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 (1) OF (2) PART(S)

Class	Subclass
370	392

ISSUE CLASSIFICATION

PATENT NUMBER

6771646

T-4

U.S. UTILITY Patent Application

O.I.P.E. SCANNED Q.A.

PATENT DATE: AUG 03 2004

APPLICATION NO.	CONT/PRIOR	CLASS	SUBCLASS	ART UNIT	EXAMINER
09/608266	D	370		2664	Nguyen, A.

APPLICANTS

TITLE

Certificate NOV 16 2004 of Correction

Certificate SEP 21 2004 of Correction

PTO-2040 12/99

CERTIFICATE OCT 28 2014

ISSUING CLASSIFICATION

ORIGINAL CLASS	ORIGINAL SUBCLASS	CROSS REFERENCE(S) CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)					
370	392	370	412	252				
INTERNATIONAL CLASSIFICATION		370	352					
G01R	31/08	701	223					
		711	119					

☐ Continued on Issue Slip Inside File Jacket

TERMINAL DISCLAIMER	DRAWINGS			CLAIMS ALLOWED	
	Sheets Drwg.	Figs. Drwg.	Print Fig.	Total Claims	Print Claim for O.G.
	21	23	15	20	7
☐ The term of this patent subsequent to ________ (date) has been disclaimed.	Alan V. Nguyen (Assistant Examiner)		4/19/04 (Date)	NOTICE OF ALLOWANCE MAILED 4/20/04	
☐ The term of this patent shall not extend beyond the expiration date of U.S Patent. No. ________	RICKY NGO PRIMARY EXAMINER (Primary Examiner)		4/16/04 (Date)	ISSUE FEE Amount Due 1330	Date Paid 6/4/04
☐ The terminal ____months of this patent have been disclaimed.	(Legal Instruments Examiner)		4/23/04 (Date)	ISSUE BATCH NUMBER	

WARNING:
The information disclosed herein may be restricted Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368 Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors-only.

Form PTO-436A (Rev. 6/99)

FILED WITH: ☐ DISK (CRF) ☐ FICHE ☐ CD-ROM (Attached in pocket on right inside flap)

ISSUE FEE IN FILE

(FACE)

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 / 35 USC 119 (a-d) conditions met / Verified and Acknowledged		STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
Foreign Priority claimed	☐ yes ☒ no	TX	21	20	3
35 USC 119 (a-d) conditions met	☐ yes ☒ no ☐ 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	FEES: Authority has been given in Paper No. _________ to charge/credit DEPOSIT ACCOUNT No. _________ for following:	
840		☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other ____________ ☐ 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:

X 65 sheet(s) of specification, claims, and abstract
X 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 ANAPPLICATION, 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.
X 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: [signature]

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

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.* 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: EI417961895US 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.: 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.

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

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.

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.

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.

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 ~~application~~ No. 6,651,099 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. 6,651,099 ~~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.

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 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 (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.

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 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.

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 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 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.

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 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.

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, *etc.*
4	Transport	TCP, Novel SPX, UDP, *etc.*
3	Network	IP, Novell IPX, VIP, AppleTalk, *etc.*
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 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.

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 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.

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 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 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.

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 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 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 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.

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 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 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 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 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 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 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 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.

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.

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 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 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 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 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 if 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.

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.

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 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 state to the next state according to a set of state instructions obtained form 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 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 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 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 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 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 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 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 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 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 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 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 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 (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 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 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 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 (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 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 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.

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 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.

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 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 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 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 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 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.

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 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 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 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 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 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 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 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 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 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 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 a find the instructions in the extraction operations database memory (i.e., slicer instruction database) 1002.

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 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 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 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 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 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 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.

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 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 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 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 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.

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 in set in the flow key of the UFKB record, a protocol 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 form 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 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-or-ing them together.

Thus, after the UFKB sets the program counter, a sequence of one or more state operations are be 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 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 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 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 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 1108 can now obtain 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 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 (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 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 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 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 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.

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 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 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 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 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 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 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 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 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 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 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.

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.

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, 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.

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.

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).

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.

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'.

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'.

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

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.
- 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.

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-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 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 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 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 "C_1". If such address identifies the server 110 (shown also as server 204), the label used in the drawing is "S_1". The first two fields 214 and 215 in packet 206 are "S_1" and C_1" because packet 206 is provided from the server 110 and is destined for the client 106. Suppose for this example, "S_1" is an address numerically less than address "C_1". A third field "p^1" 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 "i^1" 219 is an element that is being requested by the client from the server. A seventh field denoted "s_1a" 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 "s_1a". 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.

Packet 207 includes ten fields 224–233. The destination and source addresses are carried in fields 224 and 225, *e.g.,* indicated "C_1" and "S_1", respectively. Notice the order is now reversed, since the client-server message direction is from the server 110 to the client 106. The protocol "p^1" is used as indicated in field 226. The request "i^1" is in field 229. Values have been filled in for the application port number, *e.g.,* in field 233 and protocol ""p^2"" 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 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 operations. In this case, therefore, since we assume "S_1"<"C_1", the order is address "S_1" followed by client address "C_1". 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 "p^1". 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 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 p^1 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 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 "a^1" (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 "S_1" followed by (the numerically higher address) client "C_1". A protocol field 252 defines the protocol to be used, *e.g.*, "p^2" 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 "a^2". 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 "st^1". 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 "a^2". 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 "S_1" followed by client "C_1". 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 S_1 and C_1, in a pair of fields 260 and 261. A field 262 defines the protocol as "p^2", 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.

Thus the flow signature for the recognition of application "a^2" 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 complex state transition diagram to be traversed before a "final" resting state such as "st_1" 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 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.

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 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.

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 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.

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 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 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 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.

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.

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 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 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 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 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 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, …, 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 $hash_4$, and a lookup with $hash_4$ will produce a match and the address $page_4$ in cache memory. Furthermore,

$page_4$ in cache memory contains the flow-entry, $entry_4$, that in this notation is the flow-entry matching hash value $hash_4$. This table also indicates that $hash_0$ was more recently used than $hash_1$, $hash_5$ more recently than $hash_2$, and so forth, with $hash_{31}$ 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 $hash_{31}$. The LUE looks up the cache subsystem via the CAM array. CAM_{31} gets a hit and returns the page number of the hit, i.e., $page_{31}$. The cache subsystem now indicates to the LUE 1007 that the supplied hash value produced a hit and provides a pointer to $page_{31}$ of the cache memory which contains the flow-entry corresponding to $hash_{31}$, i.e., $flow_{31}$. The LUE now retrieve the flow-entry $flow_{31}$ from the cache memory at address $page_{31}$. In the preferred embodiment, the lookup of the cache takes only one clock cycle.

The value $hash_{31}$ 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 $hash_{31}$ is put into CAM_0 (pointing to $page_{31}$). Furthermore, $hash_{30}$ is now the LRU hash value, so is moved to CAM_{31}. The next least recently used hash value, $hash_{29}$ is now moved to CAM_{30}, 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 $hash_5$. This produces a hit in CAM_6 pointing to $page_5$ 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 $hash_5$, so $hash_5$ and cache memory address $page_6$ are entered into CAM_0. The contents of the remaining CAMs are all shifted down one up to and including the entry that contained $hash_5$. That is, CAM_7, CAM_8, …, CAM_{31} 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 $hash_{43}$. 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 $hash_{43}$. 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—$entry_{43}$— in database 324 matching hash value $hash_{43}$. 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 CAM_{31} 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 $hash_{43}$ is put in CAM_{31} and flow-entry $entry_{43}$ is placed in the cache page pointed to by CAM 31. The CAM array and 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 CAM_{31} are now moved to CAM_0 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 CAM_{31} are now moved to CAM_0 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, $hash_{43}$ is in MRU CAM_0, CAM_0 correctly points to $page_{30}$, but the information in $page_{30}$ of the cache, $entry_{43}$, does not correspond to $entry_{43}$ in database 324. That is, the contents of cache page $page_{30}$ 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 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.

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 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:

(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.

(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 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 system1115. Updating cache memory from the database 324 uses a cache 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 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 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 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 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 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 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:

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 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 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.

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 memory element flushed.

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 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.

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.

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 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, ~~comprising~~ includes 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. ~~Which~~ The 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.

