_1 for future indexes.

On the other hand, if the rehash block is set to zero, then upon a first-time miss, the second hashing function h_2 access is attempted. If the second hashing function indexes to valid data, then there is a second time hit. For a second time hit,

the data is retrieved from that cache set and the data in the cache sets indexed by the first and second hashing functions, $h_1[a]$ and $h_2[a]$, are swapped (SWAP) so that the next access will likely result in a first time hit (temporal locality). However, if the second hashing function provides a second time miss, then the data in that set is replaced (Clobber2). Data from the main memory is retrieved and placed in the cache set indexed by the second hashing function, $h_2[a]$. Then the data in the cache sets indexed by the first and second hashing function, h_1 and h_2 are swapped with each other (SWAP).

TABLE 1

Mnemonic	Action	Cycles
$h_1[a]$	bit-selection access	1
$h_2[a]$	bit-flipping access	1
swap	swap data in sets accessed by $h_1[a]$ and $h_2[a]$	2
clobber1	get data from memory, place in set accessed by $h_1[a]$	M
clobber2	get data from memory, place in set accessed by $h_2[a]$	M
Rbit=1?	check if set accessed by $h_1[a]$ is a rehashed location	0
inCAM?	check if a (or its index) matches a CAM entry	0
putinCAM	place a (or its index) in the CAM	1
putinCAM*	place the index of a and the tag present in the cache location accessed with $h_1[a]$ into the CAM	1

At startup, all of the empty cache sets have their rehash blocks set to one so that compulsory misses are handled immediately.

The rehash block 18 increases the hit rate and decreases the access time for the column associative cache. The increase in performance is due to the fact that the data in the non-rehashed location are the most recent accessed data and, according to temporal locality, this data is more likely to be needed again. The removal of older data which will probably not be referenced again whenever a conflict miss occurs reduces the amount of clobbering. In addition, the ability to immediately replace a rehashed location on the first access reduces the number of cycles consumed by rehash accesses.

In addition to limiting rehash accesses and clobbering, the column-associative cache with rehash block corrects a problem associated with indexing a reference pattern $a_i, a_j, a_x, a_j, a_x, \dots$ where the addresses a_i and a_j map into the same cache location with bit selection, h_1 , and a_x is an address which maps into the same location with bit flipping, h_2 . FIG. 4 shows how a single column associative cache and a column associative cache with a rehash block will index the above reference pattern. The figure shows at each location, the data stored in that location after the data request indicated by the input sequence. In the column associative cache, address a_i is shown indexed into set S1 by the first hashing function, h_1 . Address a_j attempts to index S1 by the first hashing function, but there is a miss because address i is there. Then using the second hashing function, h_2 , address a_j is indexed to S5 and with a miss that data is retrieved and stored in S5. The data in S1 and S5 is then swapped. Thus, j is now in S1 and i is now in S5. The next address, a_x , attempts to access S5 but will miss because i is there. Then the second hashing function is applied to a_x and it attempts to access S1, but there is a miss because j is there. Since this is a second time miss, the address a_j is removed from S1 and replaced by a_x . Then a_x and a_i are swapped so that i is in S1 and x is in S5. This pattern continues as long as a_i and a_x alternate. Thus, the data referenced by one of a_j and a_x is

clobbered as the data i is swapped back and forth but never replaced.

This detrimental effect is known as thrashing, but as shown in FIG. 4, it does not occur in a column-associative cache with a rehash block. In the column associative cache with a rehash block, a_i is indexed to S1 by the first hashing function h_1 . Address a_j attempts to index S1 but misses because i is there. Since there is a miss, the rehash block for S1 is checked to see if that set has been already indexed by the second hashing function h_2 . Since S1 has not been indexed by h_2 , its rehash block is 0. Then, the second hashing function indexes a_j to S5 and the rehash block is set to 1. Then the data in S1 and S5 are swapped so that j is now in S1 and i is now in S5. Address a_x attempts to access S5 but misses because i is there. However, because the rehash block of S5 is set to 1, j is removed and replaced by x . Thus S1 contains j and S5 contains x , eliminating the thrashing of j . Of course, this column-associative cache suffers thrashing if three or more conflicting addresses alternate, as in $a_i, a_j, a_x, a_i, a_j, a_x, \dots$, but this case is much less probable than in the case of two alternating addresses. Thus, the rehash block alleviates thrashing, reduces the number of rehash accesses and nearly eliminates clobbering.

To further reduce the access time of the column associative cache, a content addressable memory (CAM) 20 is added thereto. The purpose of the CAM is to reduce the number of unnecessary rehash accesses and swaps in the column associative cache. FIG. 5 shows the CAM 20 coupled to the column associative cache 16. The CAM stores addresses that potentially cause conflict misses, such as addresses that have been swapped with the rehashed location in a second-time hit. If the address in the CAM matches requested data address, then the controller attempts to index the referenced data using another hashing function, such as h_2 , as the first hash.

FIG. 6 shows a decision tree for indexing an address a to the column associative cache with the CAM. Table 1 provides the decision tree mnemonics and cycle times for each cycle. First, the CAM is checked to determine whether the index of a matches the address entry within the CAM (inCAM?). If there is a match, then h_2 is used to index a . If $h_2[a]$ indexes valid data, then there is a hit and the data is retrieved. However, if there is a miss, then the data is clobbered and data from the main memory is retrieved and placed in the cache set accessed by h_2 (Clobber2).

On the other hand, if there is no match in the CAM, then h_1 is applied to a for indexing. If $h_1[a]$ indexes valid data, then there is a hit. However, if there is a miss, the rehash block is checked to determine whether the cache set accessed by $h_1[a]$ is a rehashed location (Rbit=1?). If the cache set is a rehashed location (=1), then h_2 is applied to a . A hit results in a or its index being retrieved and placed in the CAM (putinCAM) as a potential conflict. A miss causes the data in the set indexed by $h_1[a]$ to be clobbered and replaced with data retrieved from the main memory (Clobber 1). If the rehash block is not set to 1, then h_2 is applied to a for indexing. A hit results in an address from the index of $h_2[a]$ being placed into the CAM (putinCAM*). The address is reconstructed from the index of a and the tag at $h_1[a]$. Then data in cache sets accessed by $h_1[a]$ and $h_2[a]$ are swapped with each other. A miss causes the data to be clobbered and replaced with data retrieved from the main memory and placed in the set indexed by $h_2[a]$ (Clobber2). Then data in cache sets accessed by $h_1[a]$ and $h_2[a]$ are swapped with each other (SWAP).

An example of how the CAM provides better performance to the column associative cache is evident for the

following reference pattern: $a_i, a_j, a_i, a_j, \dots$. To access the above reference pattern, the column associative cache 18 wastes many cycles swapping a_i and a_j , repeatedly whereas the CAM 20 stores the address that referenced the data into the rehashed location on a second-time hit. For instance, the third reference, i , results in a second-time hit because the data j is indexed into the rehashed location as expected, but its address (i.e., tag and index) is stored in the CAM. The CAM is then checked in parallel with every first-time access, and if a match is found, the control logic will find the data directly by rehashing instead. The benefit of adding a CAM to the column-associative cache is that a swap is no longer necessary between the conflicting data because the CAM quickly points out those addresses which provide second-time hits. Thus, in the above example, a_i remains in the non-rehashed location and is accessed in one cycle by $h_1[a_i]$. The conflicting data a_j remains in the rehashed location and is accessed by $h_2[a_j]$ after a_j is matched with its entry in the CAM.

An important feature of this design is that the search of the CAM does not impose a one cycle penalty. This feature is accomplished by optimizing the CAM so that a search is completed quickly enough to precede the first-time access in the cycle. This feature can also be implemented by performing the CAM access in a previous pipeline stage. However accomplished, eliminating the penalty of searching the CAM is crucial because a significant reduction in execution time is possible only if most of the data in rehashed locations can be retrieved as quickly as those in non-rehashed locations.

Another benefit in using a CAM is evident in a first-time rehash $h_2[a]$ (due to a being in the CAM) that misses. The decision tree shows that in this case, no swap is needed because data is retrieved from the main memory and left in the set indexed by $h_2[a]$. This is done because that address is in the CAM due to a first-time rehash. Therefore, leaving the data in the rehashed location leads to future first-time rehash hits in only one cycle.

One of the drawbacks of using a CAM with a column associative cache is evident in situations when a set accessed by $h_1[a]$ is found to be a rehashed location. Instead of immediately replacing this data, a rehash access must be performed to ensure that the desired data is not located in the rehashed location. This is impossible for the single column-associative cache with rehash block, however, it is feasible when a CAM is included. For example, suppose an address exists in the CAM which causes a first-time rehash hit at $h_2[a]$. The CAM is a finite resource, so this address may be removed from the CAM after it becomes full. Now, if this address appears again in the reference stream, there is no CAM match, so a normal access is attempted when the data is in the set indexed by $h_2[a]$. Thus, replacing the non-rehashed location immediately would result in data being stored in two separate locations. The extra attempted rehash guards against this wasteful situation, but it adds a one cycle penalty.

Another embodiment of the present invention is to have the CAM coupled to the column associative cache without having a rehash block (see FIG. 7). As in the above embodiment, the CAM 20 improves the efficiency of the column associative cache by storing portions of addresses that are expected to indicate future conflict misses. This reduces the number of unnecessary rehash accesses and swaps in the column associative cache. For example, after first time misses, a rehash access is only attempted when the control logic identifies this miss as a conflict. A conflict is identified by finding a match in the CAM. This conflict may be

resolved by rehashing. Thus, fewer rehashes are attempted which improves the second time hit rate and decreases the extent of data being clobbered.

FIG. 8 discloses a controller decision tree for indexing an address to the column associative cache with CAM. Table 1 provides the decision tree mnemonics and cycle times for each cycle. First, the first hashing function, h_1 , is applied to a memory address a . If the first time access is a hit, then the data is accessed. However, if the first time access is a miss, the CAM is checked to see if address a matches a CAM entry (inCAM?).

If address a does not match a CAM entry, the data in address a is removed (clobber1) and data is retrieved from the main memory and placed in the cache set accessed by the first hashing function $h_1[a]$. Then the data from address a is placed in the CAM (putinCAM).

However, if there is a match in the CAM, then the second hashing function $h_2[a]$ is applied. A hit causes the data to be accessed and then the data in the cache sets accessed by $h_1[a]$ and $h_2[a]$ are swapped (SWAP). A miss causes that the data to be removed from the cache set and replaced by data from main memory (clobber2). Then the data in the cache sets accessed by $h_1[a]$ and $h_2[a]$ are swapped (SWAP).

For a general understanding of how to implement the column associative cache with rehash block, the column associative cache with the rehash block and CAM, and the single column associative cache with CAM, reference is made to FIGS. 9-11 and Tables 2-4. The cache implementation for both FIGS. 9-11 are discussed at the register transfer level without the disclosure of the detailed gate and transistor designs since the actual control logic can be easily synthesized from the state flow tables set forth in Tables 2-4.

Furthermore, in order to provide brief yet descriptive details about the various embodiments, several simplifications and assumptions have been made. For example, a discussion regarding the clocking and timing issues is left out. Instead, it is assumed that the controller 15 receives input signals at the start of a cycle and issues output signals at the end of the cycle. Also, for simplicity, the bus interface and driver circuits have been left out.

FIG. 9 shows a hardware implementation of the column associative cache with rehash block for the present invention. The primary element of the column associative cache memory system is a RAM array 23 having a rehash block 25. The RAM, preferably a tag memory, has a plurality of cache sets to store memory addresses. The processor sends a data address via an n -bit multiplexor 22 to a memory address register (MAR) 24. Connected in between the output of the MAR and one of the inputs of the multiplexor 22 is an inverter 26. The multiplexor 22, the MAR 24, and the inverter 26 interact to index the data address from the processor to the RAM. More specifically, the multiplexor and the inverter apply the first hashing function h_1 and the second hashing function h_2 to the data address.

The RAM 23 communicates with the data bus via a data buffer 28. In between the data buffer and the RAM is a second n -bit multiplexor 30. A swap buffer 32 communicates with both the multiplexor 30 and the data buffer 28 so that current data can be placed in the cache set most likely to be accessed.

The controller 15 provides the necessary control logic to each of the above components so that the algorithm of the decision tree in FIG. 3 is followed. The control signals for FIG. 9 are summarized in Table 2 as well as the actions taken for a given state, input, output, and next state. A discussion of the components and Table 2 is set forth below and can be followed in FIG. 3.

TABLE 2

State	Input	Output	Next state
IDLE	OP	LM,RD	b[a]
b[a]	HIT		IDLE
	!HIT,HB	STALL,MSEL,LM,RD,LS	f1[a]
	!HIT,HB	MEM,STALL	XWAIT
f1[a]	HIT	MSEL,LM,WT	f2[a]
	!HIT	MEM	WAIT1
f2[a]		DSEL,LD	f3[a]
f3[a]		MSEL,LM,WT	IDLE
WAIT1	MACK	MSEL,LM,WT	WAIT2
WAIT2		DSEL,LD	WAIT3
WAIT3		MSEL,LM,WT	IDLE
XWAIT	MACK	LD,WT	IDLE

Upon receiving an opcode signal (OP), the controller loads (LM) the MAR with an memory address a from the address bus. Then the controller issues a read or write signal (RD/WT) to the RAM so that the first hashing function h_1 is applied to address a . If the RAM returns a hit signal (HIT), then the data is automatically loaded (LD) into the data buffer 32 to be retrieved and the controller goes to an IDLE state.

If the $h_1[a]$ access misses (!HIT) and the rehash block has not been rehashed (!HB), then the controller stalls the processor (STALL), copies (LS) the data from the $h_1[a]$ access into the swap buffer, loads the MAR with the second hashing function h_2 (MSEL and LM), issues a read (RD) signal to the RAM and moves to the f1[a] state. If the access misses (!HIT) and the rehash block is set to one (HB), then the data is removed and the controller makes a request to the main memory (MEM), stalls the processor (STALL), and moves to the XWAIT state.

In the f1[a] state, a hit causes the controller to load the MAR with that index (MSEL, LM), issue a write signal (WT) to the RAM and move to the f2[a] state. For a miss (!HIT), the controller makes a request to the main memory (MEM) to retrieve data and moves to the WAIT1 state.

In the f2[a] state, the controller swaps the data in the data buffer and the swap buffer (DSEL, LD) and moves to the f3[a] state.

In the f3[a] state, the controller loads the MAR (MSEL, LM), issues a write (WT) signal to the RAM, and moves to the IDLE state.

In the WAIT1 state, the memory acknowledges completion (MACK), the data is taken from the data bus and loaded in the MAR (MSEL, LM), a write signal is issued to the RAM (WT), and the controller moves to the WAIT2 state.

In the WAIT2 state, the controller swaps the data in the data buffer (DSEL, LD) and moves to the WAIT3 state.

In the WAIT3 state, the controller loads (MSEL, LM) the MAR, issues a write signal (WT) to the RAM and moves to the IDLE state.

In the XWAIT state, the controller receives a signal that the access is complete (MACK), loads the data into the data buffer (LD), issues a write command (WT), and moves to the IDLE state.

The circuitry of the column associative cache with CAM and rehash block is more complex than the cache by itself (see FIG. 10). For example, there is a CAM 20, a first in first out (FIFO) counter 36, a CAM buffer 38, and another n -bit multiplexor 40. The FIFO counter points to the next location in the CAM that is to be replaced and the CAM buffer holds indexes while they are being compared or before they are written into the CAM. Even though this hardware consumes a great deal of area, the critical access path of the column associative cache is not affected. Besides the above addi-

tions, the MAR 24 and the swap buffer 32 are shown to have capability for storing partial addresses such as the index and tag fields, respectively.

The state flow table in Table 3 reveals that the control logic for the column associate cache with the CAM and rehash block is more complex. For example, the variables for each state have changed and are referenced differently than the column associative cache. Furthermore, upon receiving an opcode (OP), the controller searches the CAM to determine if there is a match for the address a. If there is no initial match (! MATCH) in the CAM, the controller loads the MAR (LM), issues a read signal (RD) to the RAM, and moves to the b[a] state. A match (MATCH) in the CAM enables the controller to load the MAR (MSEL, LM), issues a read signal (RD) to the RAM; and moves to the ff[a] state.

A hit (HIT) in the ff[a] state enables the controller to place the index field of the data within the MAR into the CAM buffer (LDCAM) and then move to the IDLE state. On the other hand, a miss (! HIT) enables the controller to stall the processor (STALL), make a request to the main memory (MEM), and then move to the WAIT state.

A hit (HIT) in state b[a] causes the controller to place the index field of the data within the MAR into the CAM buffer 38 (LDCAM) and moves to the IDLE state. A miss (!HIT) with a zero rehash block (! HB) or a one rehash block (HB) causes the controller to stall the processor (STALL), load the MAR (MSEL, LM), issue a read signal (RD) to the RAM, load the swap buffer (LS) with the data from b[a] and move to the fl[a] and fc[a] state, respectively.

TABLE 3

State	Input	Output	Next State
IDLE	OP,!MATCH	LM,RD	b[a]
	OP,MATCH	MSEL,LM,RD	ff[a]
ff[a]	HIT	LDCAM	IDLE
	!HIT	STALL, MEM	WAIT
b[a]	HIT	LDCAM	IDLE
	!HIT,!HB	STALL,MSEL,LM,RD,LS	fl[a]
	!HIT,HB	STALL,MSEL,LM,RD,LS	fc[a]
fl[a]	HIT	MSEL,LM,WT,CSEL, LDCAM,WT,CAM	f2[a]
	!HIT	MEM	WAIT1
f2[a]		DSEL,LD,INC	f3[a]
f3[a]		MSEL,LM,WT,LDCAM	IDLE
fc[a]	HIT	LDCAM,WT,CAM	fc2[a]
	!HIT	MEM	WAIT
fc2[a]		INC,LDCAM	IDLE
WAIT	MACK	LD,WT,LDCAM	IDLE
WAIT1	MACK	MSEL,LM,WT	WAIT2
WAIT2		DSEL,LD	WAIT3
WAIT3		MSEL,LM,WT,LDCAM	IDLE

A hit in the fl[a] causes the controller to load the MAR (MSEL, LM), issue a write signal (WT) to the RAM, place the address from the MAR in the CAM (CSEL, LDCAM, WTCAM), and move to the f2[a] state. A miss (!HIT) causes the controller to make a request to the memory (MEM) and go to the WAIT1 state.

In the f2[a] state, the controller points to the next location in the CAM (INC), swaps the data in the data buffer with the data in the swap buffer (DSEL, LD), and moves to the f3[a] state.

In the f3[a] state, the controller places an index within the MAR and the CAM buffer (MSEL, LM, WT, LDCAM) and moves to the IDLE state.

In the fc[a] state, the data is indexed. A hit (HIT) causes the controller to place the index within the MAR into the CAM buffer (LDCAM), place the current index into the CAM (WTCAM), and move to the fc2[a] state. A miss

(!HIT) causes the controller to make a request to the memory to retrieve data (MEM), and move to the WAIT state.

In the fc2[a] state, the controller issues an INC command to the FIFO counter in order to point to the next location in the CAM, places an index within the MAR into the CAM buffer (LDCAM), and moves to the IDLE state.

In the WAIT state, the controller receives a signal indicating that the access is complete (MACK), loads the MAR with the next access (LD), issues a write signal to the RAM (WT), places an index within the MAR into the CAM buffer (LDCAM) and then moves to the IDLE state.

In the WAIT1 state, the controller receives a signal indicating that the access is complete (MACK), loads the MAR (MSEL, LM), issues a write signal (WT), and moves to the WAIT2 state.

In the WAIT2 state, the controller swaps data between the data buffer 28 and the swap buffer 32, loads the data buffer with the data (DSEL,LD), and moves to the WAIT3 state.

In the WAIT3 state, the controller loads the MAR (MSEL, LM), issues a write signal to the RAM (WT), places the index within the MAR into the CAM buffer (LDCAM), and moves to the IDLE state.

Note that all states whose next state is IDLE assert the LDCAM line. This serves as a reminder that in order for the CAM search and the setting of MATCH to precede the first-time cache access, the search must be either extremely fast or part of a previous pipeline stage. LDCAM is listed as an output of the stages executed before the IDLE state as a reminder of these potential solutions. In these cases, actually, the CAM buffer would need to find the next address on the address bus, because the MAR has not yet latched the next reference. Also, note that the state flow Table 3 proceeds similarly to the state flow Table 2 for first-time hits and first-time misses when the rehash block is zero. The only exception is for a second-time hit, when the original non-rehashed address must be placed in the CAM in addition to the swap. This is accomplished by asserting CSEL, LDCAM and WTCAM during state fl[a]. Also, INC is asserted during f2[a] to increment the FIFO counter, which points to the location of the next write to the CAM but does not affect the next CAM search.

The new entries in the state table involve the paths if an initial CAM match occurs or if a first-time miss reveals a rehashed location. If the MATCH line is asserted initially, then the controller moves to set ff[a] and attempts a standard rehash access. If successful, nothing remains to be done. If it misses, then this rehashed location is simply replaced by data from the memory during the WAIT state. Note that MSEL and LM are not to be used to change the MAR contents. Since the address that accesses this location is still in the CAM, a future reference will be successful in one cycle. In the case that a first-time miss reveals a rehashed location, state fc1[a] is entered and, unlike the column-associative cache with rehash block, a rehash is performed to assure that the data does not exist in the rehashed location. If this access does indeed hit, the address is simply placed in the CAM. Thus, a feature reference immediately finds a match in the CAM and completes a rehash access in one cycle. If there is a miss, then the algorithm proceeds as in the column-associative cache with rehash block and replaces the non-rehashed location.

The circuitry of the column associative cache with a CAM is shown in FIG. 11. The control signals for FIG. 11 are summarized in state flow Table 4. A discussion of the components and Table 4 are set forth below and correspond to the decision tree of FIG. 8.

TABLE 4

state	input	output	next state
IDLE	OP	LM,RD,LDCAM	b[a]
b[a]	HIT		IDLE
	!HIT,MATCH	STALL,MSEL,LM,RD, LS	f1[a]
f1[a]	!HIT!MATCH	MSEL,STALL,WTCAM	XWAIT
	HIT	MSEL,LM,WT,DSEL,LD	f2[a]
	!HIT	MEM	WAIT1
f2[a]		MSEL,LM,WT	IDLE
WAIT1	MACK	MSEL,LM,WT,DSEL,LD	WAIT2
WAIT2		MSEL,LM,WT	IDLE
XWAIT	MACK	INC,LD,WT	IDLE

Upon receiving an opcode (OP), the controller loads the MAR (LM), issues a read signal (RD) to the RAM, places the index within the MAR into the CAM buffer (LDCAM) and moves to the b[a] state.

A hit in the b[a] state (HIT) causes the data to be accessed and then the controller moves to the IDLE state. A miss (!HIT) with a match (MATCH) in the CAM causes the controller to stall the processor (STALL), load the MAR (MSEL,LM), issue a read signal (RD) to the RAM, load the swap buffer (LS) with the data from h₁[a] and move to the f1[a] state. A miss (!HIT) without a match (!MATCH) in the CAM causes the controller to make a request to memory (MEM), stall the processor (STALL), write into the CAM (WTCAM) and move to the XWAIT state.

A hit (HIT) in the f1[a] state causes the controller to load the MAR (MSEL,LM), write the RAM (WT), load the data buffer with the data (DSEL,LD) and move to the f2[a] state. A miss (!HIT) causes the controller to make a request to memory (MEM) and move to the WAIT1 state.

In the f2[a] state, the controller loads the MAR (MSEL,LM) and issues a write signal (WT), and moves to the IDLE state.

In the WAIT1 state, the controller receives an input signal indicating that the access is complete (MACK), then loads the MAR (MSEL, LM), issues a write signal (WT), swaps data between the data buffer and the swap buffer, loads the data buffer with the data (DSEL, LD), and moves to the WAIT2 state.

In the WAIT2 state, the controller loads the MAR (MSEL,LM), issues a write signal to the RAM (WT), and moves to the IDLE state.

In the XWAIT state the controller receives an input signal indicating that the access is complete (MACK), then the controller issues an INC command to the FIFO counter in order to point to the next location the CAM, places an index into the MAR (LD), writes the RAM (WT), and moves to the IDLE state.

An important parameter for the CAM disclosed in FIGS. 10 and 11 is its size parameter. Like the victim cache, the percentage of conflicts removed increases as its size increases, because there are more locations to store conflicting data removed from the cache. However, this improvement eventually saturates to a constant level, because there exists only so many conflicting data bits which need to reside therein at one time. However, the CAM can perform without saturation for up to 128 entries, whereas the victim cache can perform only up to 16 entries before saturation occurs.

The column associative cache with a CAM can use the full index field or omit some of the low order bits from the index fields that are to be placed in the CAM. For example, if two bits are trapped from the index, then four different addresses could cause a CAM match with the same entry.

These addresses may be consecutive numbers, since the low order bits have been dropped. The use of partial index fields increase the number of rehashes attempted, because a reference is predicted to be a conflict if it indexes one of four consecutive locations. As seen previously, an increase in the number of rehashes attempted often decreases the second time hit rate and likely degrades performance. However, this modification may prove useful in applications where data or instructions are often known to be stored sequentially or in consecutive bits.

Also, note that the present invention is not limited to the two hashing functions, h₁ and h₂, bit selection operation and bit flipping operation. Other hashing functions may be used in addition to bit flipping in order to improve the randomness of accesses and to decrease the amount of clobbering.

While the invention has been particularly described in conjunction with a preferred embodiment thereof, it will be understood that many alternatives, modifications and variations will be apparent to those skilled in the art without departing from the spirit and scope of the invention as defined by the appended claims.

We claim:

1. A cache memory system comprising:

a cache memory having a plurality of cache locations, each for storing a cache line of data, separately accessed from a main memory, and having a first tag memory, each cache location being indexed by indexes, taken from memory addresses, through first and second hashing functions such that plural memory addresses having a common index access plural memory locations through the first and second hashing functions and different indexes access common memory locations through the first and second hashing functions;

hash control storage storing control data comprising hash data associated with each cache location which indicates the hashing function used to store data in the cache location; and

a controller coupled to the cache memory responsive to memory addresses in accesses to the main memory for accessing data in the cache memory through the first and second hashing functions and for replacing data in the cache memory from the main memory responsive to the control data and to comparisons between tags of the memory addresses and tags stored in the first tag memory.

2. A cache memory system as claimed in claim 1 wherein the controller checks the hash data of the cache location indexed by the first hashing function when there is a miss at that cache location and applies the second hashing function only when said hash data indicates data stored in the cache location was not stored using the second hashing function.

3. A cache memory system as claimed in claim 1 wherein the controller responds to the hash data to determine whether to replace data stored in a first location indexed through the first cache hashing function or a second cache location indexed through the second hashing function.

4. A cache memory system as claimed in claim 3 wherein the controller swaps data replaced in a cache location with data in another cache location indexed by a common index.

5. A cache memory system as claimed in claim 1 further comprising a second tag memory coupled to the controller for storing as control data at least portions of memory addresses that indicate that data stored in a cache location is likely indexed through one of the hashing functions.

6. A cache memory system as claimed in claim 5 wherein the controller accesses cache memory locations through the first hashing function or the second hashing function depen-

dent on whether at least a portion of a memory address is stored in the second tag memory and, where a miss results at a cache memory location with access through the first hashing function and the second hashing function, the controller replaces the data stored through the first hashing function if said hash data indicates the data accessed through the first hashing function had been stored using the second hashing function, or through the second hashing function if said hash data indicates the data accessed through the first hashing function had been stored using the first hashing function.

7. A cache memory system as claimed in claim 1 wherein the hash control storage comprises a second tag memory coupled to the controller for storing as control data at least portions of memory addresses that indicate a likely hashing function through which data stored in cache is indexed.

8. A cache memory system as claimed in claim 7 wherein the second tag memory is a content addressable memory.

9. A cache memory system comprising:

a cache memory having a plurality of cache locations, each for storing a cache line of data, separately accessed from a main memory, and having a first tag memory, each cache location being indexed by indexes, taken from memory addresses, through first and second hashing functions such that plural memory addresses having a common index access plural memory locations through the first and second hashing functions and such that different indexes access common memory locations through the first and second hashing functions;

hash data associated with each of the plurality of cache locations for indicating the hashing function used to store data therein; and

a controller coupled to the cache memory for accessing data in the cache locations through the first and second hashing functions and for replacing data in the cache locations from main memory, the controller being responsive to the hash data and a comparison of tags of the memory address and stored tags in cache memory in determining whether to replace data in a first location accessed through the first hashing function or in a second location accessed through the second hashing function.

10. A cache memory system according to claim 9, wherein the first hashing function is a bit selection operation.

11. A cache memory system according to claim 9, wherein the controller checks the hash data of a cache location indexed by the first hashing function when there is a miss to determine whether to apply the second hashing function.

12. A cache memory system according to claim 9, wherein the second hashing function is a bit selection and flipping operation.

13. A cache memory system according to claim 9, wherein the controller removes the data from the cache location indexed by the second hashing function after a miss and retrieves new data from the main memory in place therefor.

14. A cache memory system according to claim 13, wherein the controller swaps the new data in the cache location indexed by the second hashing function with the data in the cache location indexed by the first hashing function.

15. A cache memory system according to claim 9, wherein the controller responds to a miss at a cache location through the first hashing function, and to hash data indicating data is stored at that cache location through the second hashing function, to remove data from that cache location and retrieve data from main memory in place therefor.

16. A cache memory system as claimed in claim 15 wherein the controller swaps data replaced in a cache location with data in another cache location indexed by a common index.

17. A cache memory system according to claim 9, further comprising a second tag memory coupled to the controller for storing at least portions of addresses that indicate that data stored in a cache location is likely to be indexed through the second hashing function, the controller using the second hashing function in the initial cache indexing where an address is found in the second tag memory.

18. A cache memory system comprising:

a cache data memory having a plurality of cache locations for storing plural cache lines of data, each cache location being referenced by a memory address having an index field and a tag field, and each cache location being indexed by indexes, taken from memory addresses, through first and second hashing functions such that plural memory addresses having a common index access plural memory locations through the first and second hashing functions and such that different indexes access common memory locations through the first and second hashing functions;

a first tag memory coupled to the cache data memory for storing the tag fields of the data stored in the plurality of cache locations;

hash data coupled to the cache data memory for indicating hashing functions used to index data in the cache locations;

a second tag memory coupled to the cache data memory for storing at least portions of memory addresses that indicate that data stored in a cache location is likely indexed through one of the hashing functions; and

a controller responsive to the hash data, the first tag memory and the second tag memory for indexing memory addresses according to at least one of the plural hashing functions.

19. A cache memory system according to claim 18, wherein the controller applies first and second hashing functions to a memory address, the second hashing function being a bit selection and bit flipping operation.

20. A method for accessing data from a cache data memory, having a plurality of cache locations and a first tag memory, comprising the steps of:

indexing a memory address having an index field and a tag field into an indexed cache location according to a hashing function;

comparing a tag field of the memory address to a tag field in the first tag memory for the indexed cache location; and

generating a hit when the tag field of the memory address matches the tag field of the indexed cache location, and generating a miss when the tag field of the memory address does not match the tag field of the indexed cache location, and in generating a miss, choosing between the step of indexing another cache location through another hashing function and the step of replacing data, the step of replacing data in the cache location being chosen if hash data indicates data located in the cache location was indexed through another hashing function.

21. A method according to claim 20, further comprising the steps of connecting a content addressable memory to the cache data memory for storing portions of memory addresses, each portion indicating that data stored in a cache location is likely indexed through one of plural hashing

17

functions, and checking the content addressable memory for a match with a portion of the memory address.

22. A method as claimed in claim 20 further comprising swapping the replaced data in a cache location with data in another cache location indexed by a common index. 5

23. A method of accessing data from a cache data memory having a plurality of cache locations and first tag memory comprising the steps of:

indexing a memory address having an index field and a tag field into an indexed cache location according to a hashing function applied to the index field; and 10

comparing a tag field of the memory address to a tag field in the first tag memory for the indexed cache location; and

storing control data which identifies the hashing function used to store data in each cache location; 15

18

wherein data is accessed in the cache locations through first and second hashing functions and data is replaced in the cache locations from main memory responsive to the control data which is stored according to past cache operations and comparisons between tags of memory addresses and tags stored in the first tag memory.

24. A method as claimed in claim 23 further comprising determining from a second tag memory a hashing function through which data stored in a cache location is likely indexed and selects that hashing function for indexing the cache location.

25. A method as claimed in claim 23 further comprising swapping data in the cache location indexed by the second hashing function with the data in the cache location indexed by the first hashing function when replacing data.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,530,958
DATED : June 25, 1996
INVENTOR(S) : Anant Agarwal and Steven D. Pudar

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

At column 1, line 4, insert the following paragraph:

---GOVERNMENT SUPPORT

This invention was made with government support under Grant Number 9012773-MIP awarded by the National Science Foundation. The government has certain rights in the invention.---

Signed and Sealed this
Eighth Day of October, 1996

Attest:



BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

- [54] **CACHE MEMORY USING A LOWEST PRIORITY REPLACEMENT CIRCUIT**
 [75] **Inventor:** Shia-Jeh Chang, Naperville, Ill.
 [73] **Assignee:** AT&T Bell Laboratories, Murray Hill, N.J.
 [21] **Appl. No.:** 307,857
 [22] **Filed:** Oct. 2, 1981
 [51] **Int. Cl.³** G06F 13/00
 [52] **U.S. Cl.** 364/200
 [58] **Field of Search** 364/200 MS FILE, 900 MS FILE

[57] **ABSTRACT**

A data processing system having a processor, main memory, and a cache memory system which implements the least recently used replacement algorithm in replacing cache memory words with main memory words. The cache memory system is comprised of a cache control circuit and a plurality of cache memories. Each cache memory stores cache memory words having a similar time usage history. The first cache memory stores cache memory words which are more recently used than the cache memory words in the second cache memory, and the second cache memory stores cache memory words which are more recently used than the cache memory words in the third cache memory. When a main memory word must be transferred to the cache memory, the main memory word is stored in the first memory; and the first cache memory's least recently used cache memory word is stored in the second cache memory. The least recently used cache memory word from the second cache memory is stored in the third cache memory. These operations maintain the proper time usage history of the cache memories.