PRINT OF DRAWINGS
AS ORIGINALLY FILED

1/21

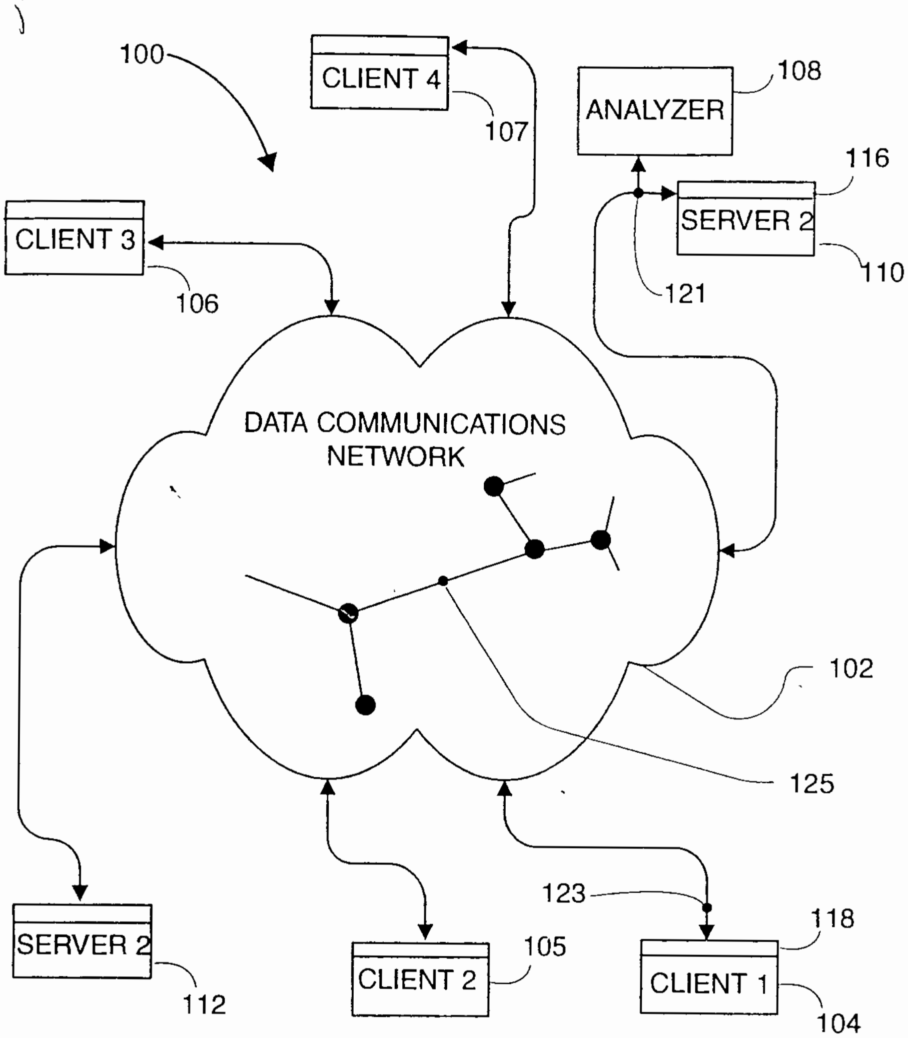


FIG. 1

PRINT OF DRAWINGS AS ORIGINALLY [F]ILED

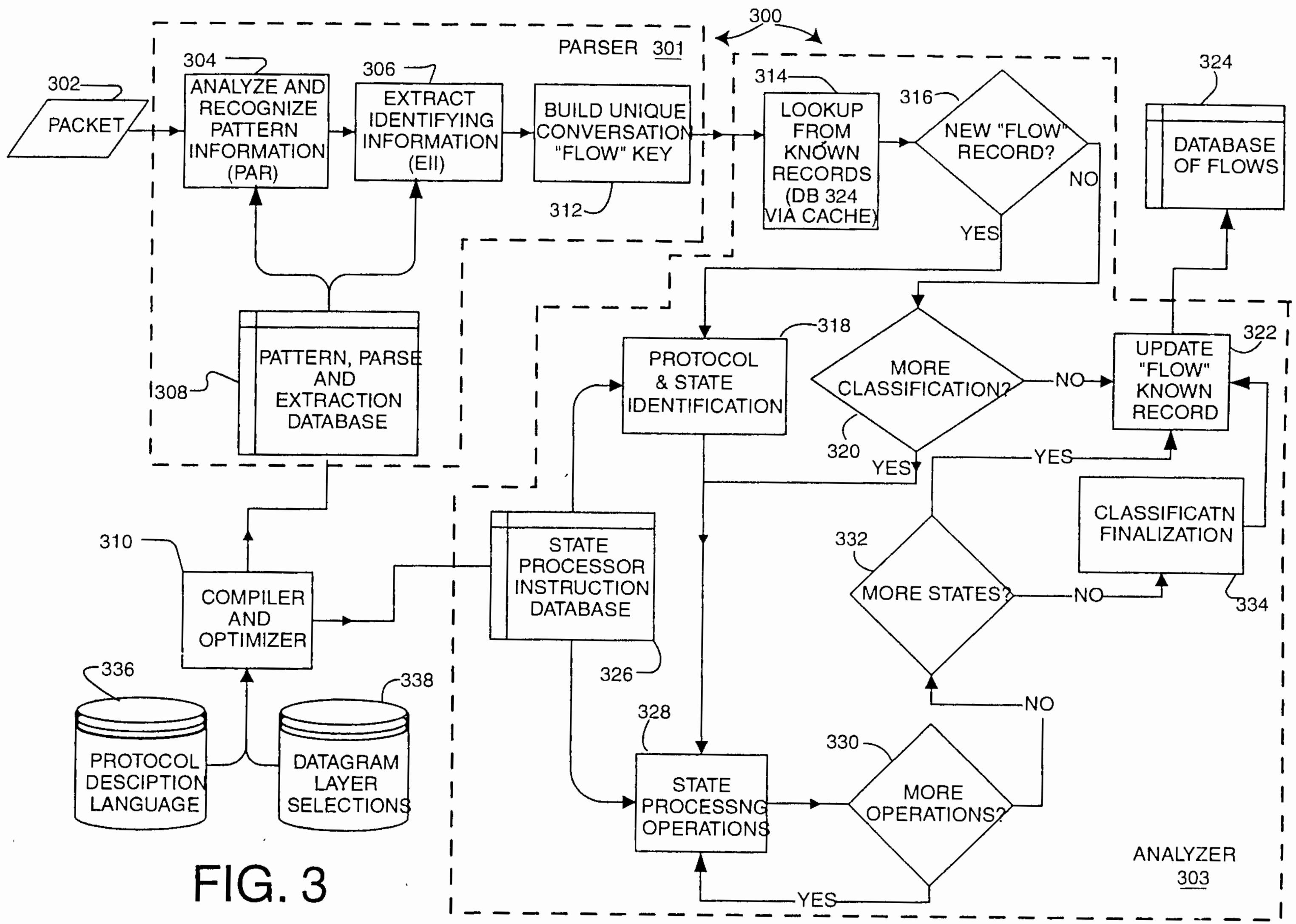


FIG. 3

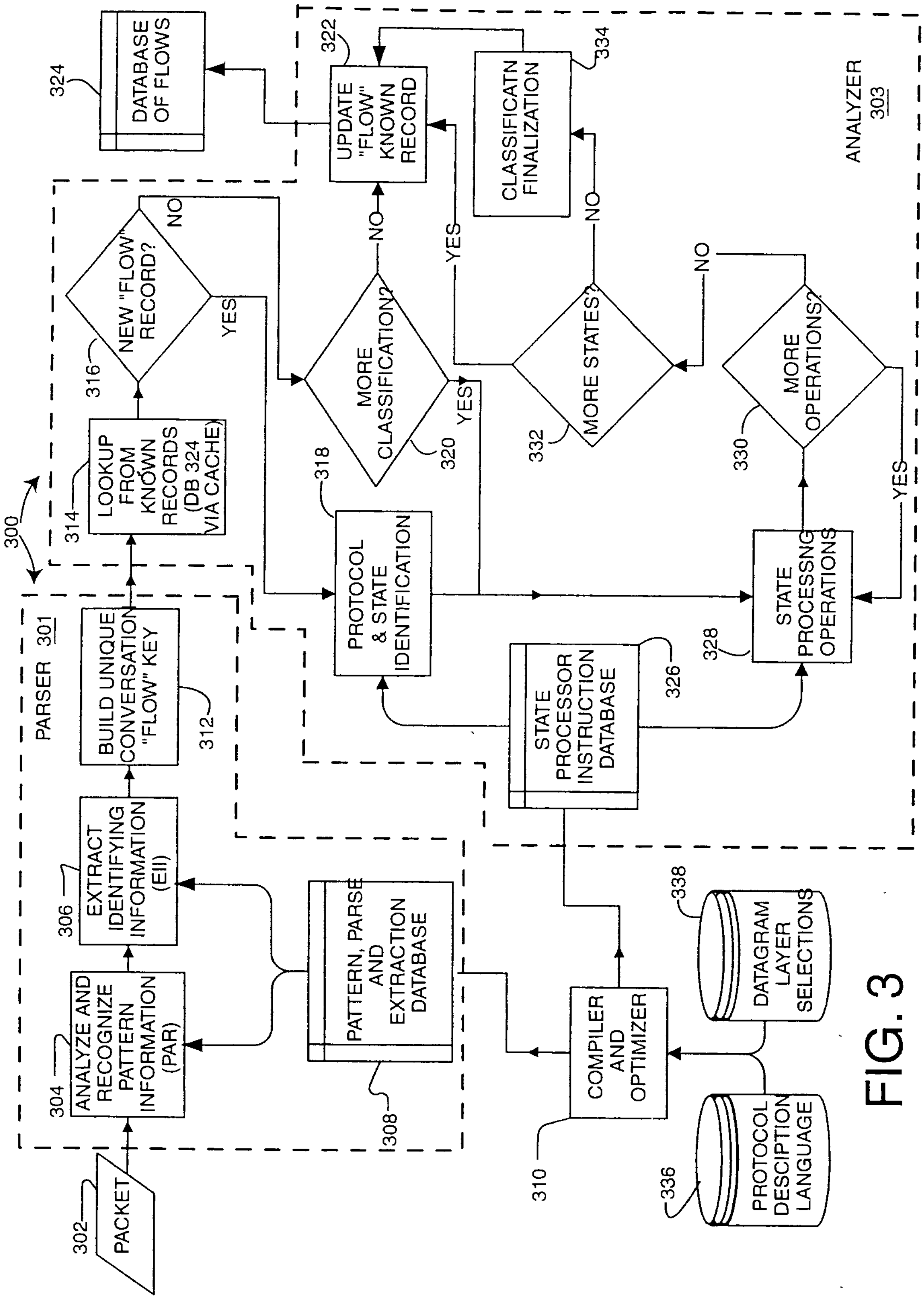

300
PARSER 301
PACKET
302
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
304
EXTRACT IDENTIFYING INFORMATION (EII)
306
PATTERN, PARSE AND EXTRACTION DATABASE
308
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
314
NEW "FLOW" RECORD?
316
YES
NO
DATABASE OF FLOWS
324
ANALYZER 303
PROTOCOL & STATE IDENTIFICATION
318
MORE CLASSIFICATION?
320
YES
NO
UPDATE "FLOW" KNOWN RECORD
322
CLASSIFICATN FINALIZATION
334
MORE STATES?
332
YES
NO
STATE PROCESSOR INSTRUCTION DATABASE
326
STATE PROCESSNG OPERATIONS
328
MORE OPERATIONS?
330
YES
NO
COMPILER AND OPTIMIZER
310
PROTOCOL DESCIPTION LANGUAGE
336
DATAGRAM LAYER SELECTIONS
338

FIG. 3

PRINT OF DRAWINGS
AS ORIGINALLY FILED

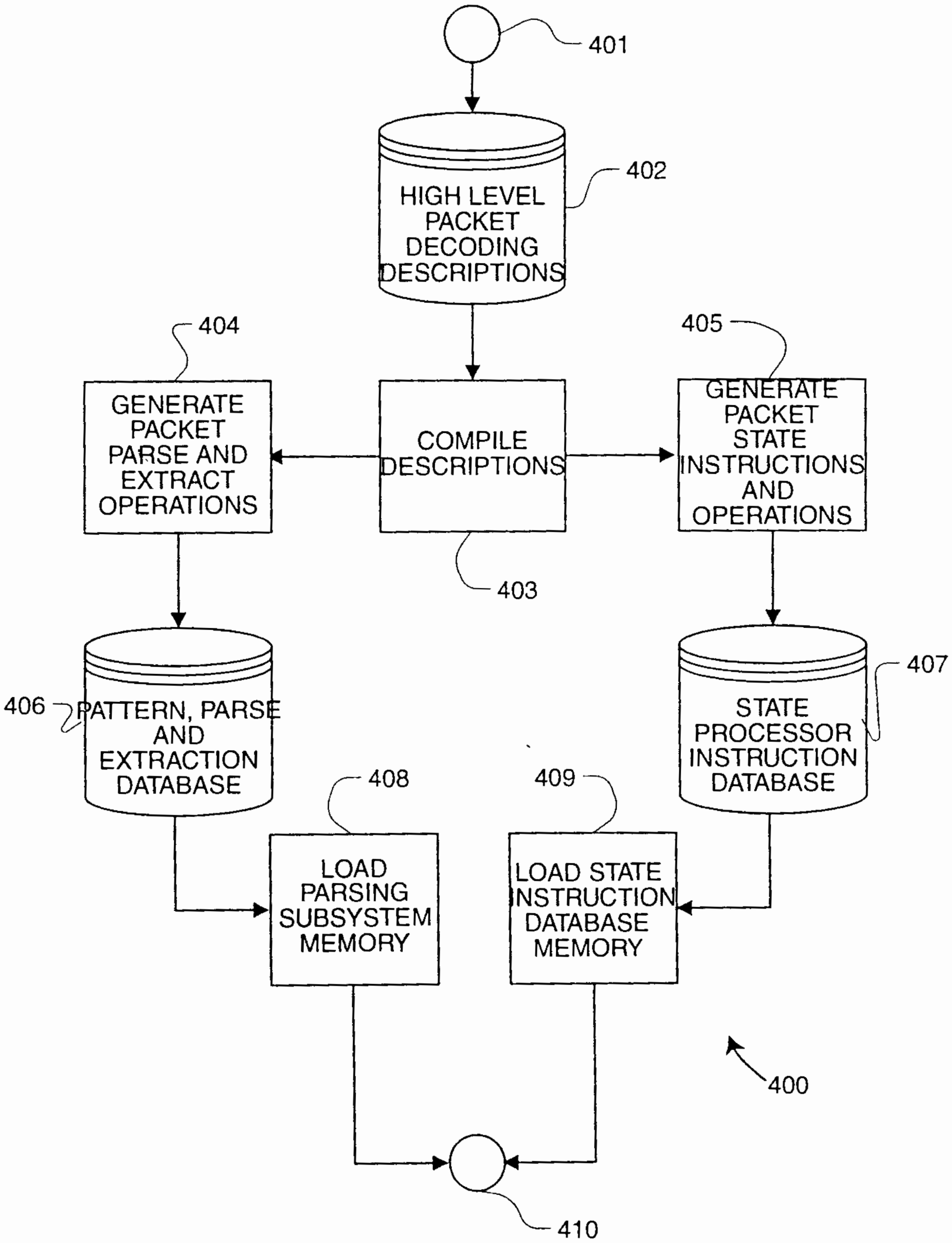


FIG. 4

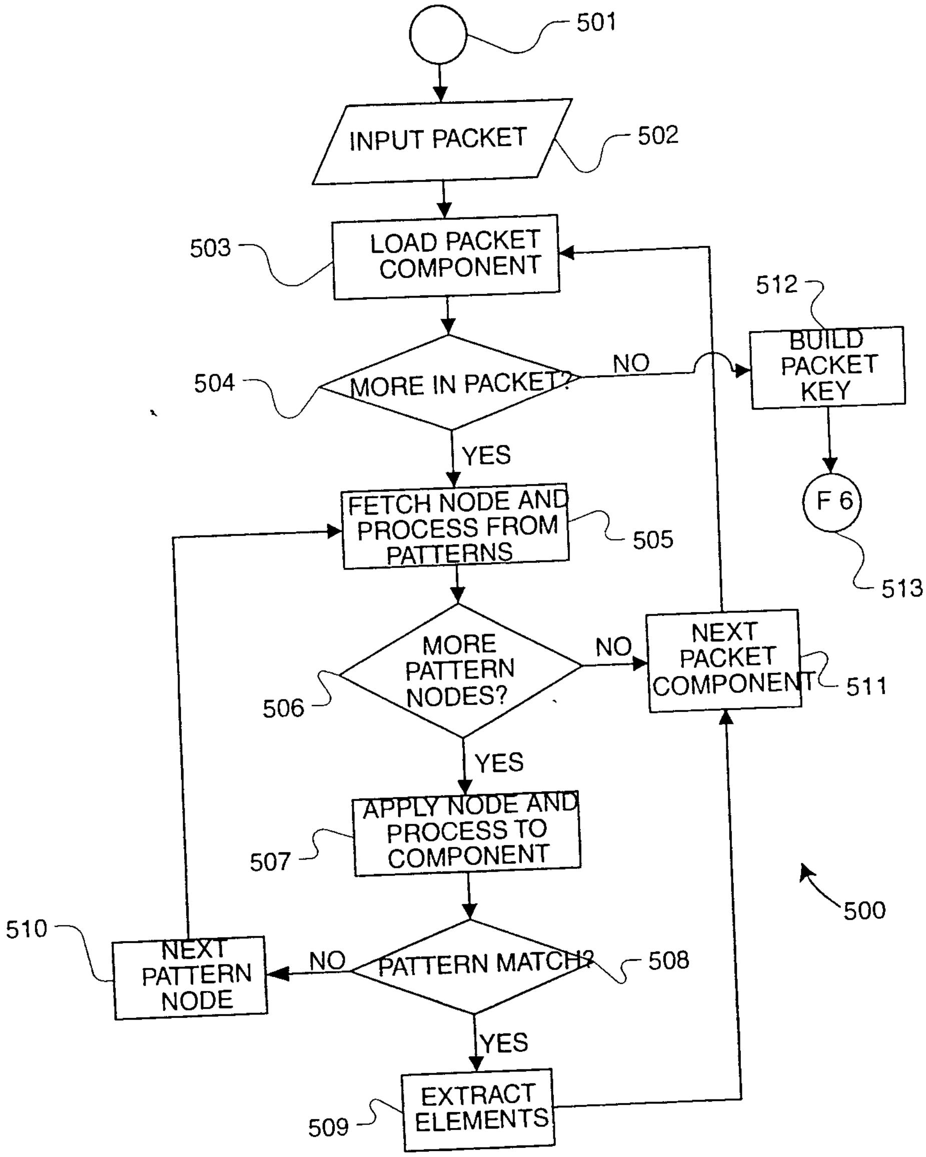

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
512
BUILD PACKET KEY
504
MORE IN PACKET?
NO
YES
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
NO
NEXT PACKET COMPONENT
511
506
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT ELEMENTS
509