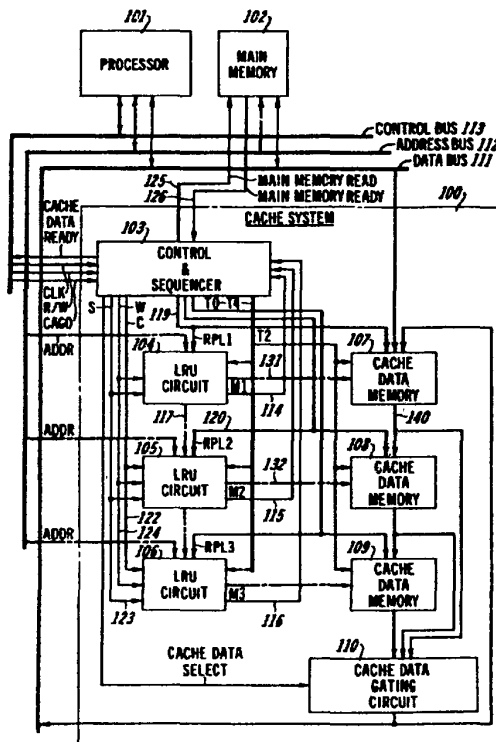
[56] **References Cited**

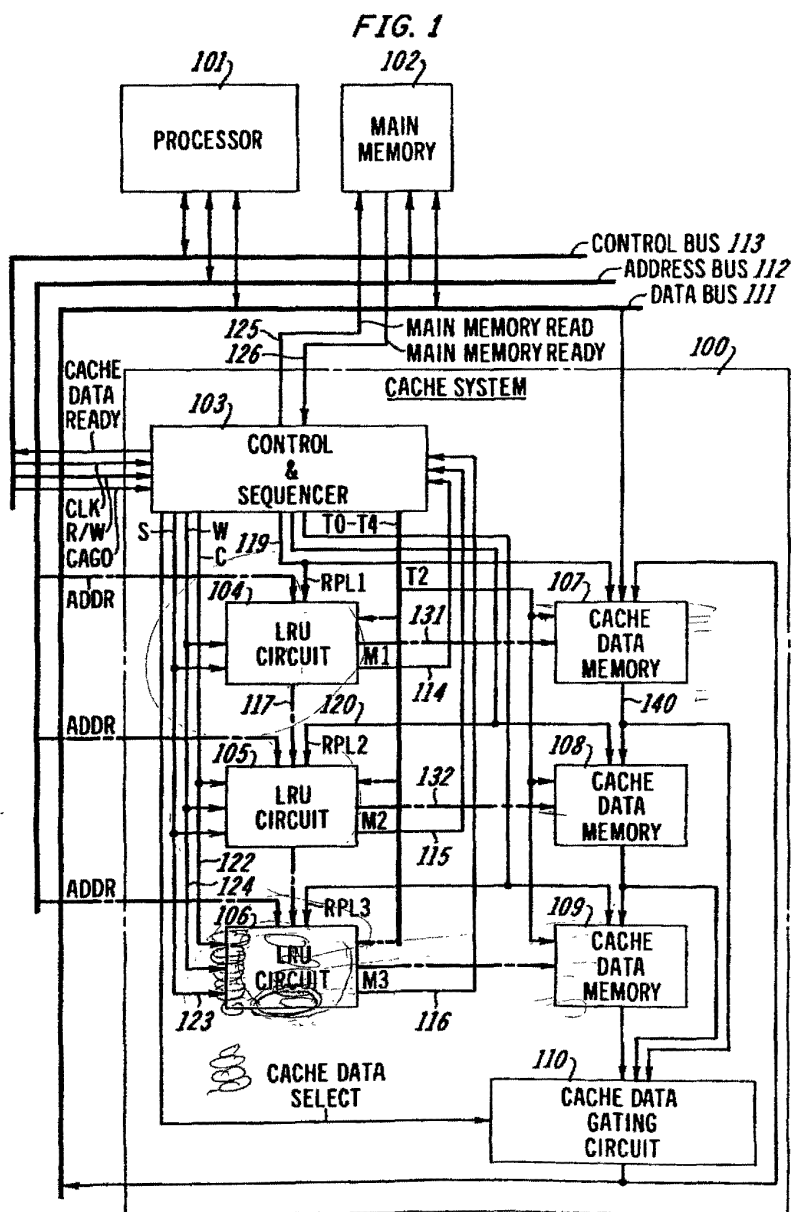
U.S. PATENT DOCUMENTS

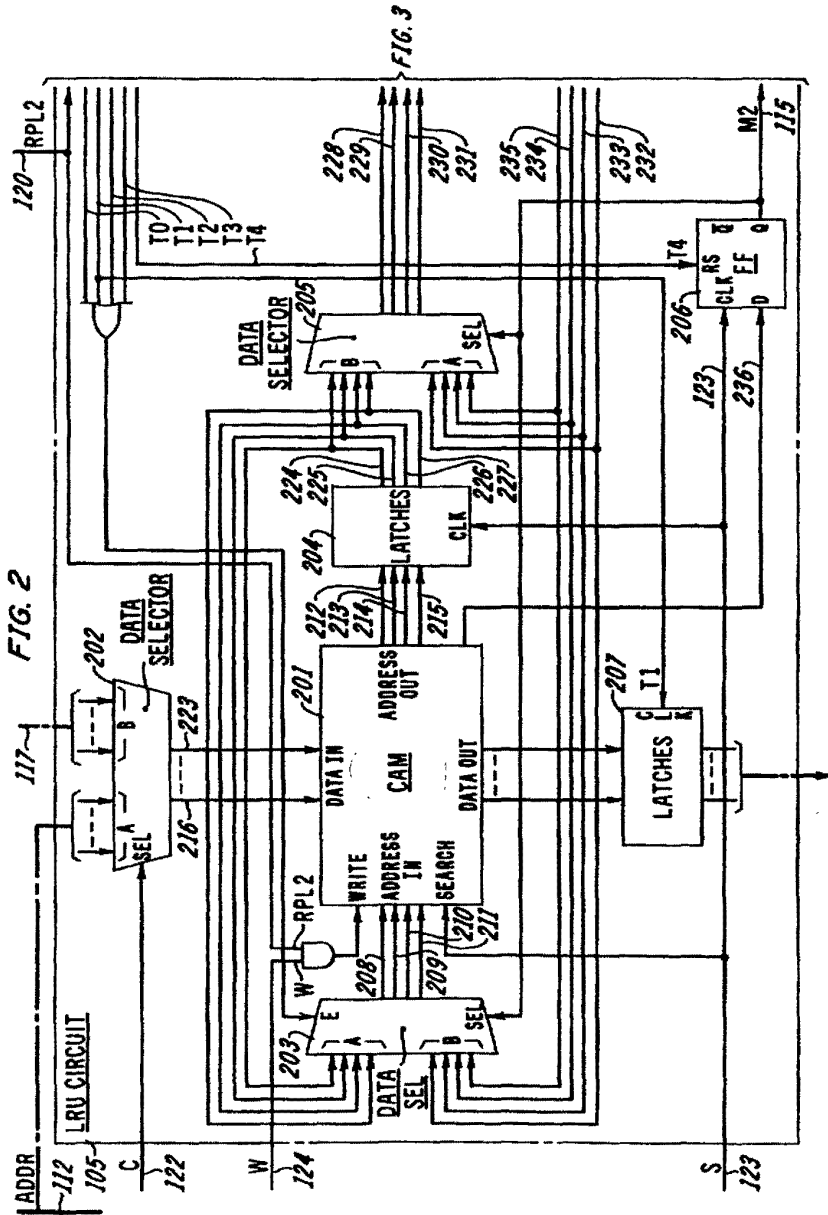
3,588,829	6/1971	Boland et al.	340/172.5
3,840,862	10/1974	Ready	364/200
3,949,368	4/1976	West	340/172.5
4,084,230	4/1978	Matick	364/200
4,128,882	12/1978	Dennis	364/200
4,322,795	3/1982	Lange et al.	364/200

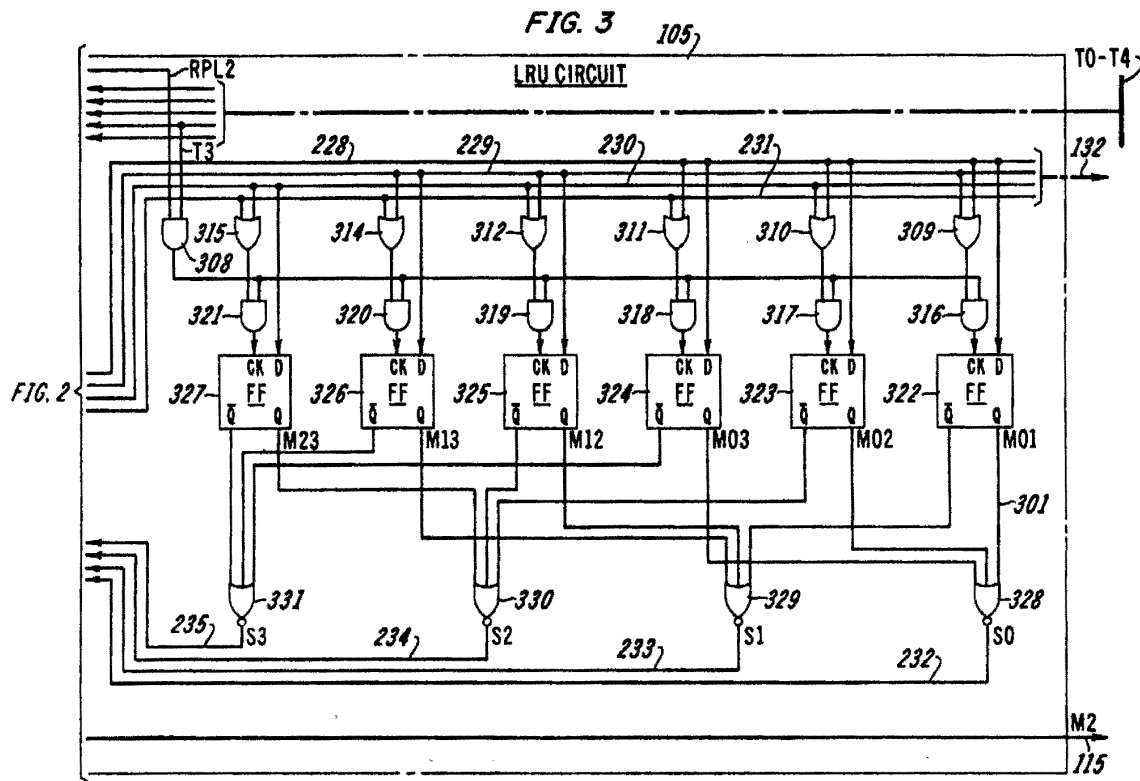
Primary Examiner—Eddie P. Chan
Assistant Examiner—O. Schatoff
Attorney, Agent, or Firm—P. Visserman

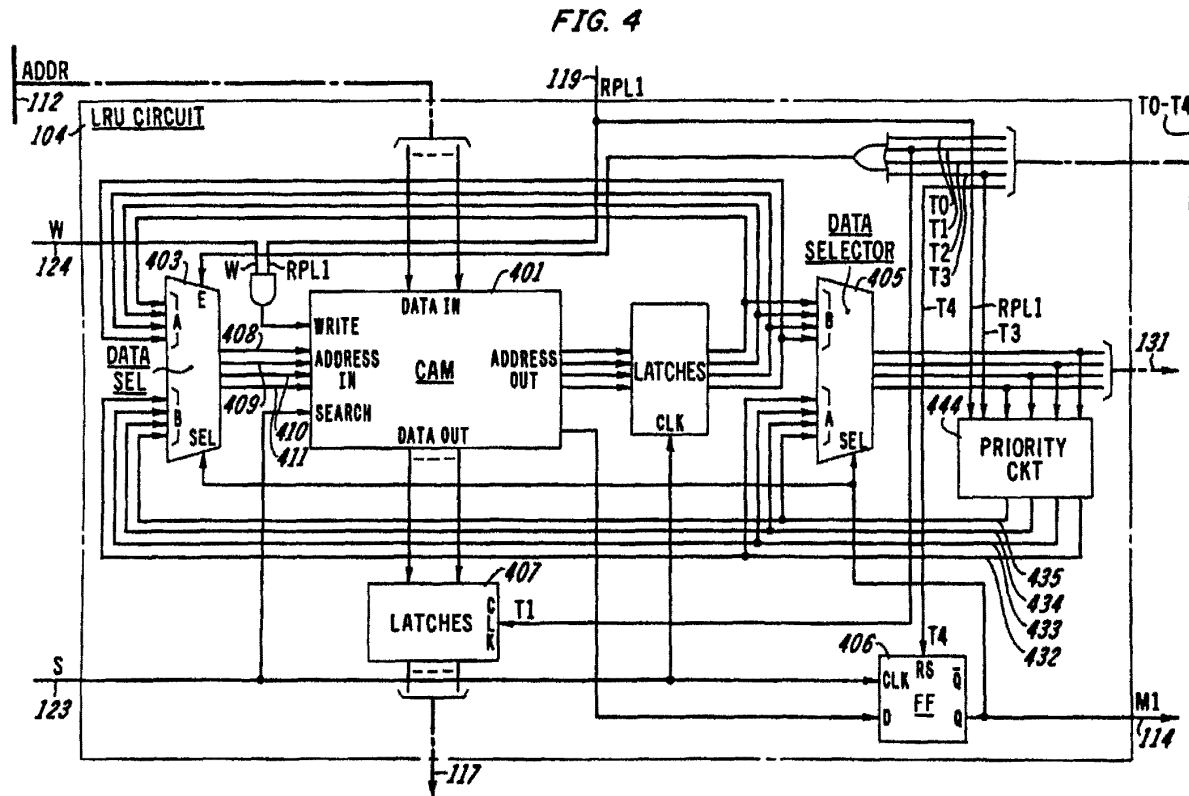
17 Claims, 5 Drawing Figures











U.S. Patent Jul. 3, 1984
 Sheet 4 of 5
 4,458,310

FIG. 5

	M01	M02	M03	M12	M13	M23	SELECTED WORD	LEAST RECENTLY USED WORD
501	1	1	1	0	0	0		1
502	1	0	1	0	0	1	2	1
503	1	0	0	0	0	0	3	1
504	0	0	0	1	1	0	1	0

CACHE MEMORY USING A LOWEST PRIORITY REPLACEMENT CIRCUIT

TECHNICAL FIELD

My invention relates to computer systems, and, particularly, to a system using a cache memory in which the cache storage location for storing new information is the location of the lowest priority word in the cache memory.

BACKGROUND OF THE INVENTION

Modern computer systems employ processors which are capable of operating at much higher rates of execution than large capacity main memories can support, and a low capacity, high-speed cache memory is commonly used in addition to a large capacity main memory to improve program execution speed. The cache memory stores a limited number of instruction or data words; and for each memory read operation, the cache memory is checked to determine if the information is available in the cache memory. If the information is there, it will be read from the cache memory; otherwise, it will be read from the main memory. If the information must be read from the main memory, the new information must replace existing information in the cache memory at some cache storage location. A satisfactory cache storage location for storing new information is identified by one of the several commonly used replacement algorithms, e.g., random replacement, least recently used, etc. In general, the least recently used replacement algorithm is considered to be the most efficient algorithm; however, implementation of this algorithm in a cost-effective manner without incurring large time delays in maintaining a priority of cache memory locations, with respect to which is the least recently used memory location, has proven difficult to achieve. In particular, it has proven difficult to design a cache memory which was capable of expansion in the field.

SUMMARY OF THE INVENTION

Advantageously, in a computer system in accordance with the present invention, the cache memory system is divided into sections with each section containing cache data words which have a similar priority. Each section has a priority circuit associated with it which maintains the relative priority of the cache data words. Furthermore, the time required to update the cache memory upon receipt of a main memory word which must be inserted into the cache memory is reduced, since the main memory data word is written into one section simultaneous with the transfer of lowest priority cache data words from sections having higher priority cache data words to sections having lower priority cache data words.

In one embodiment of the invention, the data processing system consists of a processor, which requests data words by generating main memory address signals, a main memory and a cache memory system. The cache memory system is comprised of a cache control circuit and a first and a second cache memory. The advantage of configuring the cache memory system into more than one cache memory is that the system is modular and can be expanded in the field. Also, each cache memory can be implemented as one large scale integrated circuit. Each cache memory stores cache data words which are duplicates of words stored in the main memory. Each cache memory also stores the main memory addresses

where the associated cache data words are duplicated in main memory. When the processor requests a data word by transmitting main memory address signals, the first and second cache memory compare the stored memory addresses with these memory signals to determine if the requested memory word is stored within either the first or second cache memory. If a cache memory finds a match, it transmits to the cache control circuit a match signal; otherwise, the cache memory transmits a mismatch. If the cache control circuit receives mismatch signals from both cache memories, it generates and transmits the necessary signals to cause two operations to take place. During the first operation, the main memory responds to the main memory address signals to access and transmit the desired main memory word to the processor and to the first cache memory. Also, during this first operation, the first cache memory accesses its lowest priority cache data word with the associated stored main memory address and transmits these to the second cache memory. During the second operation, the first cache memory stores the accessed main memory word and main memory address signals in the previously accessed first cache memory locations and the second cache memory stores the lowest priority cache data word and stored main memory address from the first cache memory in second cache memory locations.

Further, the cache control means is responsive to a mismatch signal from the first cache memory and a match signal from the second cache memory to cause two operations to be performed within the cache memories. During the first operation, the first cache memory accesses and transmits the lowest priority cache data word and the associated main memory address to the second cache memory and the second cache memory transmits the cache data word associated with the matched stored memory address to the first cache memory and to the processor. During the second operation, the first cache memory stores the cache data word and address from the second cache memory in the memory location formerly used by the lowest priority cache data word and memory address. Also, during the second operation, the second cache memory will store the transmitted cache data word and associated address from the first cache memory.

Additionally, each cache memory will be comprised of a match and a data memory. The match memory will be used to store the stored main memory addresses and the data memory will be used to store the cache data words. The match memory will perform a comparison for each set of main memory address signals which the processor sends out and this memory will indicate a match or a mismatch. When a match is found, the match memory transmits an address to the data memory so that it can access and transmit the designated cache data word. A content addressable memory can be used to implement the match memory.

Further, each cache memory has a priority circuit which maintains the priority of each cache data word with respect to when it was accessed within the first cache memory. The priority maintained by the priority circuit is the time usage history of the cache data words. The lowest priority cache data word is the least recently used cache data word.

In a data processing system comprising a processor, main memory and cache memory system having two sections, one illustrative method accesses and updates

the cache memory system by storing the cache data words into the cache memory system with the first section containing words which have a higher priority than the words stored in the second section. When the processor accesses a data word, each section is checked to detect whether or not the desired word is contained in that section. If the desired word is not contained in any section, then the main memory will be accessed and the desired word transmitted to the processor and the first section. The accessed main memory word will be used to replace the lowest priority cache data word of the first section and this word will be designated as the highest priority cache data word and the word which had the second lowest priority will be designated as the lowest priority cache data word. The former lowest priority cache data word will be transmitted to the second section where it will replace the lowest priority word of the second section and will become the highest priority word of that section. The word which had the second lowest priority in the second section will then be designated as the lowest priority word.

If the requested word is detected as being in the second section, then the word from the second section will be transmitted to the processor and will be stored in the first section as the highest priority word of the first section. The lowest priority word of the first section will be transferred to the second section where it will become the highest priority word of the second section. The lowest priority word can be the least recently used word, and the highest priority word can be the most recently used word.

BRIEF DESCRIPTION OF THE DRAWING

The invention may be better understood from the following detailed description when read with reference to the drawing in which:

FIG. 1 is a block diagram representation of a data processing system embodying the present invention;

FIGS. 2 and 3 show in greater detail LRU circuit 105 of FIG. 1;

FIG. 4 shows in greater detail the content addressable memory of LRU circuit 104 of FIG. 1; and

FIG. 5 shows a table giving an example of the operation of the priority circuit of FIG. 3.

DETAILED DESCRIPTION

In a data processing system as illustrated in FIG. 1, data and instruction words are stored in memory locations of main memory 102 and cache system 100. Processor 101 reads these memory locations by transmitting an address via address bus 112 and control signals via control bus 113. The cache system 100 is comprised of control sequencer 103, LRU circuits 104, 105 and 106, cache data memories 107, 108 and 109, and cache data gating circuit 110. The LRU circuits and cache data memories are grouped into pairs, and each pair represents a cache memory unit. For example, LRU circuit 104 and cache data memory 107 comprise one cache memory unit.

The cache data words stored in the cache data memories are organized into groups with each group containing cache data words which were last read by processor 101 at a similar point in time. Each group is stored in one of the cache data memories. For example, the most recently-used group of words is stored in cache data memory 107, and the least recently used group of words is stored in cache data memory 109. As processor 101 performs read operations, cache data words may have

to be transferred between cache data memories to maintain the time usage history of the memories. For example, if it is necessary to read a word from main memory 102, this main memory word will replace the least recently used cache data word of cache data memory 104; and the replaced cache data word will be transferred to cache data memory 108.