FIG. 5

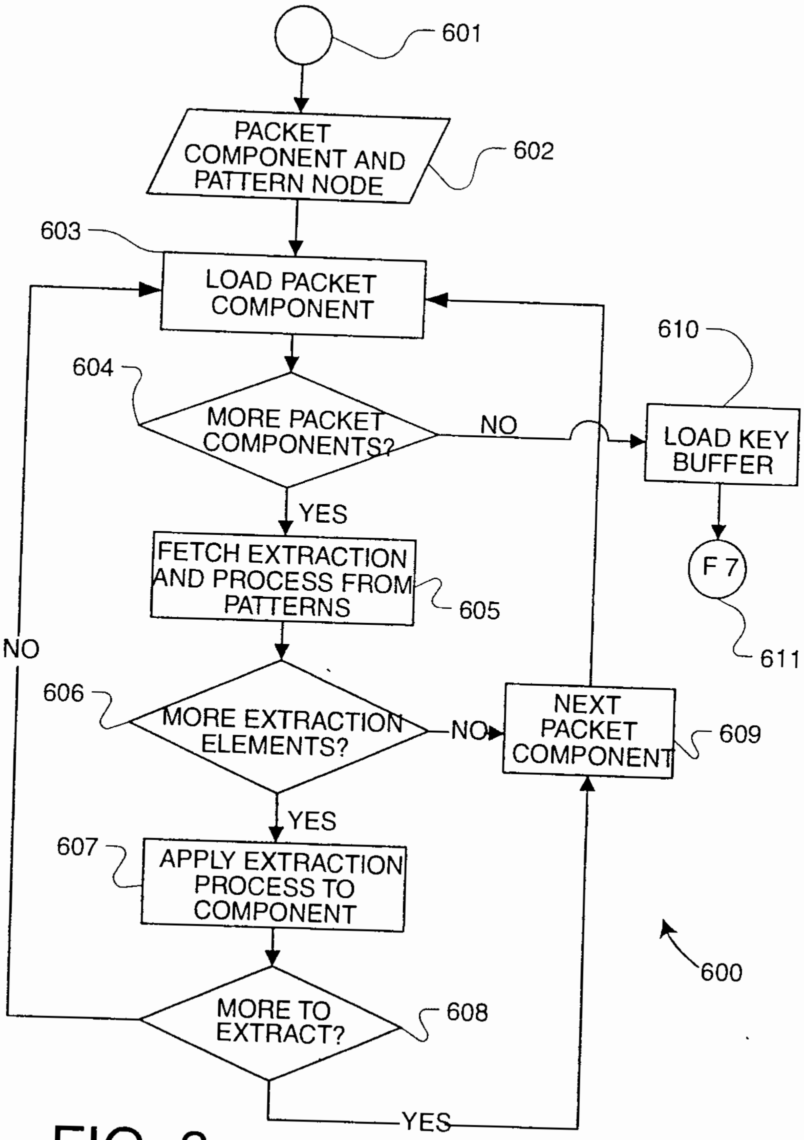

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

PRINT OF DRAWINGS
AS ORIGINALLY ED

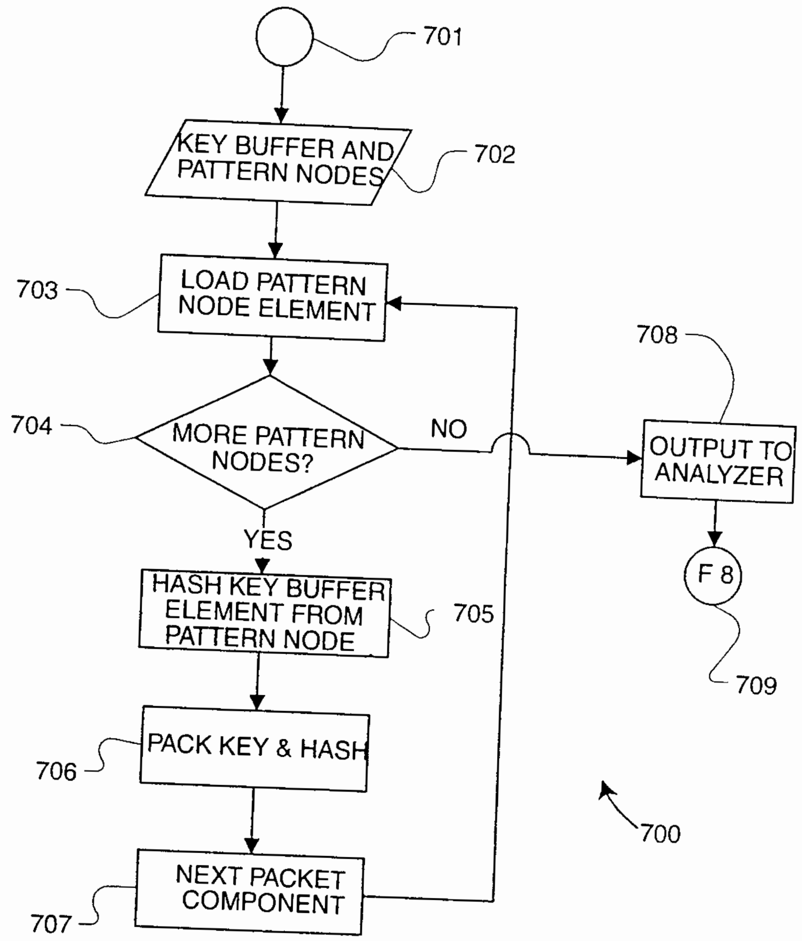


FIG. 7

PRINT OF DRAWINGS
AS ORIGINALLY FILED

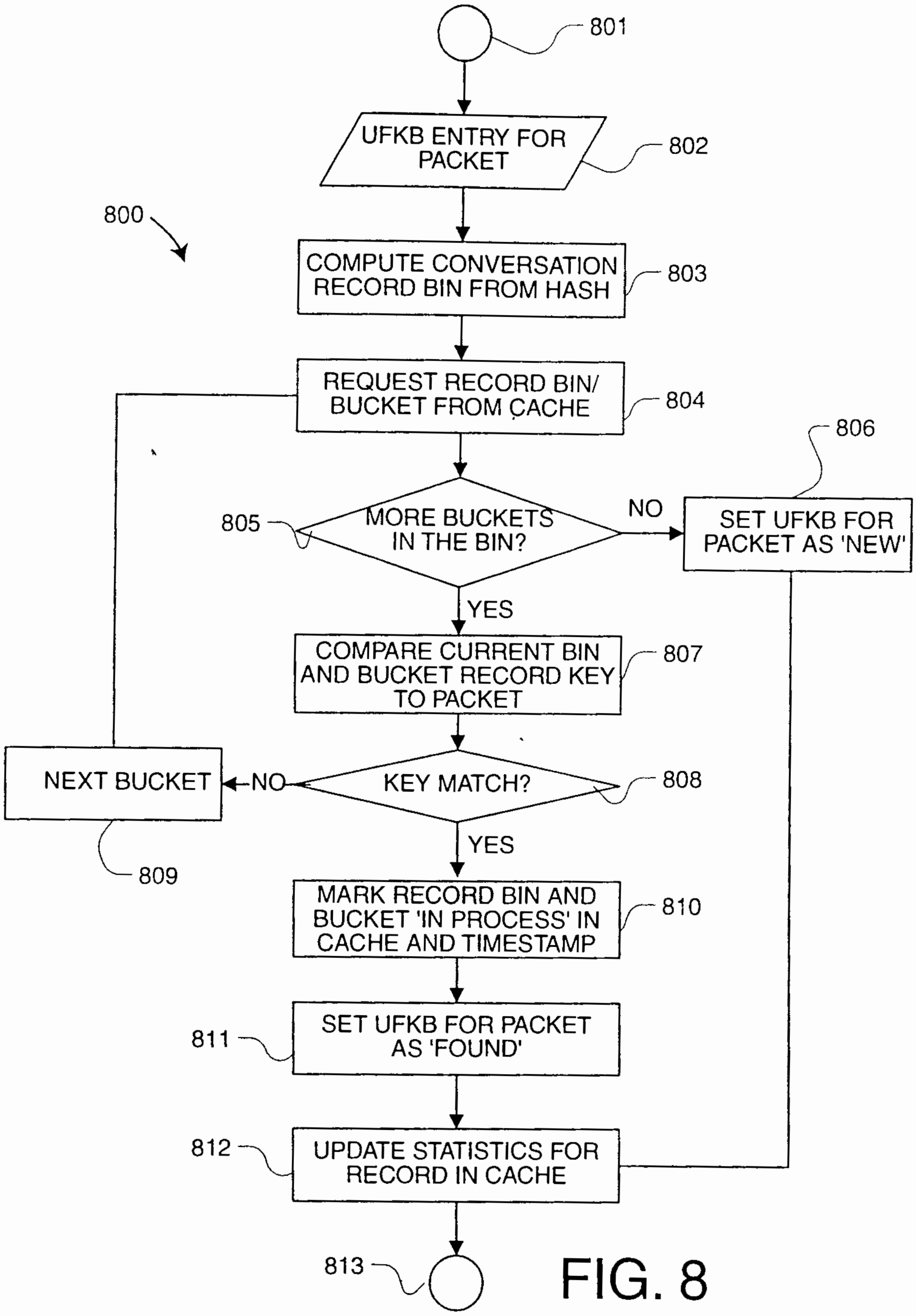


FIG. 8

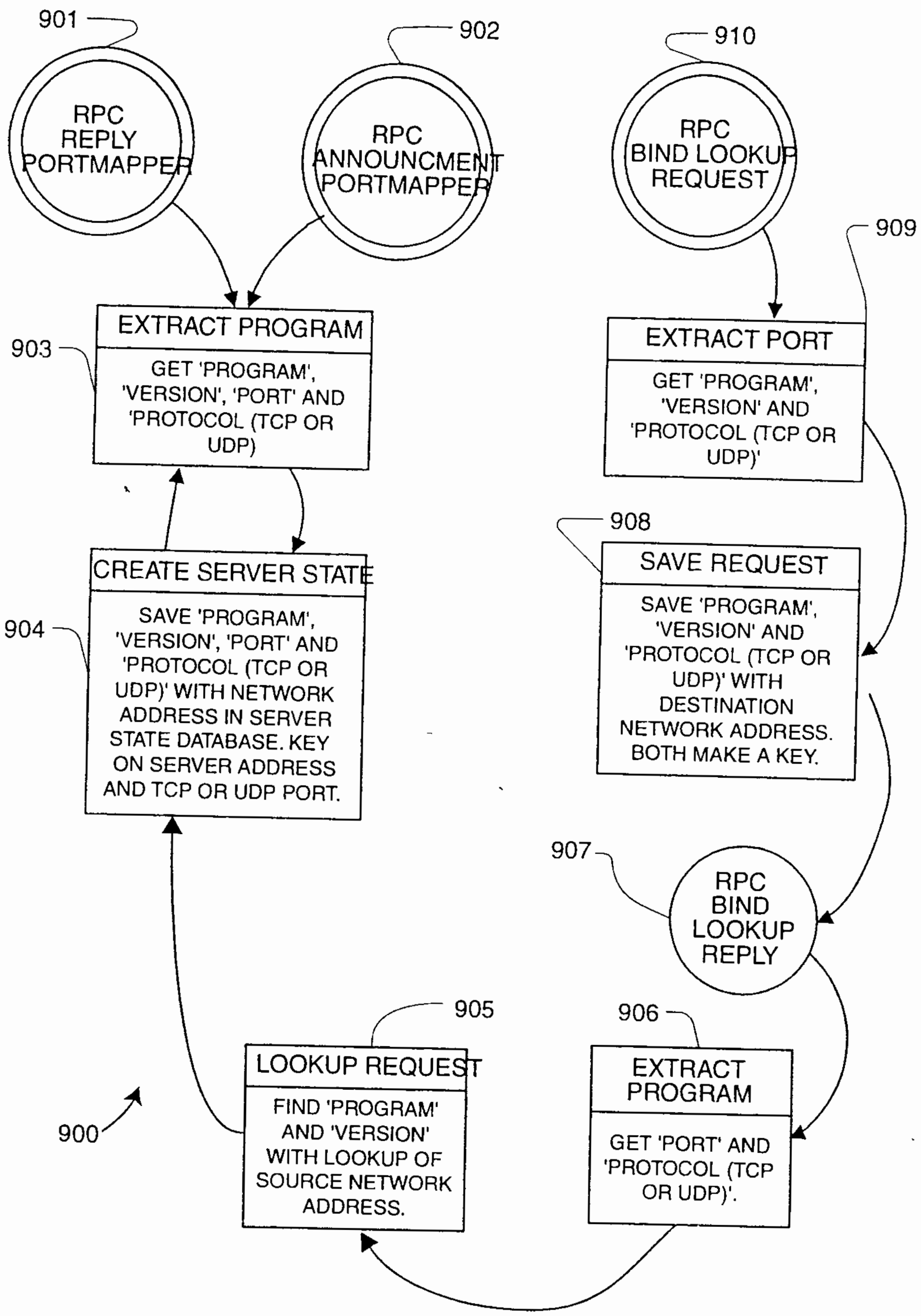

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
906
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

Sarkissi...