During a read operation, the address transmitted by processor 101 is checked by LRU circuits 104, 105, and 106 to determine if the addressed word is contained within cache data memories 107, 108, or 109, respectively.

For example, if LRU circuit 104 determines that the addressed word is contained within cache data memory 107, it transmits the address of this word to cache data memory 107 via cable 131. Cache data memory 107 responds to this address by accessing and transmitting the desired word to cache data gating circuit 110. From cache data gating circuit 110, the desired data word is transmitted to processor 101 via data bus 111. If LRU circuit 104 does not match the address being transmitted by processor 101 via address bus 112, it transmits to control sequencer 103 a "1" signal via conductor 114 which indicates a mismatch. The other LRU circuits function in a similar manner.

In addition to checking if the associated cache data memory has the desired memory word, the LRU circuits maintain the priority of each word in the associated cache data memory. This priority information is automatically updated by the LRU circuit for each access to the associated cache data memory and defines which word in the cache memory is the least recently used word.

The system's operation is further illustrated by the three following examples. In the first example, it is assumed that the desired word is not present in the cache system 100 and must be read from main memory 102. If the desired word is not in the cache system 100, then all the LRU circuits will be transmitting "1" signals via the match lines 114, 115 and 116. In response to these signals, control sequencer 103 will access main memory 102 to obtain the desired word. Since the word read from main memory 102 is the most recently used word, it must be placed in cache data memory 107, the least recently used word from cache data memory 107 must be written into cache data memory 108, and the least recently used word of cache data memory 108 must be written into cache data memory 109. The least recently used word of cache data memory 109 no longer exists in cache memory 100 at the completion of the previous operations.

In the second example of the operation of cache system 100, it is assumed that the desired word is in cache data memory 107. Since the desired word is in cache data memory 107, it is not necessary to access a word in main memory 102 or to transfer a memory word from cache data memory 107 to cache data memory 108. Rather, LRU circuit 104 will simply update the priority information stored internally to circuit 104 to properly reflect the usage order of memory words in data memory 107.

In the third example, the desired memory word is assumed to be in data memory 108. In this case, LRU circuit 105 would match the address being transmitted by processor 101 via address bus 112 and cause data memory 108 to access and transmit the desired word to data gating circuit 110. Control sequencer 103 would then cause this desired data word to be transmitted by

clm 4, 5, 6, 7

data gating circuit 110 via data bus 111 to processor 101. Since this desired word is the most recently used word, it must be written into data memory 107. The least recently used word of data memory 107 must be written into the memory location which had previously held the desired memory word in data memory 108.

LRU circuit 105 is illustrated in FIGS. 2 and 3, and LRU circuit 106 is similar in design. LRU circuit 104 is illustrated in FIG. 4. FIG. 2 shows the circuit which is used to check the address transmitted by processor 101 via address bus 112 to determine whether the desired word is in cache data memory 108, and FIG. 3 gives the details of the priority circuit which is used to keep track of the least recently used word in cache data memory 108. When processor 101 reads a word, it first transmits the CAGO signal and the clock signal via control bus 113 to the control sequencer 103 and processor 101 transmits the address via address bus 112. Control sequencer 103 responds to these signals and generates the C signal and S signal which are transmitted via conductors 122 and 123 to the LRU circuits. Data selector 202 responds to the C signal on conductor 122 by selecting the address bits being transmitted via address bus 112 and transmits these address bits via conductors 216 through 223 to the data-in inputs of content addressable memory (CAM) 201. The CAM contains four words, each word having eight bits. The CAM responds to the S input transmitted via conductor 123, and the address bits being received on the data-in inputs to compare these address bits with the contents of each of the four words stored internally. If one of the four words matches the address bits, then a "1" will be transmitted via the associated conductor 212, 213, 214 or 215. If no match is found, then a "1" is transmitted via conductor 236 and stored in flip-flop 206 at T1 time. If a match is found, the state of the conductors 212 through 215 will be stored in latches 204 by the falling edge of the S signal which is transmitted via conductor 123. Data selector 205 will select the contents of latches 204 which are being transmitted via conductors 224 through 227 to be transmitted via conductors 228 through 231 over cable 132 to cache data memory 108. Cache data memory 108 will respond to the address being transmitted via cable 132 by accessing the desired word and transmitting this word to data gating circuit 110, as previously described. Assuming that the desired word was stored in data memory 108, this word now is the most recently used word and must be transferred to data memory 107 and the least recently used word of data memory 107 must be transferred to data memory 108 and the address of this word written into CAM 201.

FIG. 4 shows the circuit which is used to check the address transmitted by processor 10 via address bus 112 to determine whether the desired word is in cache data memory 107, and FIG. 3 gives the details of the priority circuit which is used to keep track of the least recently used word in cache data memory 108. The circuit of FIG. 4 is identical in operation to FIG. 2 with the exception that FIG. 4 does not have a data selector similar to data selector 202 of FIG. 2, and includes priority circuit 444. Priority circuit 444 is identical in design to the priority circuit described with reference to FIG. 3. The reason why no data selector is needed is that the circuit of FIG. 4 always uses the address being transmitted via address bus 112. The circuit of FIG. 4 does not need a data selector because this circuit is associated with the most recently used words in cache memory 100, hence, does not have to decide whether to use the

address from address bus 112 or from an LRU circuit having higher priority, as does the circuit shown in FIG. 2. This distinction will be illustrated more clearly in the following example.

To illustrate the operations of the circuits shown in FIG. 2 and FIG. 4, the previously described example 3 is used. Example 3 described the operations which must take place when the desired word is in data memory 108. A more detailed description of this example will now be given by first describing it from the point of view of LRU circuit 105, and then describing the corresponding actions in LRU circuit 104. It is presumed that the word 1 in data memory 108 and word 3 in data memory 107 are the least recently used words. To perform these different operations, the controller sequencer 103 generates a variety of timing signals, the most important of which are T0 through T4. During T0, the address bits on address bus 112 are selected through data selector 202 and used to search CAM 201 for a match. Assuming that these address bits match the contents of word 2 in CAM 201, a "1" will be transmitted on conductor 213; conductors 212, 214, and 215 will be conducting "0s". This operation is done under control of the S signal transmitted via conductor 123 and the C signal transmitted via conductor 122 to data selector 202. The information on conductors 212 through 215 is stored in latches 204 at the end of the S signal. In addition, the S signal also clocks the match output terminal of CAM 201 into flip-flop 206. The output of flip-flop 206 is the M2 signal which is transmitted to control sequencer 103 via conductor 115.

During T1, data selector 203 responds to the M2 signal by selecting the output of latches 204 as an address which is transmitted to CAM 201 via conductors 208 through 211, and data selector 205 responds to the M2 signal by selecting the output of latches 204 as an address which is transmitted to data memory 108 via cable 132. In response to the address on conductors 208 through 211, CAM 201 reads the contents of the second word and transmits these contents to latches 207 in which these contents are stored at the end of T1. Data memory 108 reads the contents of its second word in response to the address transmitted via cable 132. These contents are stored internal to data memory 108 and transmitted to data gating circuit 110. During T1, LRU circuit 104 accesses the address of the least recently used word and transmits this via cable 117 to LRU circuit 105, and data memory 107 accesses the least recently used word and transmits this via cable 140 to data memory 108, as will be described later. The address from LRU circuit 104 must be written into CAM 201 and the corresponding data word written into data memory 108. During T2, data selector 203 will again select the output of latches 204 which contain the address for word 2 to be used as an address for CAM 201. The least recently used address word from LRU circuit 104 will be stored in word 2. During T2, control sequencer 103 will transmit the W signal via conductor 124 and the RPL2 signal via conductor 120 which causes CAM 201 to write the information present at the data input terminals into word 2. At the same time, the least recently used word of data memory 107 is written into word 2 of data memory 108 with the address being supplied by the output of latches 204 via data selector 205 and cable 132. As will be described later, the priority circuit shown in FIG. 3 must be updated during T3 to reflect the fact that word 2 is now the most recently

used word in LRU circuit 105. During T4, flip-flop 206 is reset.

Example 3 is now described with respect to LRU circuit 104 with reference to FIG. 4. During T0, a search is performed of CAM 401; however, since no match is found, the match output terminal is a "0" which is stored in flip-flop 406, and no M1 signal is transmitted to control sequencer 103.

During T1, since there is no M1 signal, CAM 401 is addressed by the address from the priority circuit 444 with an address which is transmitted to the ADDRESS IN terminals of CAM 401 via conductors 432 through 435, data selector 403 and conductors 408 through 411. This address bit is the address of the least recently used word of CAM 401 and data memory 107. Also, during T1, data memory 107 is addressed by the outputs of the priority circuit 444 via data selector 405 and cable 131. At the end of T1, the output data of CAM 401 is clocked into latches 407. The contents of latches 407 are transmitted via cable 117 to LRU circuit 105.

During T2 control sequencer 103 transmits the PRL1 and W signals to LRU circuit 104 and data memory 107 via conductors 119 and 124, respectively. In response to these signals, the contents of address bus 112 are written into the location of the least recently used word as determined by the bits on conductors 432 through 435 in CAM 401. At the same time, the word present on data bus 111 is written into data memory 107 at the address transmitted via cable 131.

During T3, the priority circuit 444 must be updated. Note, that during this example, it was not necessary to change any information connected with LRU circuit 106 or data memory 109.

Another previous example to be considered is example 1 where the desired word is not contained within data memories 107 through 109 and must be read from main memory 102. For this example, none of the LRU circuits will find a match during time T0, and at the end of time T0, control sequencer 103 will access main memory 102 to obtain the desired word. Control sequencer 103 accesses main memory 102 by transmitting the main memory read signal via conductor 125. When main memory 102 has accessed the desired word, it responds by transmitting the main memory ready signal via conductor 126 and placing the desired memory word on data bus 111. Control sequencer 103 is responsive to the main memory ready signal to generate the cache data ready signal which informs processor 101 that the data is available on data bus 111 and to execute the following steps to update the LRU circuits and the data memories.

After receipt of the main memory ready signal, the control sequencer 103 transmits the T1 signal. The results of the transmission of the T1 signal are first described with reference to FIG. 2, since no match was found, the M2 signal is not being transmitted via conductor 115, data selector 203 selects the address of the least recently used word which is transmitted via conductors 232 through 235 from the priority circuit of FIG. 3 to perform a read on CAM 201. The word read out of CAM 201 is the address of the least recently used data word which is stored in data memory 108. At the same time, a read is performed on data memory 108 based on the address being transmitted via cable 132, which, again, is the address of the least recently used word. At the end of T1, the address of the least recently used word is clocked into latches 207 and the data being accessed from data memory 108 is similarly clocked

into a similar set of latches in data memory 108. The same type of operation is being performed in LRU circuits 104 and 106 and data memory 107 and data memory 109.

During T2, the addresses being transmitted via cable 117 from LRU circuit 104 is written into CAM 201 at the address of the least recently used word as defined by the address transmitted via conductors 232 through 235 from the priority circuit of FIG. 3. Similarly, the data which had been accessed from data memory 107 is written into data memory 108.

With respect to LRU circuit 104, the address on address bus 112 is written into the location in CAM 401 which is addressed by information transmitted via conductors 432 through 435 from priority circuit 444 which designates the least recently used word address. The data which is present on data bus 111 is written into the least recently used word of data memory 107 at the address of the least recently used word. Similar operations take place in LRU circuit 106 and data memory 109. During T3, the priority circuits of LRU circuits 104, 105, and 106 must be updated to reflect the fact that the previously least recently used words are now the most recently used words.

To illustrate the operation of the priority circuit shown in FIG. 3, reference is made to example 3 which described the operations when the desired word is contained in data memory 108. The operation of the priority circuit of FIG. 3 is similar in operation to priority circuit 444 of FIG. 4 and the priority circuit of LRU circuit 106. In the previous example, the least recently used word was word 1 in data memory 108 and the corresponding address in CAM location 1 of LRU circuit 105. During the match operation which took place during time T0, word 2 of CAM 201 was found to contain the address which processor 101 was attempting to read. During time T3, the priority circuit shown in FIG. 5 must be updated to reflect the fact that word 2 is now the most recently used word. However, word 1 still remains the least recently used word. Flip-flops 322 through 327 are used to maintain the priority of the words contained in CAM 201 and data memory 108 with respect to the usage order. NOR gates 328 through 331 decode the information contained in flip-flops 322 through 327 so as to indicate which word is the least recently used word. For example, if NOR gate 328 is transmitting a "1" via conductor 232, this indicates that word 0 is the least recently used word. OR gates 309 through 315 and AND gates 316 through 321 are used to determine which flip-flops 322 through 327 should be modified during an update operation on the priority circuit. Table I defines the significance of one of these flip-flops being set. For example, if flip-flop 322 is set, then flip-flop 322 will transmit the M01 signal as a "1" to NOR gate 328 via conductor 301. The significance of the flip-flop 322 being set is that word 0 has been used more recently than word 1.

TABLE I

Flip-flop Set	Signal Transmitted by Flip-flop	Defines		
		Word used more recently	than	Word
322	M01	0		1
323	M02	0		2
324	M03	0		3
325	M12	1		2
326	M13	1		3

TABLE I-continued

Flip-flop Set	Signal Transmitted by Flip-flop	Defines	
		Word used more recently	than Word
327	M23	2	3

The functions performed by NOR gates 328 through 331 are defined by Table 2.

TABLE 2

S0 = M01 · M02 · M03
S1 = M01 · M12 · M13
S2 = M02 · M12 · M23
S3 = M03 · M13 · M23

By convention, if a "1" is transmitted via conductor 232, this is defined to mean that the S0 signal is being transmitted. If flip-flop 322 is set, then the value in Table 2 for M01 is a "1", and the value for M01 is a "0"; and if flip-flop 322 is reset, then the value for M01 is a "0" and the value for M01 is a "1". For example, if flip-flops 322, 323 and 324 are reset, then the S0 signal is transmitted via conductor 232.

The operations of OR gates 309 through 315 and AND gates 316 through 321 at update time is defined by Table 3.

TABLE 3

"1" transmitted via conductor at update time	Flip-flops which are set	Flip-flops which are reset
228	322, 323, 324	
229	325, 326	322
230	327	323, 325
231		324, 326, 327

Update time occurs at time T3 when the RPL2 signal is being transmitted via conductor 120 from control sequencer 103. T3 and RPL2 and ANDed together by AND gate 306 which enables the OR gates 309 through 315 and AND gates 316 through 321. For example, if a "1" is being transmitted via conductor 231 during the update time, then flip-flops 324, 326 and 327 will be reset. A "1" being transmitted via conductor 231 indicates that word 3 is now the most recently used word, hence, by Table 1, flip-flops 324, 326 and 327 cannot be set because they indicate that word 0, word 1 and word 2, respectively, have been more recently accessed than word 3.

To more clearly illustrate the operations of the circuit shown on FIG. 3, the previous example of word 2 being matched during the operation at time T0 will now be described with respect to FIG. 5. Line 501 shows the initial state of the flip-flops 322 through 327. When word 2 is determined to contain the desired word, the contents of word 2 are accessed in both CAM 201 and data memory 108 and transmitted and stored within LRU circuit 104 and data memory 107. The least recently used words from LRU circuit 104 and data memory 107 are transmitted to LRU circuit 105 and data memory 106 and are stored in word 2 of each of these memories. After this information has been stored in word 2, then word 2 is the most recently used word and flip-flops 322 through 327 must be updated accordingly. Since word 2 was the selected word, data selector 205 of FIG. 2 is transmitting a "1" via conductor 230. OR gates 309 through 315 and AND gates 316 through 321 respond to the "1" being transmitted via conductor 230

to set flip-flops 327 and reset flip-flops 323 and 325. This is shown on line 502 of FIG. 5. Note, that the least recently used word is still word 1 in line 502. If, in the next search operation, the desired word is word 3, the flip-flops 322 through 327 will be updated during time T3 to reflect the states shown in line 503. If, on the next search operation, word 1 is found to contain the desired information, then the flip-flops 322 through 327 will be updated to reflect the state shown in line 504. Note, that the least recently used word is now word 0 which has not been accessed in the last three operations during which words 2, 3 and 1 were both accessed.

It is to be understood that the above-described embodiment is merely illustrative of the principles of the invention and that other arrangements may be devised by those skilled in the art without departing from the spirit and scope of the invention.

What is claimed is:

1. A data processing system comprising:
 - a processor means for generating main memory address signals;
 - a main memory having a plurality of memory locations for storing main memory words;
 - a cache control means;
 - first and second cache memories each having a plurality of memory locations for storing main memory addresses and corresponding cache data words in a priority order, and each responsive to main memory address signals which mismatch all of the main memory addresses stored therein to generate and transmit a mismatch signal to said cache control means;
 - said cache control means responsive to concurrent generation of said mismatch signals by said first and second cache memories to generate and transmit a first control signal to said main memory and said first and second cache memories;
 - said main memory responsive to said first control signal and said mismatched main memory address signals to access and transmit a main memory word to said first cache memory;
 - said first cache memory responsive to said first control signal to transmit the lowest priority cache data word and its corresponding stored main memory address to said second cache memory, and to store said transmitted main memory word and said main memory address signals; and
 - said second cache memory responsive to said first control signal to store the transmitted lowest priority cache data word and its corresponding main memory address.
2. A data processing system in accordance with claim 1 wherein said second cache memory is further responsive to main memory address signals which match a main memory address stored therein to generate and transmit a match signal to said cache control means;
 - said cache control means is further responsive to a mismatch signal from said first cache memory and said match signal from said second cache memory to generate and transmit a second control signal to said first and second cache memories;
 - said first cache memory responsive to said second control signal to transmit the lowest priority cache data word and its corresponding stored main memory address to said second cache memory; and
 - said second cache memory responsive to said second control signal to store said lowest priority cache

data word and said corresponding stored main memory address transmitted in response to said second control signal from said first cache memory in the cache memory locations associated with the stored main memory address which matched said main memory address signals.

3. A data processing system in accordance with claim 2 wherein said second cache memory is further responsive to said second control signal to transmit said matched main memory address and its corresponding cache data word to said first cache memory; and said first cache memory further comprises means responsive to said second control signal to store said matched stored main memory address and said corresponding cache data word in the cache memory locations of said transmitted corresponding main memory address and said transmitted lowest priority cache data word of said first cache memory, respectively.