PRINT OF DRAWINGS
AS ORIGINALLY ...ED

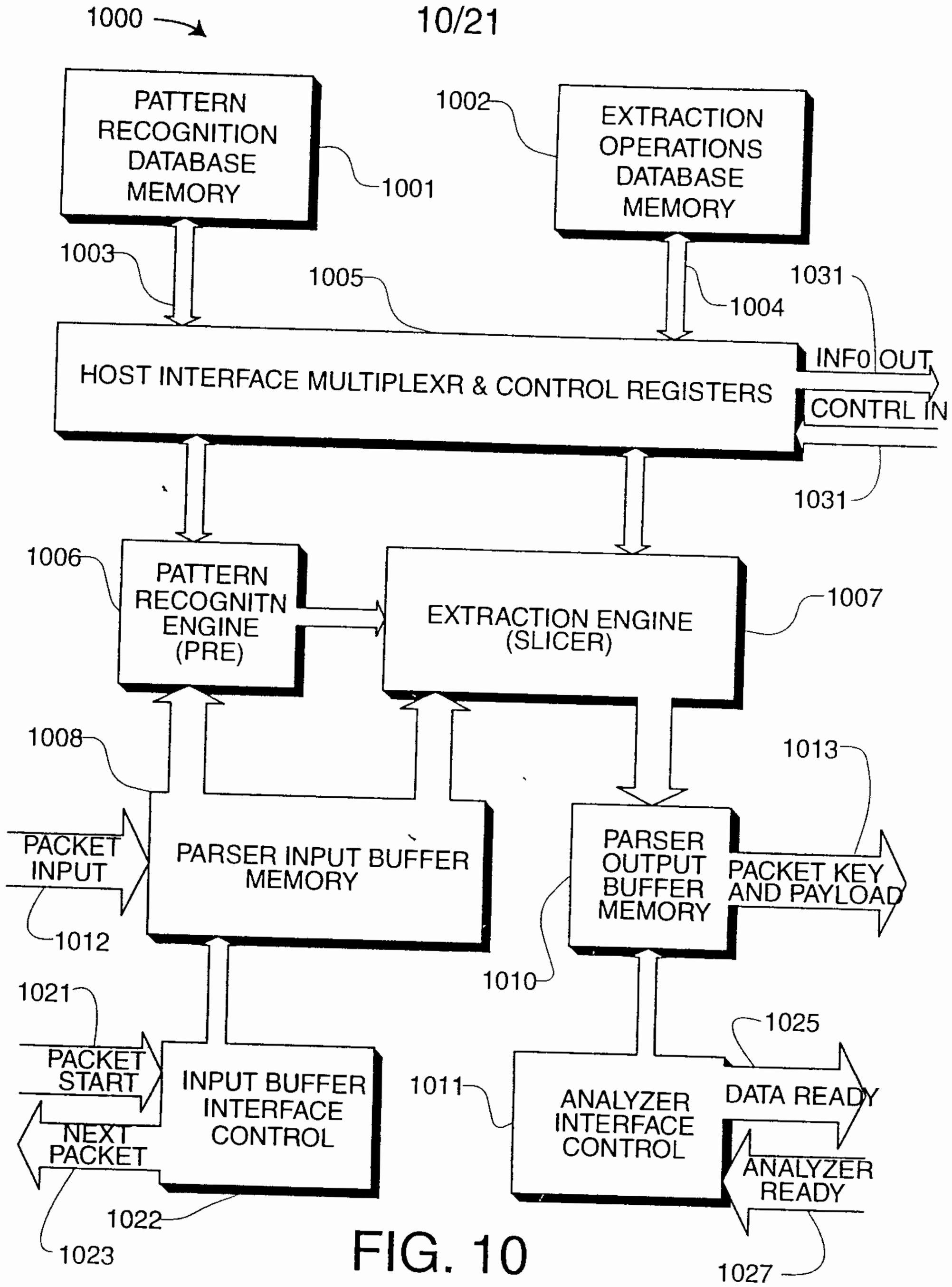


FIG. 10

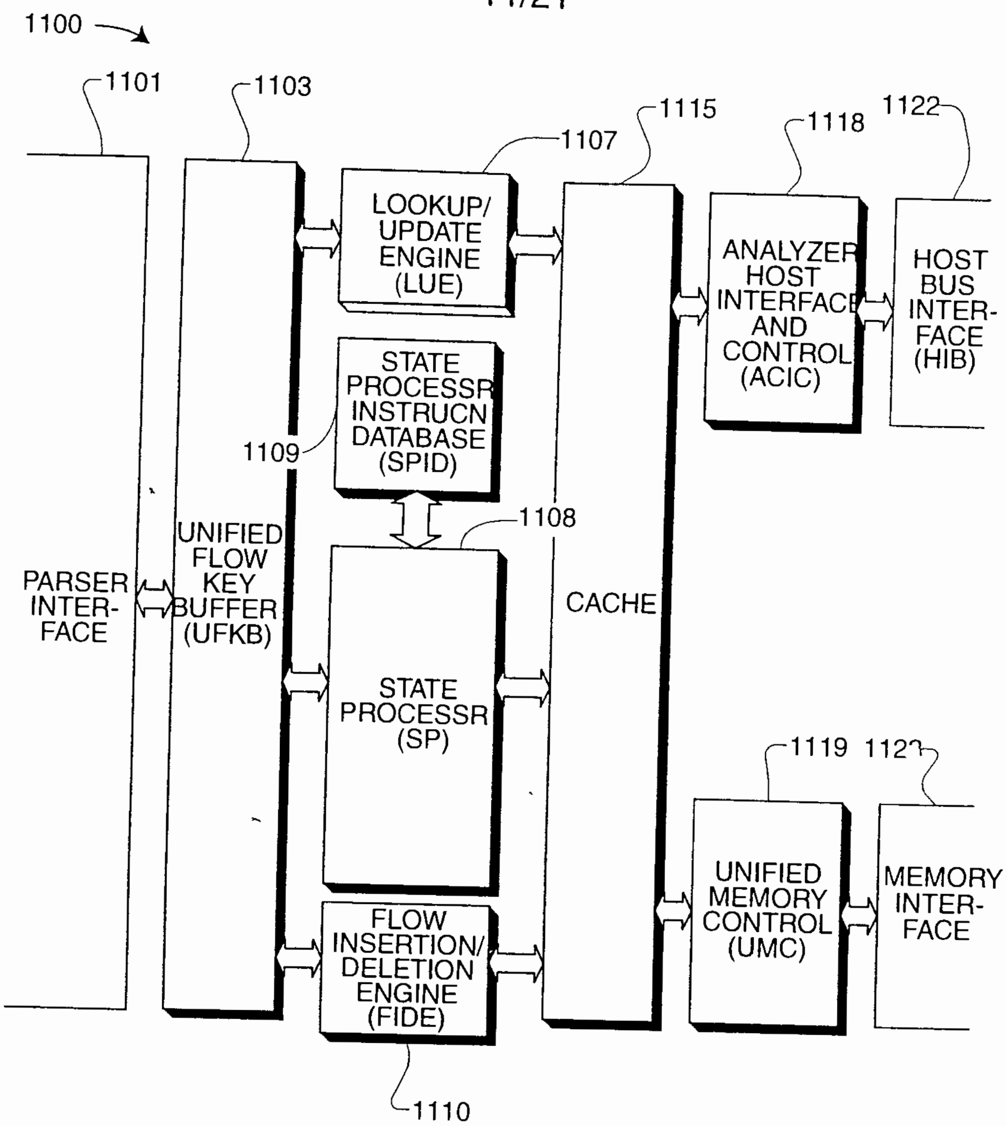

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
UNIFIED FLOW KEY BUFFER (UFKB)
PARSER INTER-FACE
CACHE
STATE PROCESSR (SP)
1119
UNIFIED MEMORY CONTROL (UMC)
MEMORY INTER-FACE
FLOW INSERTION/ DELETION ENGINE (FIDE)
1110

FIG. 11

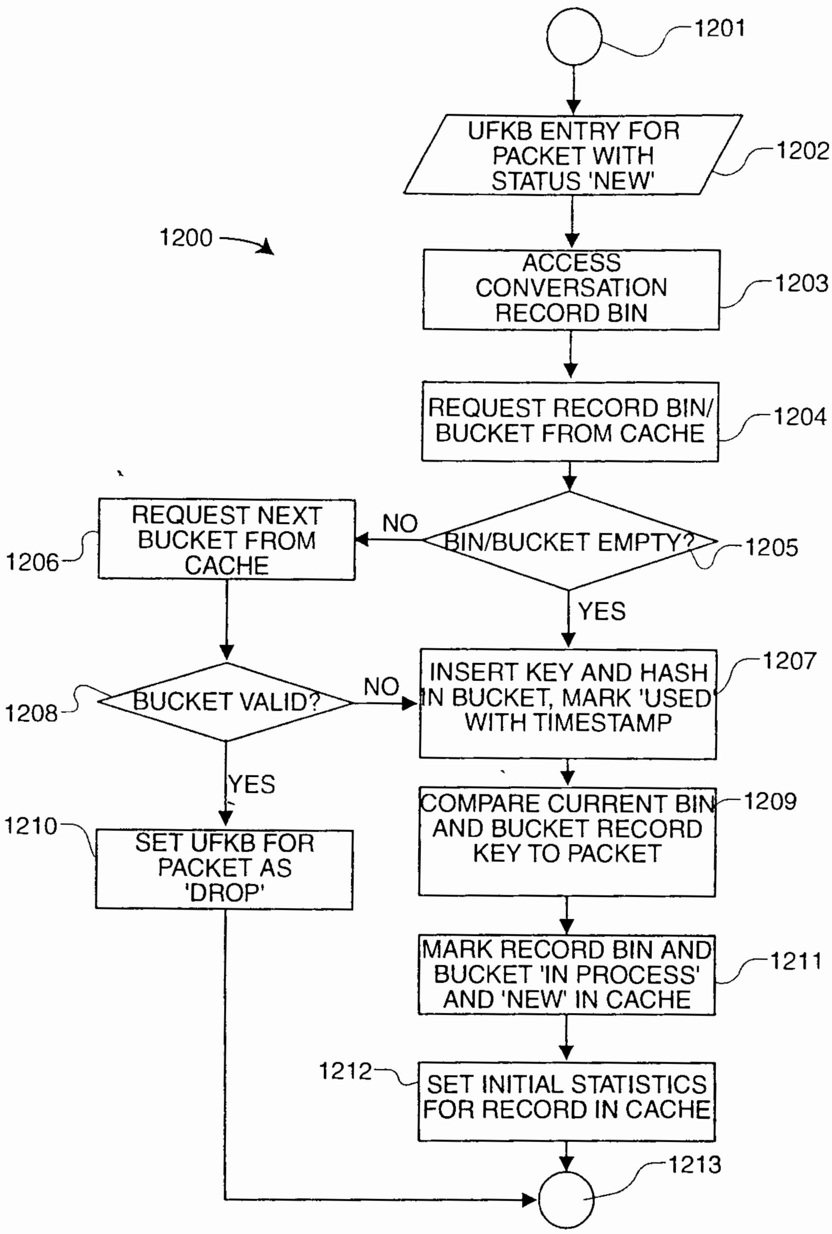

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
REQUEST NEXT BUCKET FROM CACHE
1206
NO
BIN/BUCKET EMPTY?
1205
YES
BUCKET VALID?
1208
NO
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
YES
1210
SET UFKB FOR PACKET AS 'DROP'
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
1212
SET INITIAL STATISTICS FOR RECORD IN CACHE
1213