4. A data processing system in accordance with claim 1 wherein said first cache memory is further responsive to said first control signal to store said main memory word and said mismatched main memory address signals in the cache memory locations of said transmitted lowest priority cache data word and said transmitted corresponding stored main memory address in said first cache memory.

5. A data processing system in accordance with claim 1 wherein said second cache memory is further responsive to said first control signal to store said transmitted lowest priority cache data word and said transmitted corresponding stored main memory address from said first cache memory in the cache memory locations of the lowest priority cache data word and corresponding stored main memory address of said second cache memory, respectively.

6. A data processing system in accordance with claim 2 wherein said second cache memory further comprises a match memory having a plurality of memory locations for storing said stored main addresses and a data memory having a plurality of memory locations for storing said cache data words;

said match memory is responsive to said matched main memory address signals to transmit said match signal and to generate and transmit a cache memory address of the memory location whose contents matched said matched main memory address signals to said data memory, and responsive to said mismatched main memory address signals to generate and transmit said mismatch signal; and said data memory is responsive to said cache memory address to access and transmit said corresponding cache data word.

7. A data processing system in accordance with claim 6 wherein said match memory is comprised of a content addressable memory.

8. A data processing system in accordance with claim 6 wherein each of said first and second cache memories further comprises a priority means for determining the least recently used cache data word which is the lowest priority cache data word.

9. A data processing system in accordance with claim 8 wherein each of said priority means is further adapted for generating the address of the least recently used data word.

10. A data processing system in accordance with claim 9 wherein said priority means of said first cache

memory further comprises a storage means and a logic means; and

said logic means responsive to contents of said storage means and said cache memory address to generate and store information defining the accessed order of said cache data words of said first cache memory in said storage means.

11. In a data processing system having a processor for generating main memory address signals, a main memory for storing main memory words, first and second cache memories for storing main memory addresses and corresponding cache data words and for matching a stored main memory address word with the main memory address signals, and a cache control for controlling said first and second cache memories, a method of accessing said cache memories and said main memory;

comprising the steps of:

storing a set of said cache data words and corresponding main memory address words having a higher priority than another set of said cache data words and corresponding main memory address words in said first cache memory;

storing said other set of said cache data words and corresponding main memory address words in said second cache memory;

detecting main memory address signals which mismatch all of main memory address words stored in said first and second cache memories;

reading from said main memory, the main memory word addressed by the mismatched main memory address signals;

transferring said main memory word to said processor and said first cache memory;

storing said main memory word and said mismatched main memory address signals in said first cache memory;

transmitting the lowest priority cache data word of said first cache memory to said second cache memory;

replacing said lowest priority cache data word of said first cache memory with said main memory data word;

identifying within said first cache memory said main memory data word as the highest priority cache data word and another cache data word as the lowest priority cache data word; and

storing said transmitted cache data word from said first cache memory in said second cache memory.

12. The invention of claim 11 wherein said transmitting step comprises the steps of:

replacing the lowest priority cache data word of said second cache memory with said transmitted cache data word; and

identifying within said second cache memory said transmitted cache data word as the highest priority and another cache data word as the lowest priority cache data word.

13. In a data processing system having a processor for generating main memory address signals, a main memory for storing main memory words, first and second cache memories for storing main memory addresses and corresponding cache data words and for matching the stored main memory addresses with the main memory address signals, and a cache control for controlling said first and second cache memories, a method of accessing said cache memories and said main memory;

comprising the steps of:

13

storing a set of said cache data words and corresponding main memory addresses having a higher priority than another set of said cache data words and corresponding main memory addresses in said first cache memory;
 storing said other set of said cache data words and corresponding main memory words in said second cache memory;
 detecting main memory address signals which match one of the stored main memory addresses in said second cache memory;
 transferring the cache data word corresponding to the matched one of said stored main memory addresses from said second cache memory to said processor and said first cache memory; and
 storing said transferred cache data word from said second cache memory in said first cache memory.

14. The invention of claim 13 wherein said storing of said transferred cache data word step comprises the steps of:

transmitting the lowest priority cache data word of said first cache memory to said second cache memory;
 replacing said lowest priority cache data word of said first cache memory with said transferred cache data word from said second cache memory; and
 identifying within said first cache memory said transferred cache data word from said second cache memory as the highest priority cache data word

14

and another cache data word as the lowest priority cache data word.

15. The invention of claim 14 wherein said transmitting step comprises the steps of:

replacing the lowest priority cache data word of said second cache memory with said transmitted cache data word from said first cache memory; and
 identifying within said second cache memory said transmitted cache data word from said first cache memory as the highest priority cache data word and another cache data word as the lowest priority cache data word.

16. The invention of claims 11 or 14 wherein said lowest priority cache data word of said first cache memory comprises a least recently used cache data word of said first cache memory and said transmitting step comprises the step of transmitting said least recently used cache data word of said first cache memory; and

said replacing step comprises the step of replacing said least recently used cache data word of said first cache memory.

17. The invention of claim 14 wherein said highest priority cache data word from said cache memory comprises a most recently used cache data word and said step of transferring comprises the step of transferring said most recently used cache data word; and

said step of replacing comprises the step of replacing with said most recently used cache data word.

* * * * *

35

40

45

50

55

60

65

IW 7696177



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 16, 2018

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE
RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS
OF:

APPLICATION NUMBER: *09/608,266*

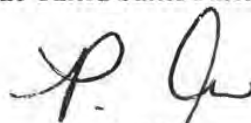
FILING DATE: *June 30, 2000*

PATENT NUMBER: *6,771,646*

ISSUE DATE: *August 03, 2004*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office




P. SWAIN
Certifying Officer

PART *2* OF *2* PART(S)



US006003123A

United States Patent [19]
Carter et al.

[11] **Patent Number:** **6,003,123**
[45] **Date of Patent:** **Dec. 14, 1999**

- [54] **MEMORY SYSTEM WITH GLOBAL ADDRESS TRANSLATION**
- [75] **Inventors:** Nicholas P. Carter, Somerville; Stephen W. Keckler, Cambridge; William J. Dally, Framingham, all of Mass.
- [73] **Assignee:** Massachusetts Institute of Technology, Cambridge, Mass.
- [21] **Appl. No.:** 09/021,658
- [22] **Filed:** Feb. 10, 1998

Related U.S. Application Data

- [62] Division of application No. 08/314,013, Sep. 28, 1994, Pat. No. 5,845,331.
- [51] **Int. Cl.⁶** G06F 12/10
- [52] **U.S. Cl.** 711/207; 711/207
- [58] **Field of Search** 711/147, 202, 711/203, 206, 207, 209

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,241,396	12/1980	Mitchell et al.	364/200
4,408,274	10/1983	Wheatley et al.	364/200
5,075,842	12/1991	Lai	395/479
5,251,308	10/1993	Frank et al.	395/425
5,404,478	4/1995	Arai et al.	395/416
5,465,337	11/1995	Kong	395/417

OTHER PUBLICATIONS

Carter, Nicholas P., et al., "Hardware Support For Fast Capability-based Addressing," Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI), Oct. 5-7, 1994, pp. 1-9.

Tyner, Paul, "APX 432 General Data Processor Architecture Reference Manual, Chapter 3, Objects for Program Environments," Intel Corporation, Jan. 1981, pp. 3-1 to 3-37.

Fabry, R.S., "Capability-Based Addressing," Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, October 15-17, 1973, pp. 413-412.

Dally, William J. et al., "An Object Oriented Architecture," IEEE, 0149-7111/85/0000/0154, 1985, pp. 154-161.

Goodman, James R. et al., "The Wisconsin Multicube: A New Large Scale Cache-Coherent Multiprocessor," IEEE, CH2545-2/88/0000/0422, 1988, pp. 422-431.

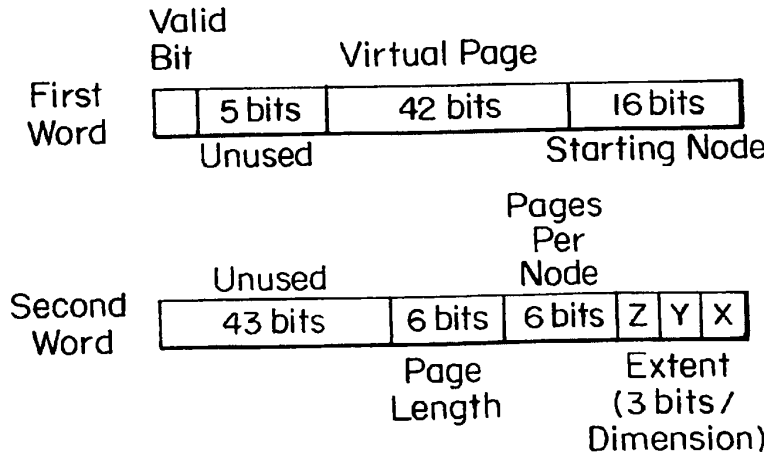
Dally, William J. et al., "M-Machine Architecture v1.0 MIT Concurrent VLSI Architecture Memo 58," Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Aug. 24, 1994, pp. 1-50.

Primary Examiner—Eddie P. Chan
Assistant Examiner—Kevin Verbrugge
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds, P.C.

[57] **ABSTRACT**

A multiprocessor system having shared memory uses guarded pointers to identify protected segments of memory and permitted access to a location specified by the guarded pointer. Modification of pointers is restricted by the hardware system to limit access to memory segments and to limit operations which can be performed within the memory segments. Global address translation is based on grouping of pages which may be stored across multiple nodes. The page groups are identified in the global translation of each node and, with the virtual address, identify a node in which data is stored. Pages are subdivided into blocks and block status flags are stored for each page. The block status flags indicate whether a memory location may be read or written into at a particular node and indicate to a home node whether a remote node has written new data into a location.

12 Claims, 17 Drawing Sheets



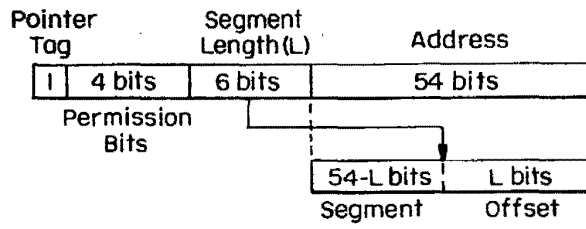


FIG. 1A

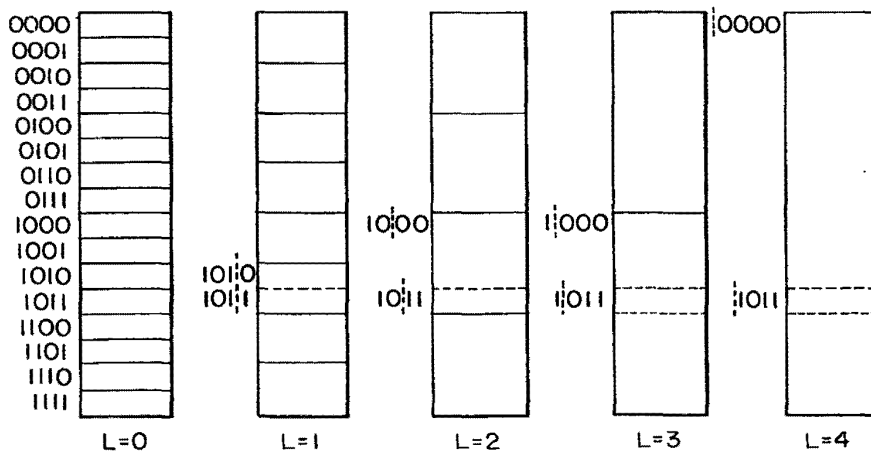


FIG. 1B

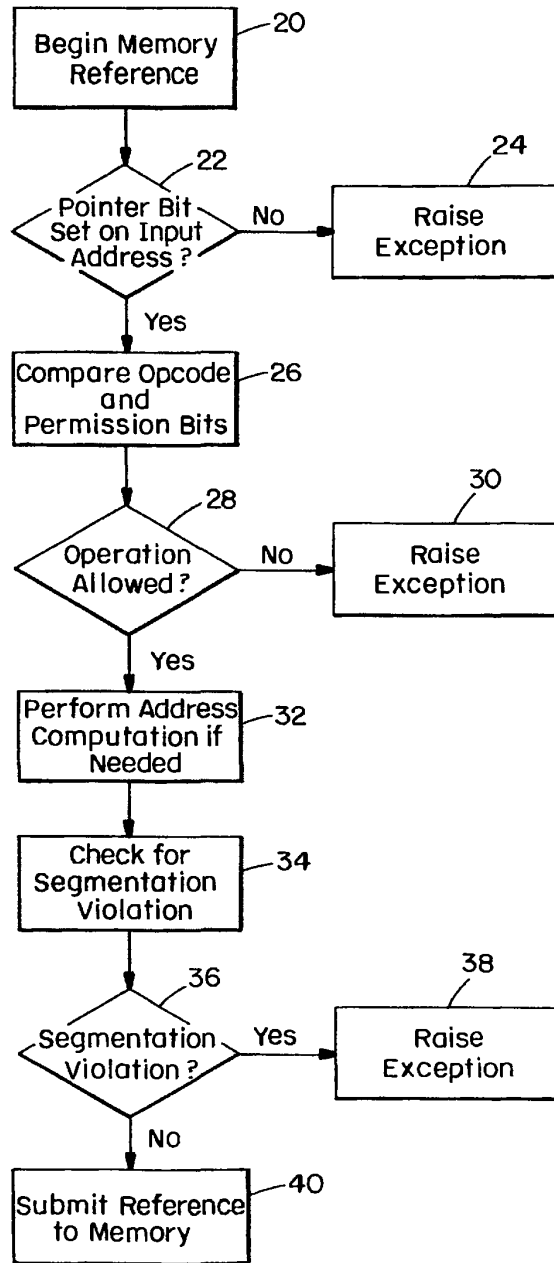


FIG. 2A

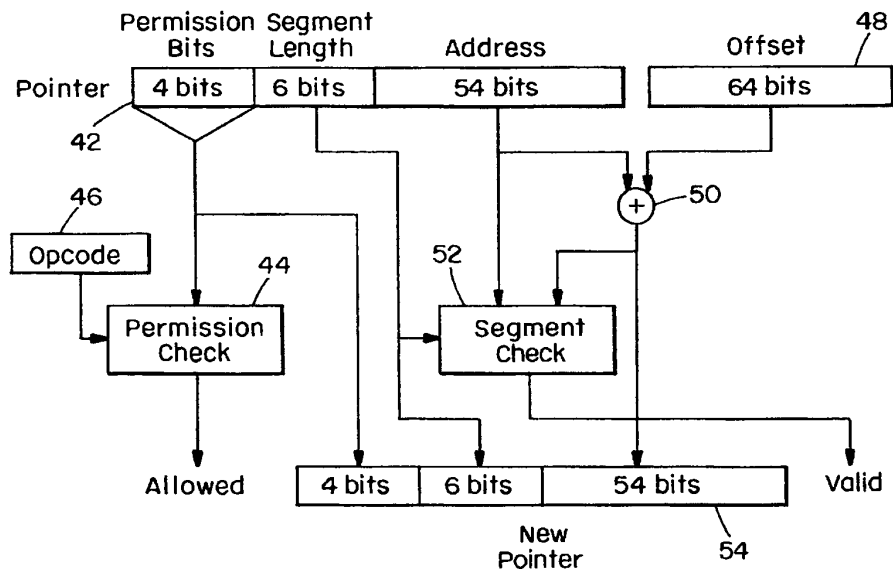


FIG. 2B

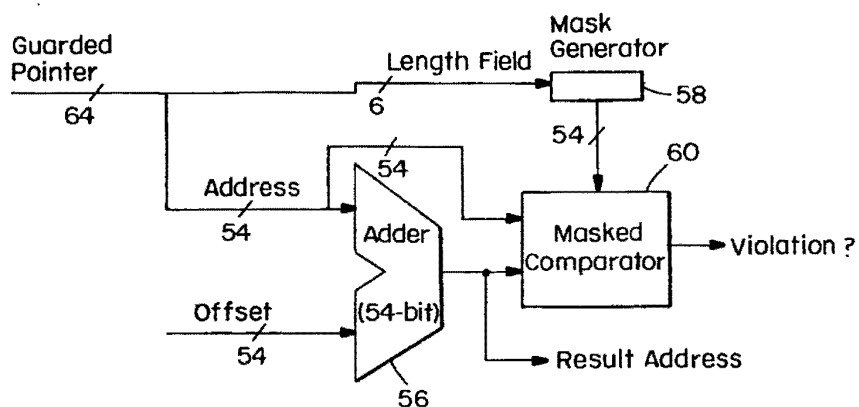


FIG. 3

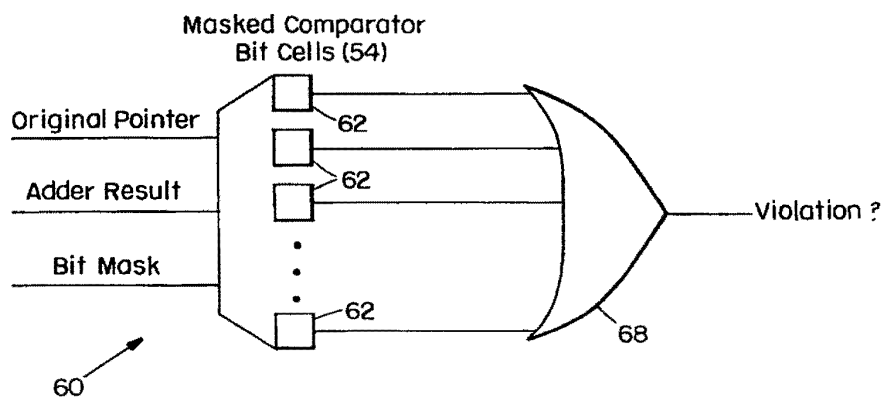


FIG. 4

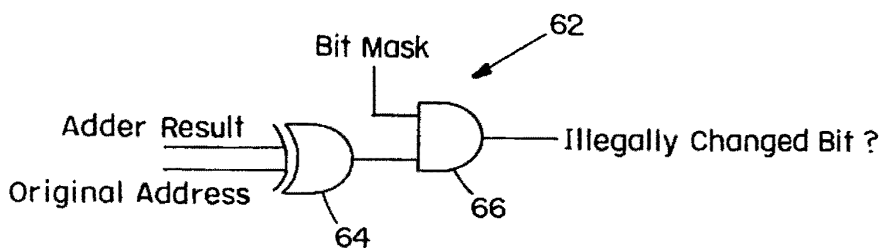


FIG. 5

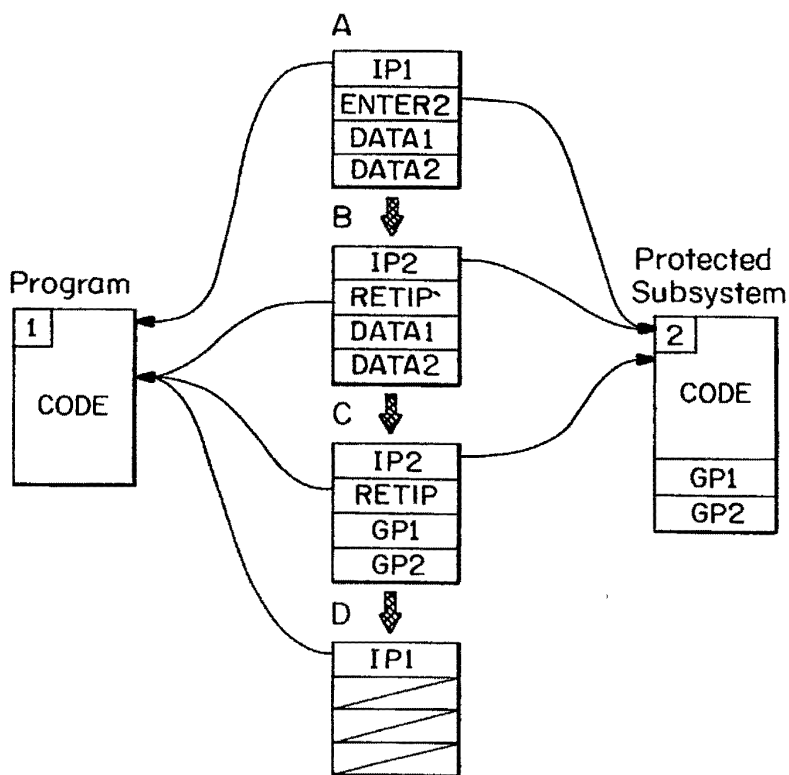


FIG. 6

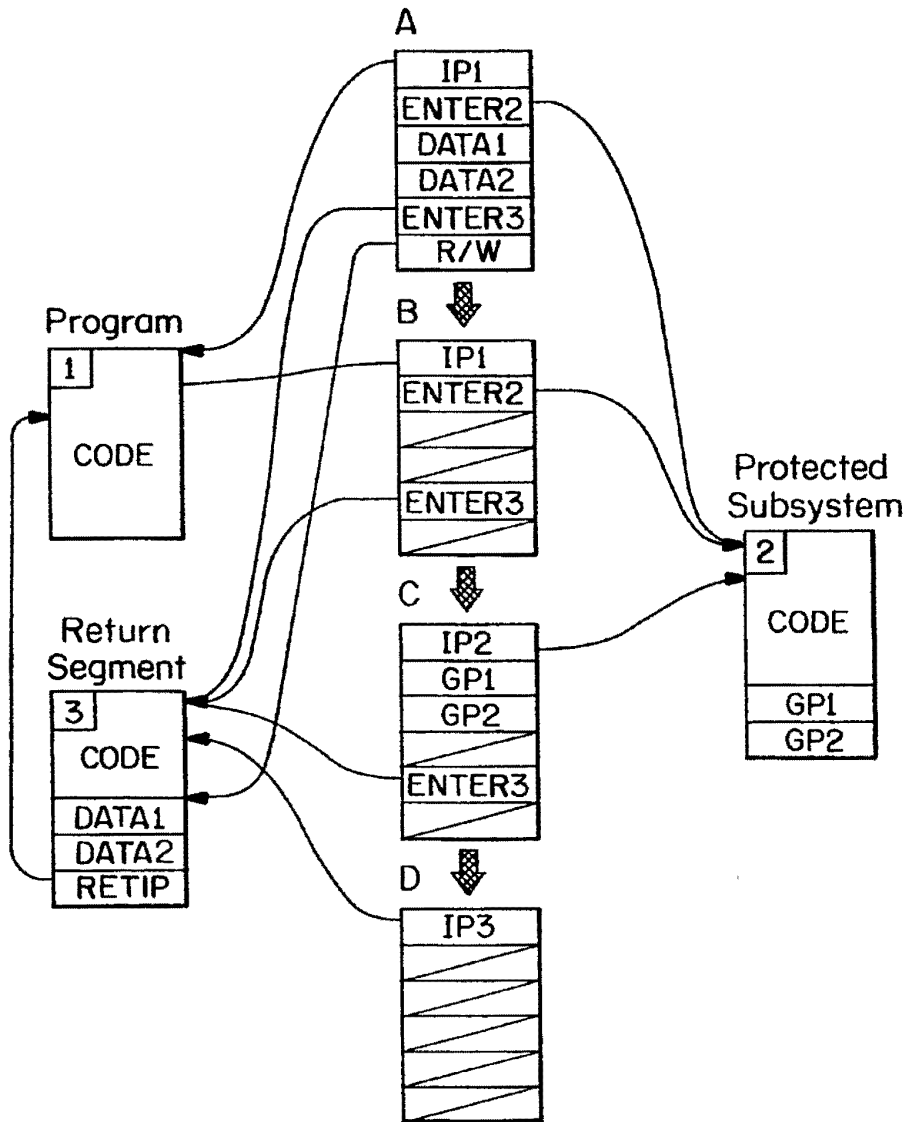


FIG. 7

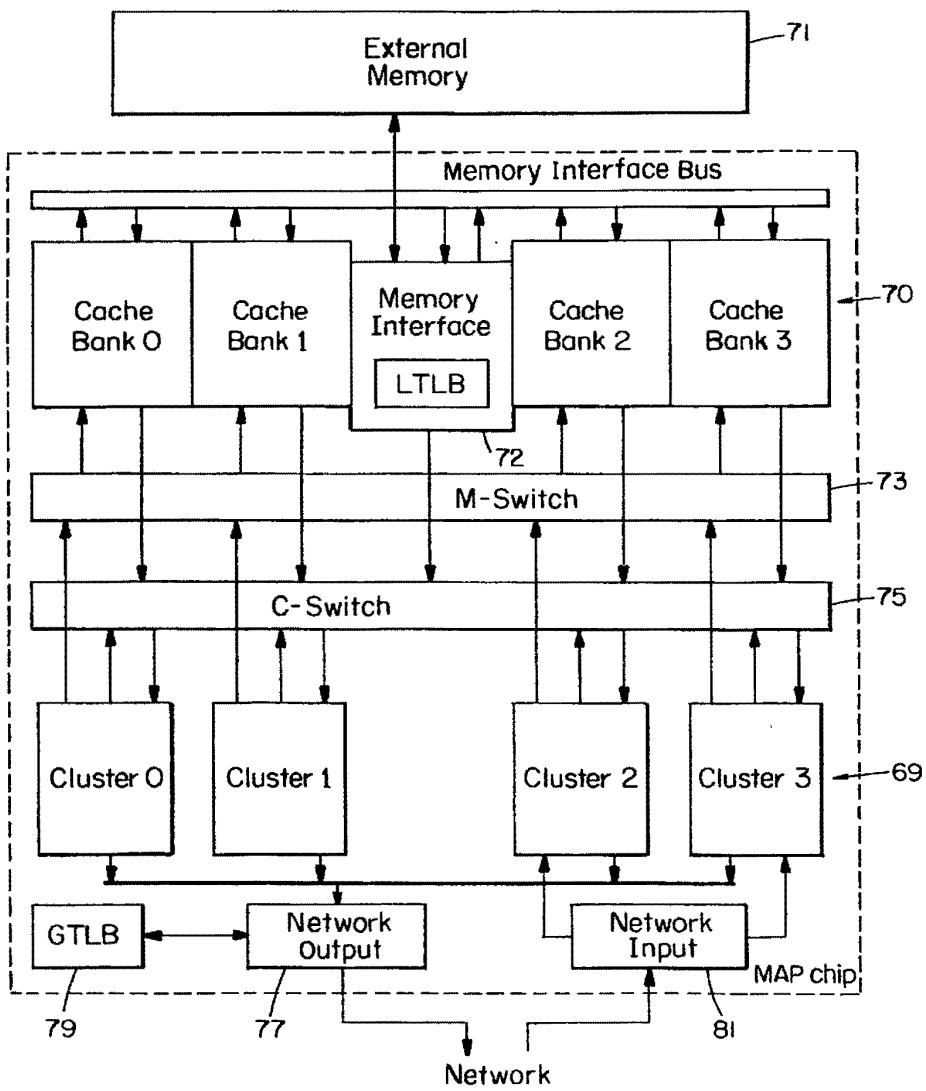


FIG. 8

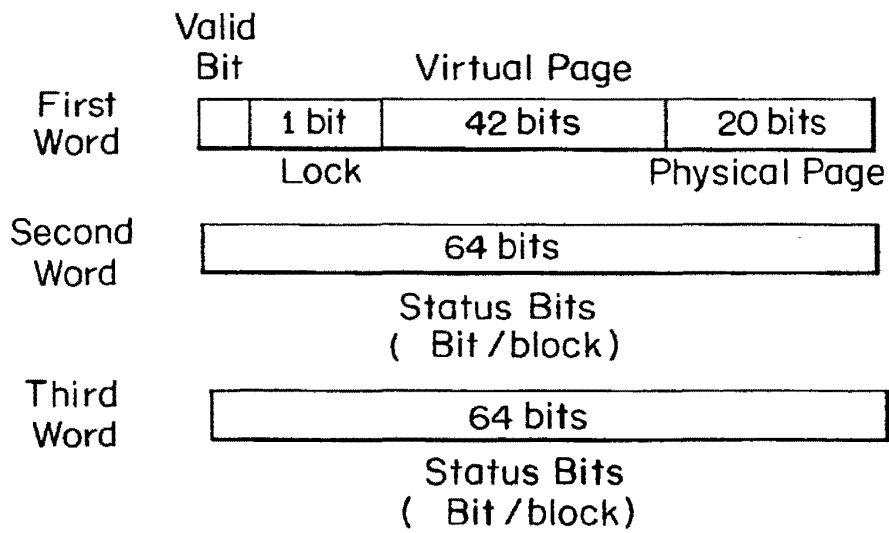


FIG. 9

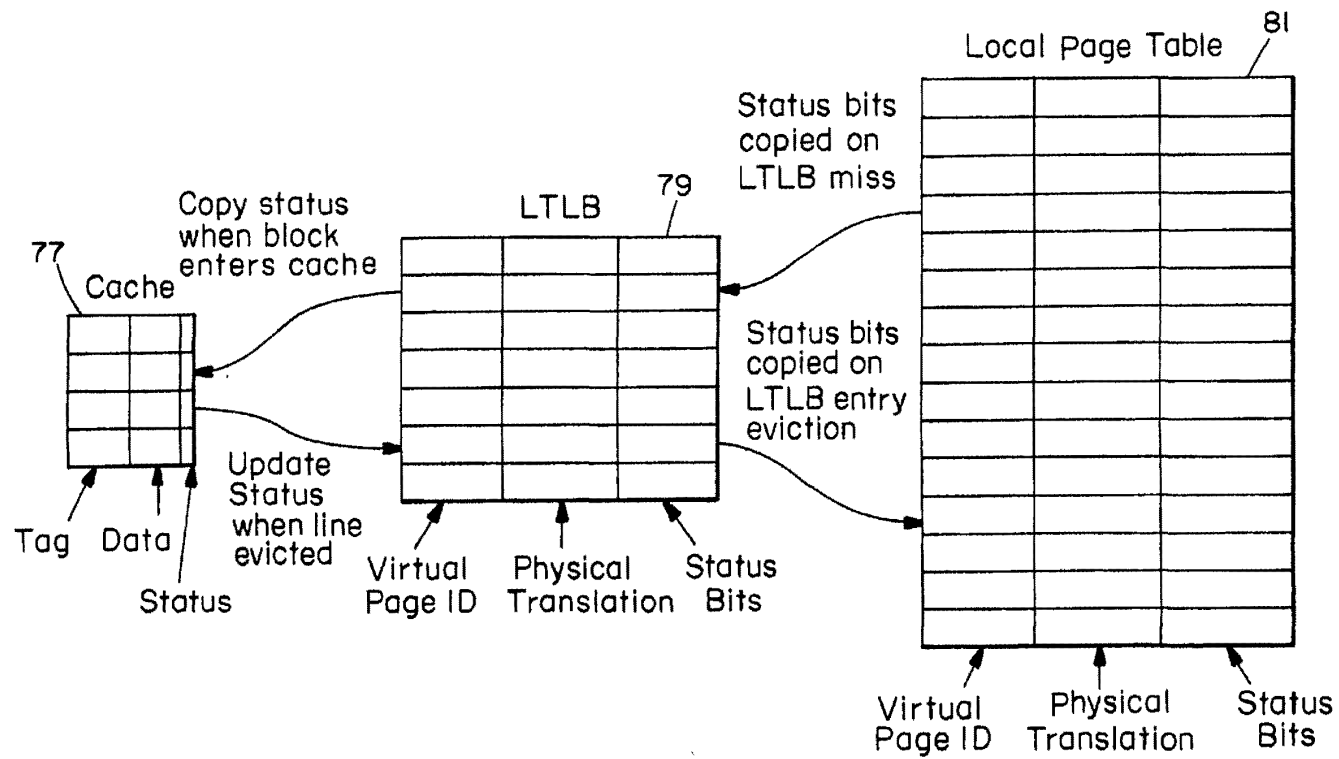


FIG. 10

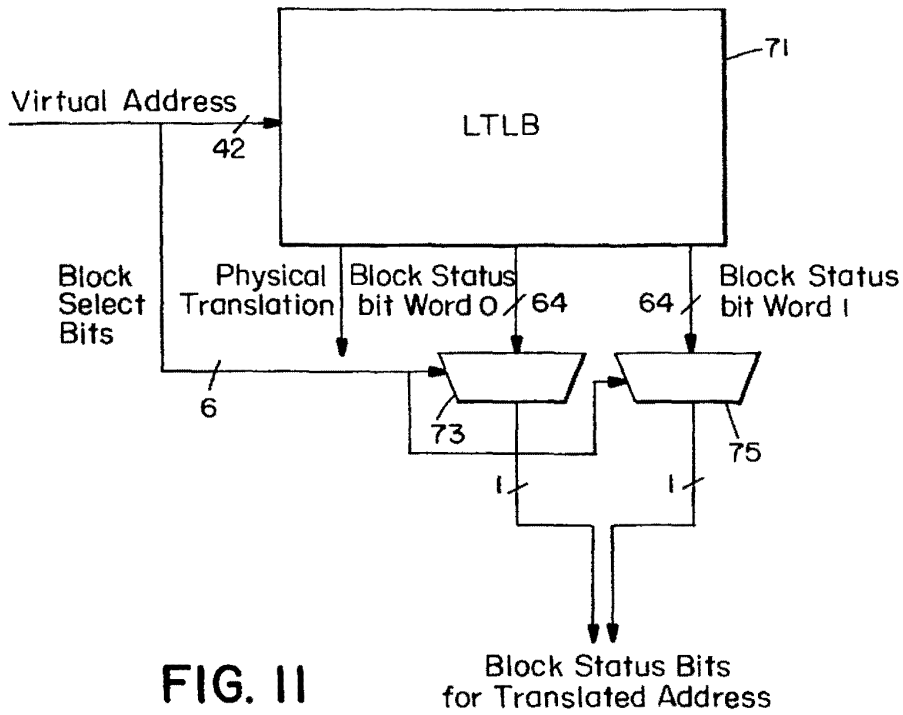


FIG. II

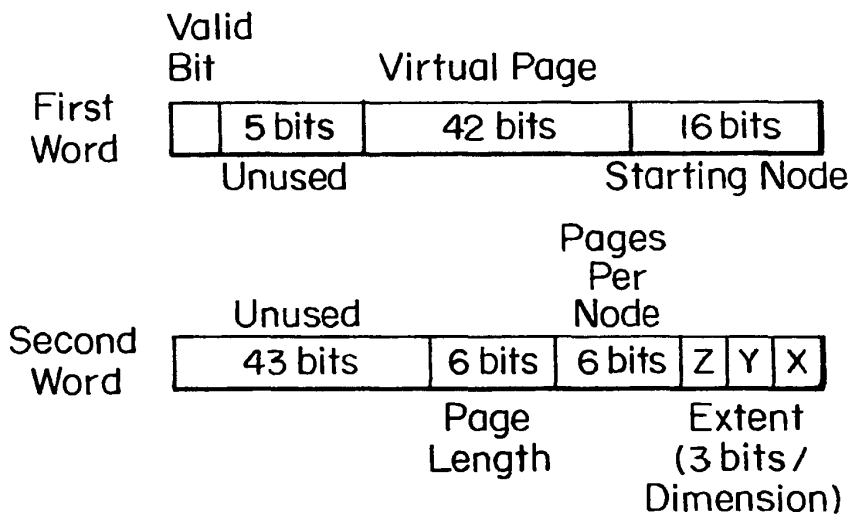


FIG. 13

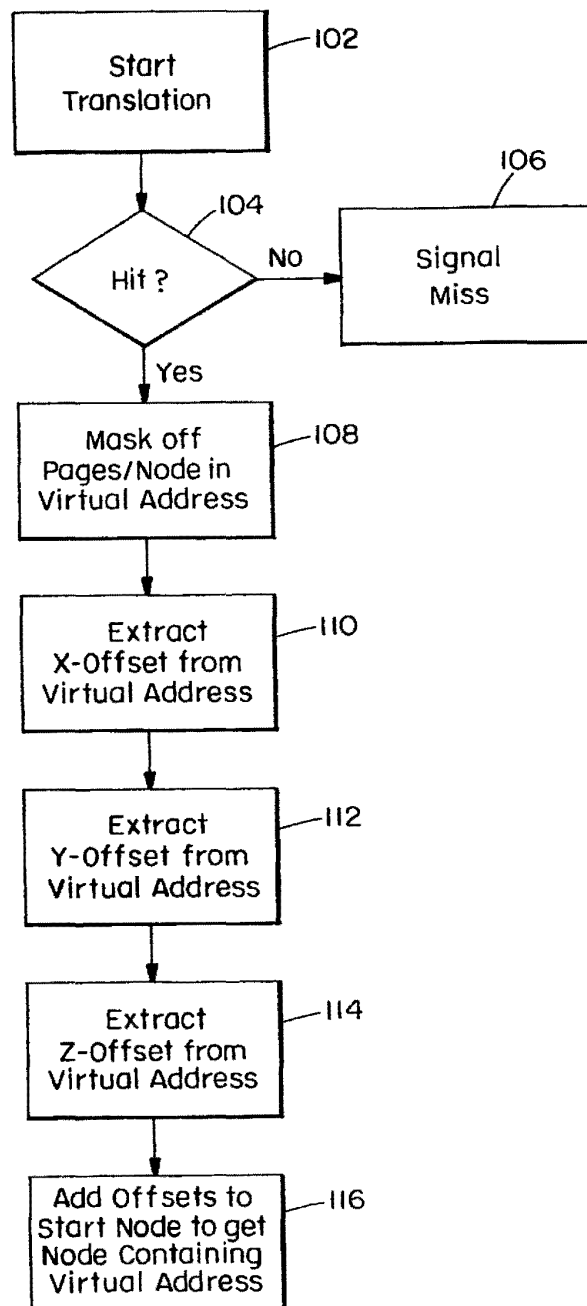


FIG. 14A

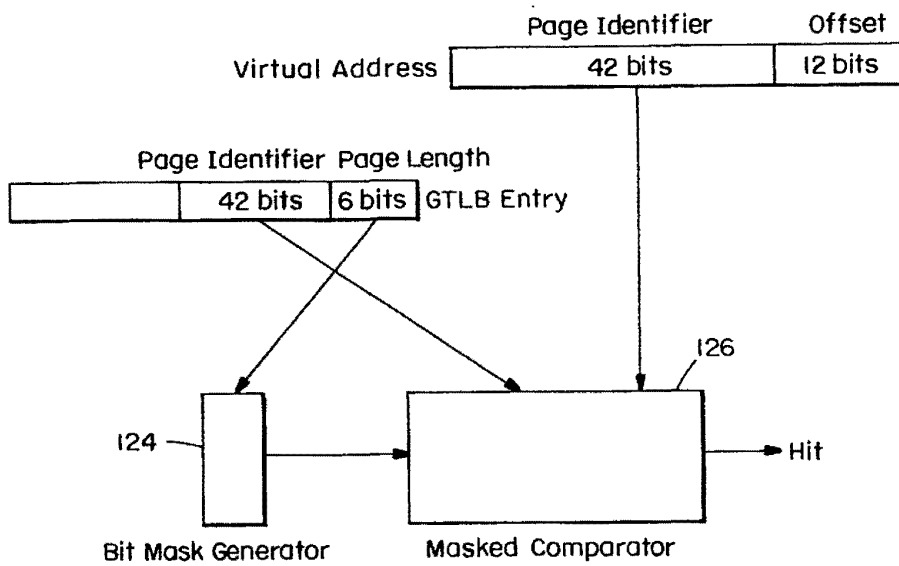


FIG. 14B

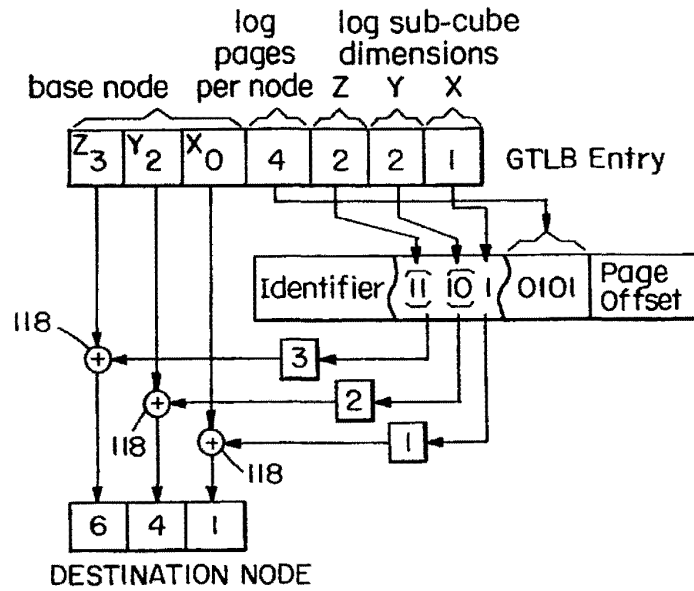


FIG. 15A

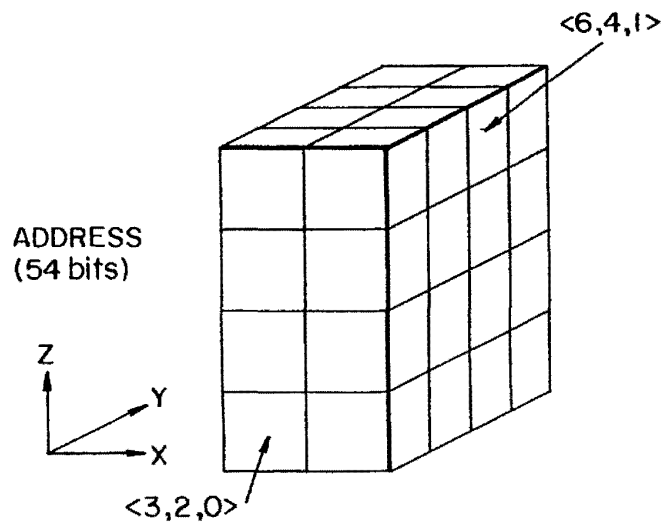


FIG. 15B

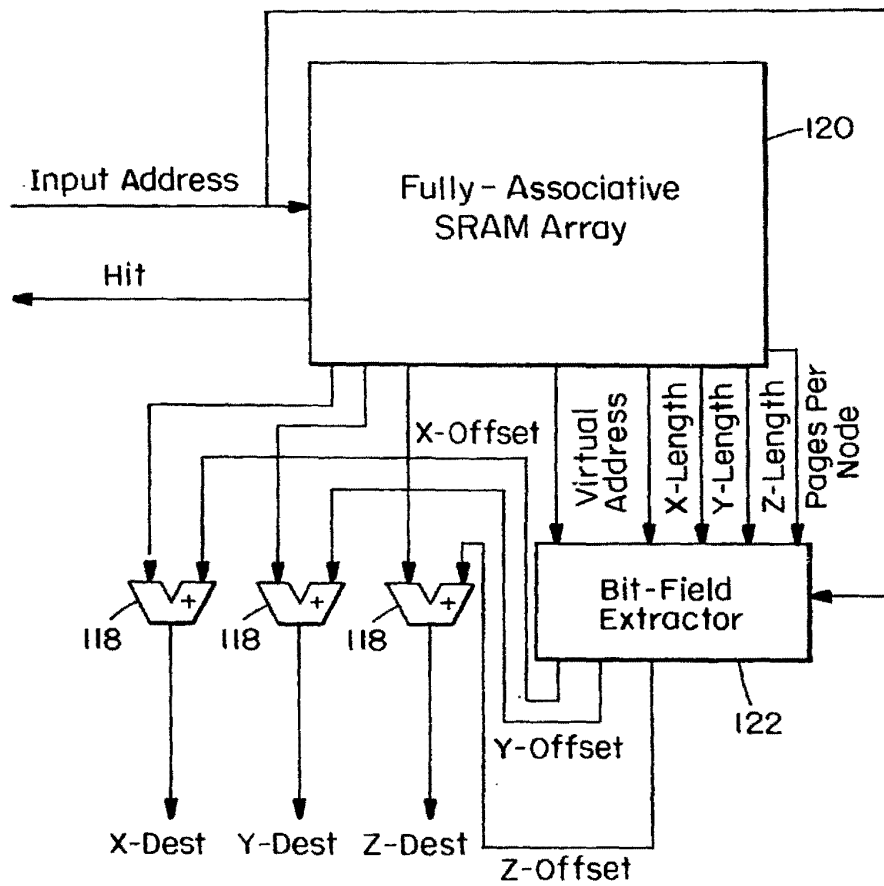


FIG. 16

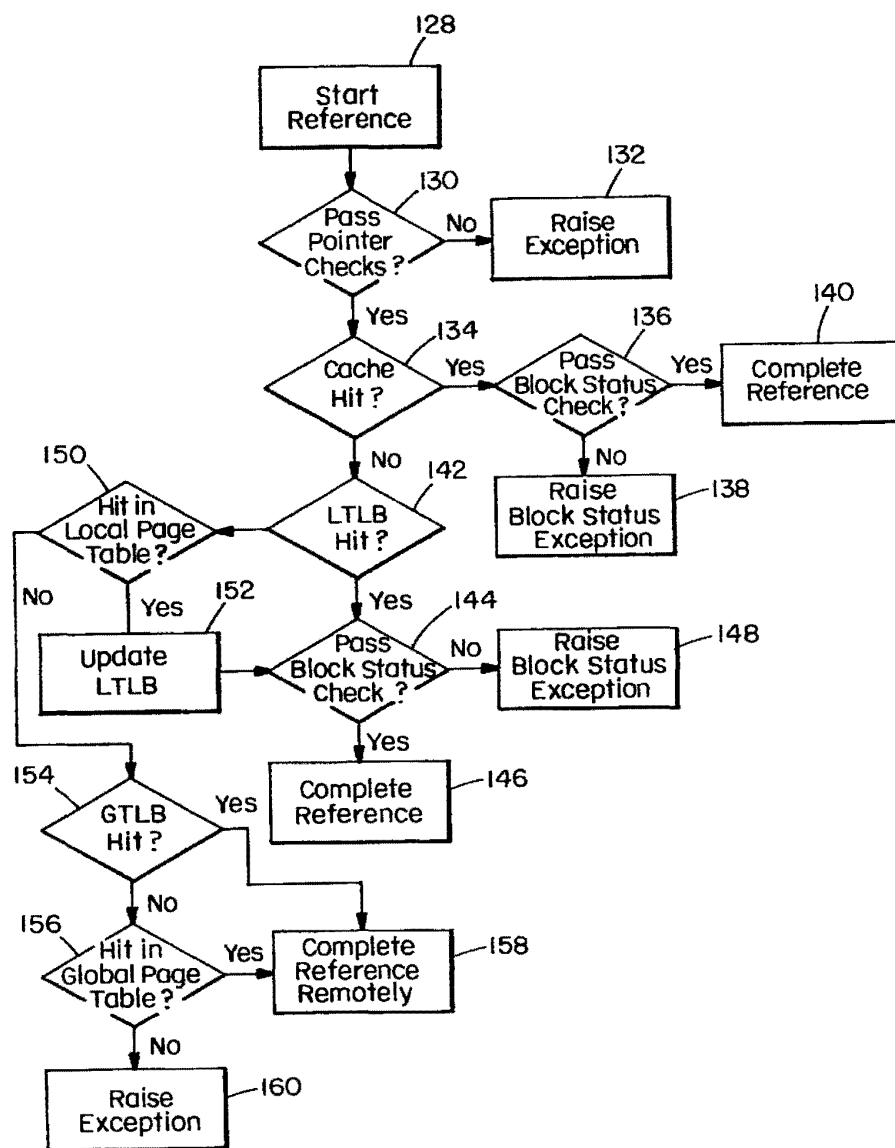


FIG. 17

MEMORY SYSTEM WITH GLOBAL ADDRESS TRANSLATION

RELATED APPLICATION

This application is a divisional of Ser. No. 08/314,013, filed Sep. 28, 1994, now U.S. Pat. No. 5,845,331, Dec. 1, 1998 the entire teachings of which are incorporated herein by reference.

GOVERNMENT SUPPORT

The invention was supported, in whole or in part, by a grant Contract No. F19628-92-C-0045 from the Air Force Electronic Systems Division. The Government has certain rights in the invention.

BACKGROUND OF THE INVENTION

In most computer systems, individual programs access code and data by addressing memory through a virtual address space. That virtual address space for each program must then be translated into the physical address space in which the code and data is actually stored in memory. Thus, distinct programs may use identical virtual addresses which translate to different locations in physical memory. The physical address space utilized by several programs may be completely distinct or they may overlap. Some level of security must be provided in order to permit common access to certain memory locations while protecting against unauthorized access to other locations.

Memory system designers must provide security without sacrificing efficiency and flexibility. One process' objects must be protected from modification by other, unauthorized processes, and user programs must not be allowed to affect the execution of trusted system programs. It must be possible to share data between processes in a manner that restricts data access to authorized processes; merely providing the ability to have data be private to a process or accessible to all processes is insufficient. An efficient mechanism must also be provided to change protection domains (the set of objects that can be referenced) when entering a subsystem.

The current trend towards the use of multithreading as a method of increasing the utilization of execution units make traditional security schemes undesirable, particularly if context switches may occur on a cycle-by-cycle basis. Traditional security systems have a non-zero context switch time as loading the protection domain for the new context may require installing new address translations or protection table entries.

A number of multithreaded systems such as Alewife (Agarwal, A., et al., "The MIT Alewife machine: A large-scale distributed-memory multiprocessor," *Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers, 1991.), and Tera (Alverson, R., et al., "The tera computer system," *Proceedings of the 1990 International Conference on Supercomputing*, September, 1990, ACM SIGPLAN Computer Architecture News, pp 1-6) have avoided this problem by requiring that all threads which are simultaneously loaded share the same address space and protection domain. This may be sufficient for execution of threads from a single user program, but disallows interleaving threads from different protection domains, which may restrict the performance of the machine.

SUMMARY OF THE INVENTION

The present invention relates to several aspects of a memory system which may be used independently or

together. The invention is particularly applicable in a virtual addressing, multiprocessor environment which requires sharing of data among multiple tasks across multiple nodes.

In accordance with one aspect of the invention, a data processing system comprises shared memory for storing instructions and data for plural programs, the shared memory being accessed in response to pointers. Guarded pointers address memory locations to which access is restricted. Each guarded pointer is a processor word which fully identifies a protected segment in memory and an address within the protected segment. Processor hardware distinguishes guarded pointers from other words and is operable under program control to modify guarded pointers. Modification of guarded pointers is restricted so that only addresses within the identified segment can be created. Thus, access outside of a protected segment is prevented. A permission field in the guarded pointer indicates permissible access to the identified memory segment such as read only or read/write. By providing the full virtual address, segment information, and a permission field, segment checks and permission checks can be performed during a memory access without requiring additional machine cycles.