FIG. 12

Sarki

PRINT OF DRAWINGS
AS ORIGINALLY FILED

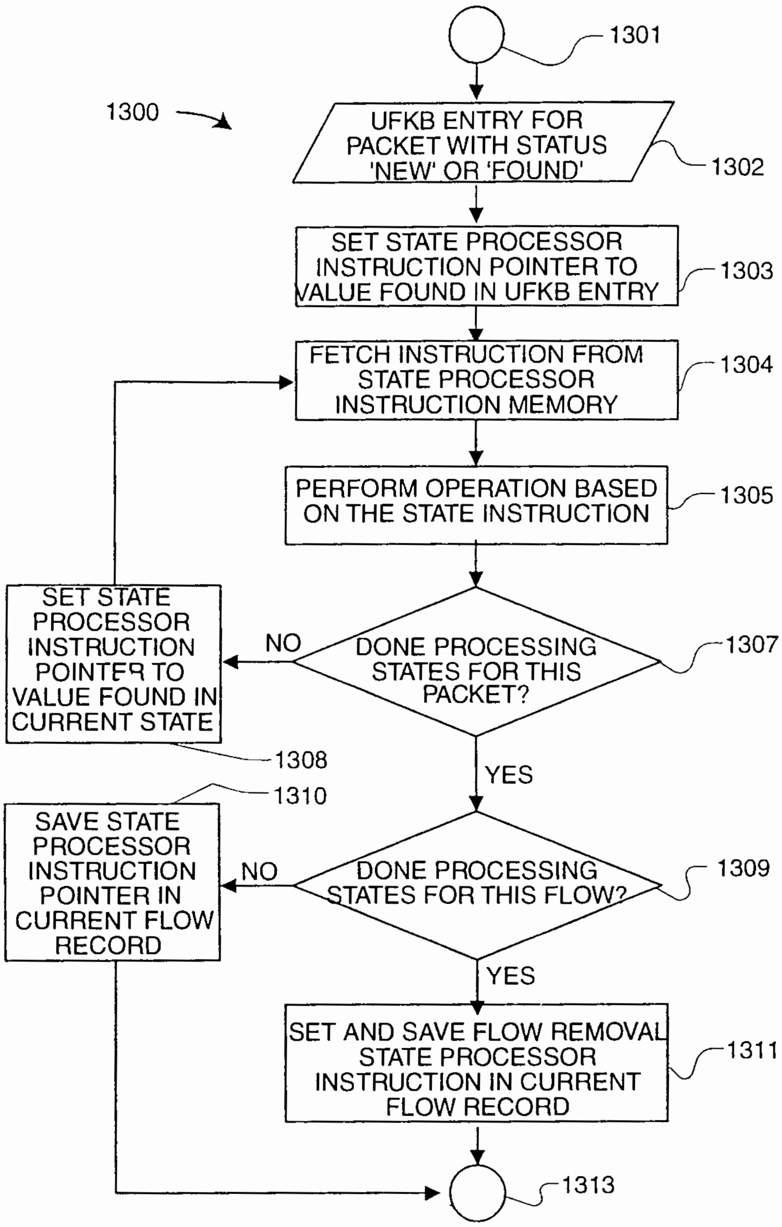


FIG. 13

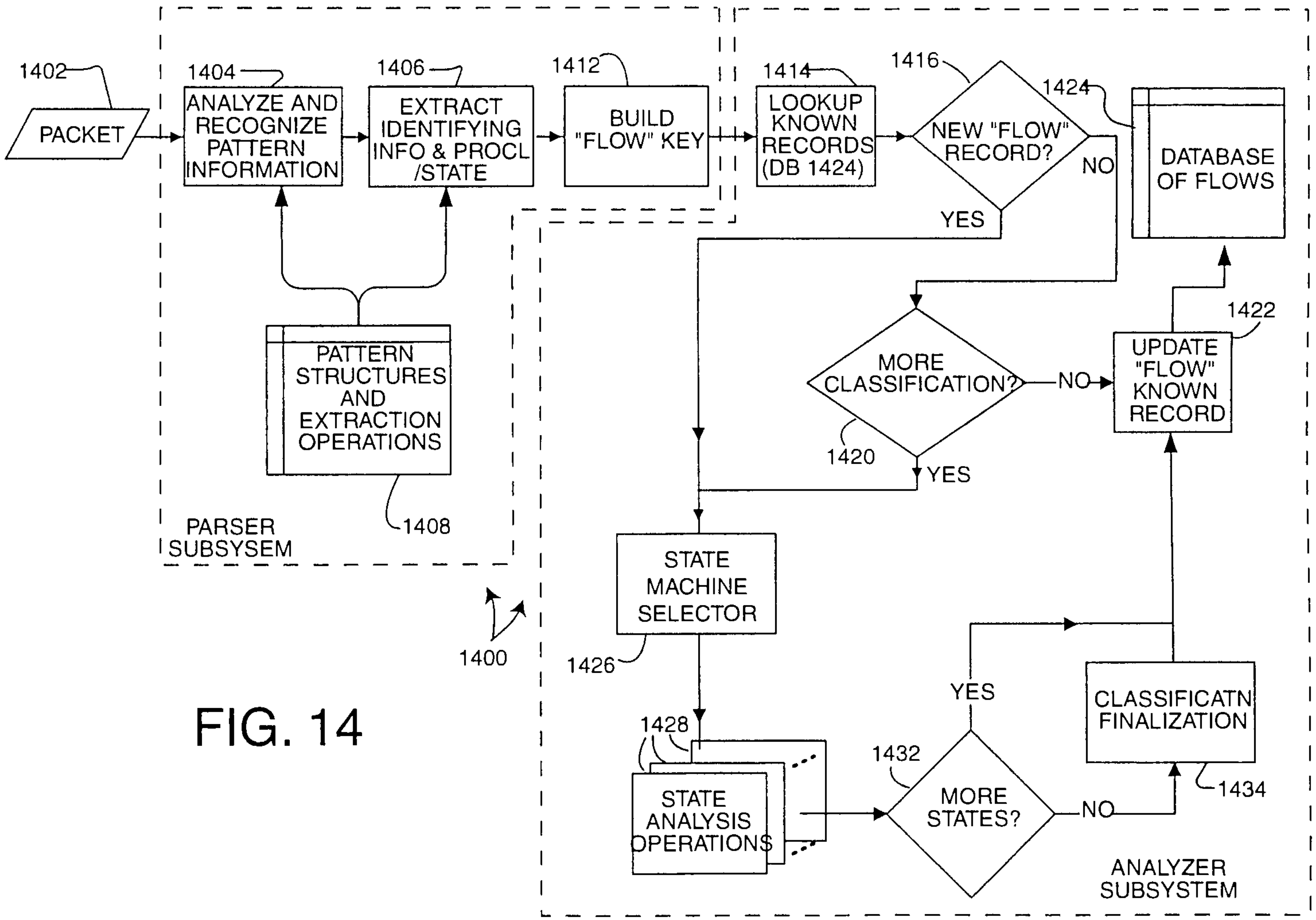


FIG. 14

14/21

PRINT OF DRAWINGS AS ORIGINALLY FILED

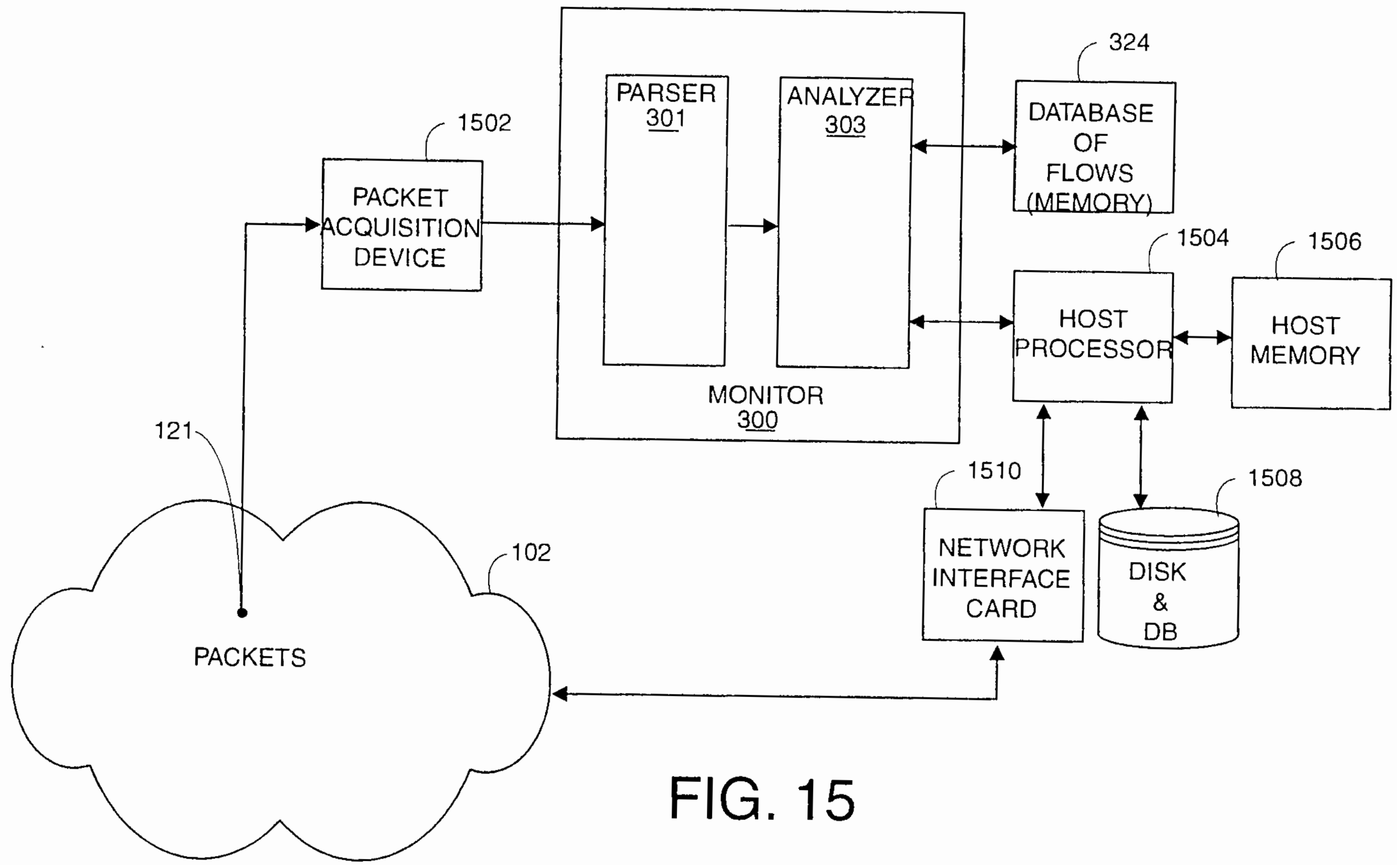

324
DATABASE
OF
FLOWS
(MEMORY)
1502
PACKET
ACQUISITION
DEVICE
PARSER
301
ANALYZER