Preferably, each guarded pointer comprises a length field and an address field. The value in the length field indicates a division of the address field into a segment subfield which identifies a segment location and an offset subfield which identifies an offset within an identified segment. The value in the length field is preferably logarithmically encoded using a base 2 logarithm. A tag field may be provided to identify the word as a guarded pointer, and the pointer must be so identified if it is to be used to access a memory location. By limiting the ability to set the flag bit and to freely modify addresses in pointers to the operating system, the creation of forged pointers by application programs to gain access to protected segments is avoided.

The processor hardware may be operable to generate a second guarded pointer from a first guarded pointer, the second guarded pointer identifying a subsegment within the segment identified by the first guarded pointer. To that end, the processor changes a value in the length field to decrease the number of bits in the offset subfield and to increase the number of bits in the segment subfield. The result is decreased offset range and finer segment location resolution within the original segment. However, the segment can not be enlarged by an application program.

The processor hardware may also be operable to generate a second guarded pointer from a first guarded pointer by performing an arithmetic operation on the offset. The processor hardware checks the second guarded pointer for overflow or underflow by detecting a change in value in the segment subfield. The hardware may also modify the permission field of a guarded pointer to generate a pointer having only more restricted access to the indicated segment. For example, a program having permission to read/write may create a pointer to the same memory segment with permission only to read.

ENTER guarded pointers may be restricted for processing by the processor hardware to only jump to the identified address within the protected segment and to execute. Such pointers allow access to code beginning at the pointer address but prevent bypass of portions of the code and prevent changing or copying of the code. Other preferred pointer types are read-only pointers, read/write pointers, execute pointers and key pointers. Key pointers may not be modified or used for data access.

In accordance with another aspect of the invention, a method is provided for global addressing across plural

processor nodes. A virtual address is applied to a global translation buffer to identify a mapping of a page group to a set of nodes in a system. From the virtual address and the identified mapping, the system determines a destination node at which a page containing the virtual address resides. A message including the address, which may be in a guarded pointer, may be forwarded to the destination node, and translation of the virtual address to a physical address may be performed at that node. By translating to groups of nodes, rather than an individual node for each virtual address, the size of the global translation buffer can be substantially reduced.

Preferably, the global translation buffer identifies each page group by a group size which is logarithmically encoded. It also specifies, in each group entry, a start node and the physical range of nodes within the group. Preferably, the range is specified in plural dimensions, specifically in the X, Y and Z dimensions of an array. That range is preferably also logarithmically encoded. Finally, the translation buffer may specify the number of pages of the page group per node.

In accordance with another aspect of the invention, virtual page addresses are translated to physical page addresses at each processor node and each virtual page is subdivided into blocks. At each processor node on which data from a virtual page is stored, a block status flag is provided for each block of the virtual page. Blocks of data may be copied between nodes and, based on the block status flags, access to individual blocks on a node is restricted. The use of the blocks allows for finer granularity in data transfers. The status flags are preferably stored in a virtual to physical translation buffer. Block status flags may also be stored with the data in cache memory, and the block status flags in the translation buffer may be updated from cache memory.

The preferred states of the status flags include invalid, read only, read/write and read/write but dirty. The dirty designation is provided in order to indicate to the home node that the data has been changed since being loaded from the home node.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

FIG. 1A illustrates the format of a guarded pointer embodying the present invention.

FIG. 1B illustrates a simple application of a guarded pointer having only a four bit address field.

FIG. 2A is a flow chart of a memory request in a system that includes guarded pointers.

FIG. 2B illustrates the hardware utilized in an LEA operation in which an offset is added to an existing pointer.

FIG. 3 illustrates the adder and segment check of FIG. 2B.

FIG. 4 illustrates the masked comparator of FIG. 3.

FIG. 5 illustrates a masked comparator bit cell in FIG. 4.

FIG. 6 illustrates register states when a program enters a protected subsystem by jumping to an enter pointer.

FIG. 7 illustrates register states when two way protection is provided by creating a return segment.

FIG. 8 is a block diagram of a processor chip used in an M-Machine embodying the present invention.

FIG. 9 illustrates an LTLB entry having block status bits in accordance with the present invention.

FIG. 10 illustrates status bit caching in a system using block status bits.

FIG. 11 is a block diagram of hardware utilized in determining status bits for a block in the LTLB.

FIG. 12 is a flow chart of a memory request in a system that includes block status bits.

FIG. 13 is an illustration of a GTLB entry in a system using global translation in accordance with the present invention.

FIG. 14A is a flow chart of a GTLB translation process.

FIG. 14B illustrates a masked comparator used in the GTLB.

FIG. 15A illustrates an example GTLB translation of an address, and FIG. 15B illustrates the node within a group identified by the translation of FIG. 15A.

FIG. 16 is a block diagram of a GTLB.

FIG. 17 is a flow chart of a memory request in a system that includes guarded pointers, block status bits, and a GTLB.

DETAILED DESCRIPTION OF THE INVENTION

Guarded Pointers

Guarded pointers are provided as a solution to the problem of providing efficient protection and sharing of data. Guarded pointers encode permission and segmentation information within tagged pointer objects to implement capability requirements of the type presented in Fabry, R., "Capability-based addressing," *Communications of the ACM* 17,7 (July 1974), 403-412. A guarded pointer may be stored in a general purpose register or in memory, eliminating the need for special storage. Because memory may be accessed directly using guarded pointers, higher performance may be achieved than with traditional implementations of capabilities, as table lookups to translate capabilities to virtual addresses are not required.

FIG. 1A shows the format of a guarded pointer. A single pointer bit is added to each 64-bit memory word. Fifty-four bits contain an address, while the remaining ten bits specify the set of operations that may be performed using the pointer (4 bits) and the length of the segment containing the pointer (6 bits). Segments are required to be a power of two words long, and to be aligned on their length. Thus, a guarded pointer specifies an address, the operations that can be performed using that address, and the segment containing the address. No segment or capability tables are required. Since protection information is encoded in pointers, it is possible for all processes to share the same virtual address space safely, eliminating the need to change the translation scheme on context switches and facilitating the use of virtually-addressed caches.

All memory operations in a system that use guarded pointers require that one of their operands be a guarded pointer and that the permission field of the pointer allow the operation being attempted. Users are not allowed to set the pointer bit of a word, although they may manipulate pointers with instructions that maintain the protection scheme. This prevents users from creating arbitrary pointers, while allowing address arithmetic within the segments that have been allocated to a user program. Privileged programs are allowed to set the pointer bit of a word and thus can create arbitrary pointers.

Memory systems that use guarded pointers provide a single virtual address space shared by all processes. Each

guarded pointer identifies a segment of this address space that may be any power of two bytes in length, and must be aligned on its size. These restrictions allow six bits of segment length information and 54 bits of virtual address to completely specify a segment. The length field of a guarded pointer encodes the base-two logarithm of the segment length. This allows segments ranging in length from a single byte up to the entire 2^{34} byte address space in power of two increments. As shown in FIG. 1 the length field also divides the address into segment identifier and offset fields. A four-bit permission field completes the capability by identifying the set of operations permitted on the segment.

FIG. 1B presents a simple illustration of the segment length and address fields of the guarded pointer assuming an address field of only 4 bits and a length field of 3 bits. With the length L equal to zero, each segment is of length $2^0=1$ word in length. As illustrated by the vertical broken line, the segment length L positions the division between offset and segment to the far right of the address, so there would be no offset. Each segment base address would also be the address of the addressed word. With L equal one, each segment is of $2^1=2$ words long. The broken line indicates a one bit offset. Where the full address is 1011, the base address 1010 of the segment is defined by setting the offset to zero.

Similarly, with increasing values of L the number of words in the segment defined by the guarded pointer increases exponentially, and the base address for the segment is defined by setting all offset bits to zero.

It can be seen from FIG. 1B that two pointers having a common address 1011 may indicate that the address is within a segment ranging in length from one byte to 16 bytes. Since the base address is determined by setting the offset to zero, segments must be a power of two words long and must be aligned on their length. As discussed below, the segment definition is important to define the segment of addresses within which a particular program may operate by modifying a given pointer. Generally, permission is granted to modify addresses only within a segment.

The permission field of a pointer indicates how a process may access the data within the segment. Pointer permissions may specify data access, code access, protected entry points, and protected identifiers (keys). The permissions granted are with respect to use of the pointers. All pointers may themselves be stored in memory and loaded from memory. It is use of the pointers to access data at the indicated addresses which is restricted. The following is a representative set of permissions:

A Read Only pointer may be used to load data and the pointer may be altered within segment bounds. Store and jump operations using the pointer are not permitted.

A Read/Write pointer may be used by load and store operations, but not jump operations. It may be altered within its segment bounds.

Execute pointers may be used by jump and load operations and may be modified within segment bounds. Thus, holding an execute pointer to a code segment enables a program to jump to any location within the segment and to read the segment. Execute pointers may be either execute-user or execute-privileged, which encodes the supervisor mode bit explicitly within the instruction pointer. Privileged instructions, such as SETPTR, may only be executed with an execute-privileged instruction pointer.

Enter pointers may be used only by jump operations. They cannot be used for loads and cannot be modified in any way. Thus, holding an enter pointer enables a program

to enter a code segment at a single location. Jumping to an enter pointer converts it to an execute pointer which is then loaded into the instruction pointer. There are two types of enter pointers: enter-user and enter-privileged, which are converted to the corresponding type of execute pointer by a jump.

A Key pointer may not be modified or referenced in any way. It may be used as an unforgeable, unalterable identifier.

Physical: The pointer references data in physical memory on the local node. This bypasses the virtual memory system ignoring the LTLB on cache misses. If the address exceeds the size of local physical memory, the top bits are ignored.

Since the set of pointer states does not require all of the possible four bit values, architects may implement pointer types to support particular features of their architecture, such as the following pointer types, which are implemented on the M-Machine.

Execute Physical: Data may be read or executed as code, but not written. On cache misses, the TLB is not accessed. The thread is in privileged mode.

Enter Message: Code at this address may be executed in a message handler. A send operation faults if the designated IP is not in this state.

Configuration Space: Indicates that the address refers to an internal register in the processor.

Errval: The pointer has been generated by a deferred exception. Any attempt to use an Errval pointer as an operand will cause an exception.

As noted, each pointer contains a six bit segment length field that holds the log base 2 of the size of the segment in which the address resides. Thus, segments may be of any power of 2 size between 1 and 2^{34} bytes. This encoding allows the base address and the extent of a pointer's segment to be determined without accessing any additional information. User-level instructions that manipulate pointers (LEA, LOAD, STORE) have the lower and upper bounds of their segment checked automatically to ensure that the operation does not access memory outside of the allowed segment.

This segmentation and access control system provides flexibility to the user, while still permitting strictly enforced security. Segments can be overlapped and shared as long as each segment is aligned to an address that is a multiple of its size. Since all of the necessary segmentation information is contained within each pointer, a separate table of segment descriptors is unnecessary. More importantly, instructions need not access such a table to check segmentation restrictions on memory accesses. Also, access to system functions and other routines can be given to non-trusted programs, as the enter-privileged and enter-user permission states ensure that a user may only execute code starting at the specified entry point. A MEMBAR (memory barrier) instruction is used to block further instructions from executing until all outstanding memory references have completed.

Pointer Operations

Guarded pointers may be implemented by adding a few pointer manipulation instructions to the architecture of a conventional machine and adding checking hardware to verify that each instruction operates only on legal pointer types and that address calculations remain within pointer bounds.

FIG. 2A shows a flow chart of the steps involved in performing a memory reference beginning at 20 in a system that incorporates Guarded Pointers. First, the pointer bit of the operand containing the address being referenced is

checked at 22 to determine if the address operand is a guarded pointer. If the pointer bit is not set, an exception occurs at 24. Second, the permission field of the pointer is checked at 26 and 28 to verify that it allows the operation being attempted, and an exception raised at 30 if it does not. If the operation involves address computation, an integer offset is then added to the address field of the pointer at 32. Segmentation violation is checked at 34 and 36. An exception 38 is raised if the result of this add overflows or underflows into the fixed segment portion of the address, which would create a pointer outside the original segment. If all of these checks pass, the operation is submitted to the memory system at 40 to be resolved.

Load/Store: Every load or store operation requires a guarded pointer of an appropriate type as its address argument. Protection violations are detected by checking the permission field of the pointer. If the address is modified by an indexed or displacement addressing mode, bounds violations are checked by examining the length field as described below. The protection provided by guarded pointers does not slow load or store operations. All checks are made before the operation is issued without reference to any permission tables. Once these initial checks are performed, the access is guaranteed not to cause a protection violation, although events in the memory system, such as TLB misses, may still occur.

Pointer Arithmetic: An LEA (load effective address) instruction may be used to calculate new pointers from existing pointers. This instruction adds an integer offset to a data (read or read/write) or execute pointer to produce a new pointer. An exception is raised if the new pointer would lie outside the segment defined by the original pointer. For efficiency, a LEAB operation, which adds an offset to the base of the segment contained in a pointer may be implemented, as well. If a guarded pointer is used as an input to a non-pointer operation, the pointer bit of the guarded pointer is cleared, which converts the pointer into an integer with the same bit fields as the original pointer.

FIG. 2B details the protection check hardware used on a pointer calculation. The permission field of the pointer 42 is checked at 44 against the opcode 46 to verify that the requested operation using the pointer is permissible. In that respect, the permission check hardware need only decode the opcode to identify permission bits which are appropriate for that opcode and compare those bits to the permission bits of the pointer 42 in combinational logic. An integer offset 48 may be added to the address field of the pointer at 50 to generate the new pointer 54. An exception is raised if the result of this add over or underflows into the fixed segment portion of the address, which would create a pointer outside the original segment. This may be detected in the segment check 52 by comparing the fixed portion of the address before the add to the same field of the resulting pointer.

FIGS. 3, 4 and 5 show in greater detail the hardware of FIG. 2B used in performing an address calculation on a guarded pointer. The 54-bit address field of the pointer is added in adder 56 to a 54-bit offset to get the result address. The 6-bit length field of the pointer is fed to a mask generator 58, which generates a 54-bit output applied as a mask to masked comparator 60. Each bit in this output is set to one if the corresponding bit in the address represents a bit in the segment identifier and to zero if the bit represents a bit in the offset portion of the address. Bits in the offset portion of the address are allowed to change during address calculation, while bits in the segment identifier are not.

FIG. 4 illustrates the masked comparator 60. Each bit of the original address, the corresponding bit of the result

address, and the corresponding bit of the mask are fed into a comparator cell 62, as shown in FIG. 5. The output of XOR gate 64 is one if the bit from the original address and the bit from the result address differ. This output is then ANDed at 66 with the bit from the bit mask, which is one if the bit being examined is part of the segment portion of the address, and therefore not allowed to change. The outputs of all the comparator cells are ORed together at 68 to determine if any of the segment bits changed during the addition of the offset, which indicates that a segmentation violation has occurred.

Guarded pointers expose to the compiler address calculations that are performed implicitly by hardware in conventional implementations of segmentation or capabilities. With the conventional approach, the segmentation hardware performs many redundant adds to relocate a series of related addresses. Consider, for example, the following loop:

```
for (i=0; i<N; i++) s=s+d[i];
```

In a conventional system, each reference to array *a* would require the segmentation hardware to automatically add the segment offset for each *a[i]* to the segment base. With guarded pointers, the add can be performed once in software, and then the resulting pointer can be incrementally stepped through the array, avoiding the additional level of indirection.

Languages that permit arbitrary pointer arithmetic or type casts between pointers and integers, such as C, are handled by defining code sequences to convert between pointer and integer types. The pointer-to-integer cast operation takes a guarded pointer as its input and returns an integer containing the offset field of the guarded pointer. This can be performed by subtracting the segment base, determined using the LEAB instruction, from the pointer:

```
LEAB Ptr, 0, Base SUB Ptr, Base, Int
```

The integer-to-pointer case operation uses the LEAB instruction to take an integer and create a pointer into the data segment of the process with the integer as its offset, as long as the integer fits into the offset field of the data segment. Note that neither of these case operations requires any privileged operations, which allows them to be inlined into user code and exposed to the compiler for optimization.

Pointer Creation: A process executing in privileged mode, with an execute-privileged IP, has the ability to create arbitrary pointers and hence access the entire address space. Privileged mode is entered by jumping to an enter-privileged pointer. It is exited by jumping to a user pointer (enter or execute). While in privilege mode, a process may execute a SETPTR instruction to convert an integer into a pointer by setting the guarded pointer bit. Thus, a privileged process may amplify pointer permissions and increase segment lengths while a user process can only restrict access. No other operations need be privileged, as guarded pointers can be used to control access to protected objects such as system tables and I/O devices.

Restricting Access: A process may create pointers with restricted permissions from those pointers that it holds. This allows a process to share part of its address space with another process or to grant another process read-only access to a segment to which it holds read/write permission.

A RESTRICT instruction takes a pointer, *P*, and an integer permission type, *T*, and creates a new pointer by substituting *T* for the protection field of *P*. The substitution is performed only if *T* represents a strict subset of the permissions of *P* so that the new pointer has only a more restricted access. For example, a read pointer may be created from a read/write pointer, but not vice versa. Otherwise, an exception is raised.

Similarly the SUBSEG instruction takes an integer length, *L*, and a pointer, *P*, and substitutes *L* into *P* if *L* is less than

the original length field of P, so that the created segment is a subset of the original. Changing to a lesser length decreases the length of the offset subfield for decreased offset range and increases the length of the segment field for finer segment location resolution.

The RESTRICT and SUBSEG instructions allow a user process to control access to its memory space efficiently, without system software interaction. The RESTRICT and SUBSEG instructions are not completely necessary, as they can be emulated by providing user processes with enter-privileged pointers to routines that use the SETPTR instruction to create new pointers that have restricted access rights or segment boundaries. The M-Machine, which will be described in the next section, takes this approach.

Pointer Identification: The ISPOINTER instruction is provided to determine whether a given word is a guarded pointer. This instruction checks the pointer bit and returns its state as an integer. Quick pointer determination is useful for programming systems that provide automatic storage reclamation, such as LISP, which need to find pointers in order to garbage collect physical space (Moon, D. A. *Symbolics Architecture*, IEEE Computer (1987), 43-52).

Protected Subsystems

ENTER pointers facilitate the implementation of protected subsystems without kernel intervention. A protected subsystem can be entered only at specific places and may execute in a different protection domain than its caller. Entry into a protected subsystem, such as a file system manager, is illustrated in FIG. 6. A program enters a protected subsystem by jumping to an enter pointer. After entry the subsystem code loads pointers to its data structures from the code segment. A represents the register state of the machine before the protected subsystem call, B the register state just after the call, C the register state during the execution of the protected subsystem, and D the register state immediately after the return to the caller.

Before the call, the calling program (segment 1) holds an enter pointer to the subsystem's code segment (segment 2) which contains the subsystem code as well as pointers to the subsystem's data segments, such as the file system tables. To enter the subsystem, the caller jumps to ENTER2, causing the hardware to transfer control to the entry point and convert the enter pointer to the execute pointer IP2 in register state B. The return instruction pointer (RETIP) is passed as an argument to the subsystem. The subsystem then uses the execute pointer to load GP1 and GP2, the pointers to its data structures (state C). The subsystem returns to the calling program by overwriting any registers containing private pointers and jumping to RETIP (state D).

The sequence described above provides one-way protection, protecting the subsystem's data structures from the caller. To provide two-way protection, the caller (segment 1) encapsulates its protection domain in a return segment (segment 3) as shown in FIG. 7. Before the call (state A), the caller holds both enter and read/write pointers to a return segment. The caller writes all the live pointers in its registers into the return segment to protect them from the subsystem (segment 2). It then overwrites all of the pointers in its register file except the enter pointer to the return segment (ENTER3), the subsystem enter pointer (ENTER2), and any arguments for the call (state B). The subsystem call then proceeds as described above. After entry, the subsystem holds only an enter pointer to the return segment and thus cannot directly access any of the data segments in the caller's protection domain (state C). The subsystem returns by jumping to the return segment (state D), which reloads the caller's saved pointers and returns to the calling program.

ENTER pointers allow efficient realization of protected system services and modular user programs that enforce access methods to data structures. Modules of an operating system, e.g., the file-system, can be implemented as unprivileged protected subsystems that contain pointers to appropriate data structures. Since these data structures cannot be accessed from outside the protected subsystem, the file-system's data structures are protected from unauthorized use. Even an I/O driver can be implemented as an unprivileged protected subsystem by protecting access to the read/write pointer of a memory-mapped I/O device. With protected entry to user-level subsystems, very few services actually need to be privileged.

Implementation Costs

Hardware: Guarded pointers have two hardware costs: an increase in the amount of memory required by a system and the hardware required to perform permission checking. To prevent unauthorized creation or alteration of a guarded pointer, a single tag bit is required on all memory words, which results in a 1.5% increase in the amount of memory required by the system.

The hardware required to perform permission checking on memory access, and segment bounds checking on pointer manipulation, is minimal. One decoder for the permission field of the pointer, one decoder for the opcode of the instruction being executed, and a small amount of random logic are required to determine if the operation is allowed. The pointer bit of an operand can be checked at the same time, to determine if it is a legal pointer. To check for segment bounds violations when altering a pointer, a masked comparator is needed. It compares the address before and after alteration and signals a fault if any of the segment bits of the address field change.

Memory systems based on guarded pointers do not require any segmentation tables or protection lookaside buffers in hardware, nor is it necessary to annotate cached virtual-physical translations with a process or address space identifier. As with other single address space systems, the cache may be virtually addressed, requiring translation only on cache misses.

Address Space: Since 6 to 10 bits are required to encode the permission and segment length field of a guarded pointer, the virtual address space is reduced. On a 64-bit machine, a guarded pointer virtual address is 54 bits, which provides 16 petabytes of virtual address space, enough for the immediate future. Several current processors support 64-bit addresses, but only translate some of the bits in each address. For example, the DEC Alpha 21064 only translates 43 bits of each 64-bit address (Digital Equipment Corporation, *Alpha Architecture Handbook*, Maynard, Mass., 1992).

There is an opportunity cost associated with reducing the virtual address space, however. Some system designers take advantage of large virtual address spaces to provide a level of security through sparse placement of objects. For example, the Amoeba distributed operating system (Mullender, S. J., Van Rossun, G., Tanenbaum, A. S., Van Renesse, R. and Van Staveren, H., "Amoeba: A distributed operating system for the 1990s" *IEEE Computer* 23 (May 1990), 44-53) protects objects using a software capability scheme. These capabilities are kept secret by embedding them in a huge virtual address space. This becomes less attractive if the virtual address space is 1000 times smaller. Of course, this particular use of a sparse virtual address space can be replaced by the capability mechanism provided by guarded pointers.

Virtual address space fragmentation is another potential problem with guarded pointers, as segments must be powers