303
MONITOR
300
1504
HOST
PROCESSOR
1506
HOST
MEMORY
1510
NETWORK
INTERFACE
CARD
1508
DISK
&
DB
121
102
PACKETS

FIG. 15

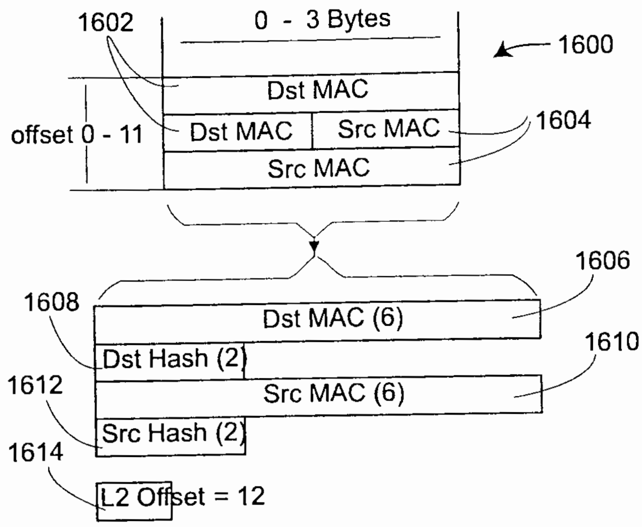

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

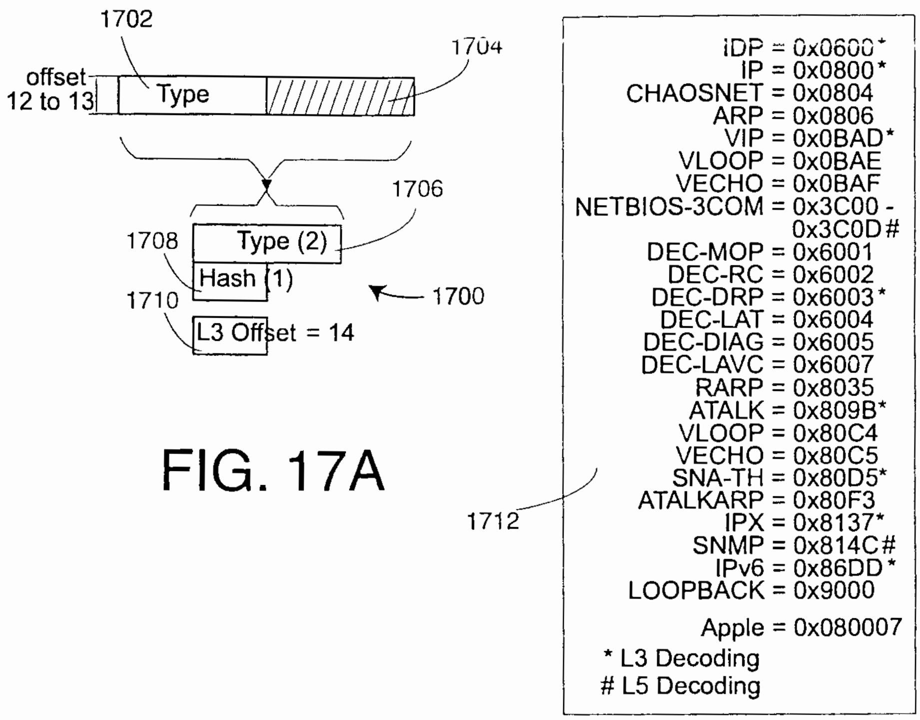


FIG. 17A

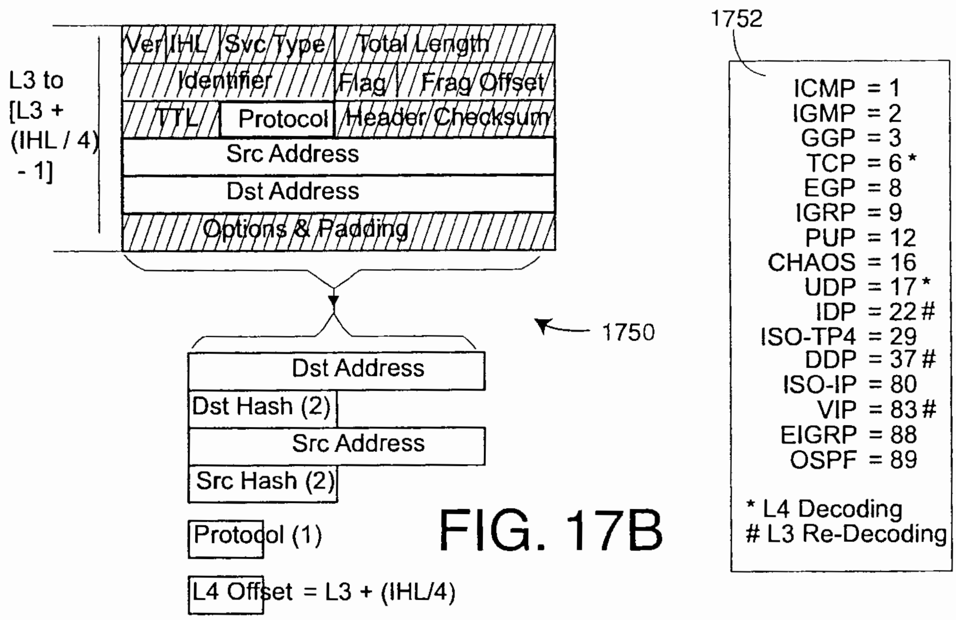


FIG. 17B

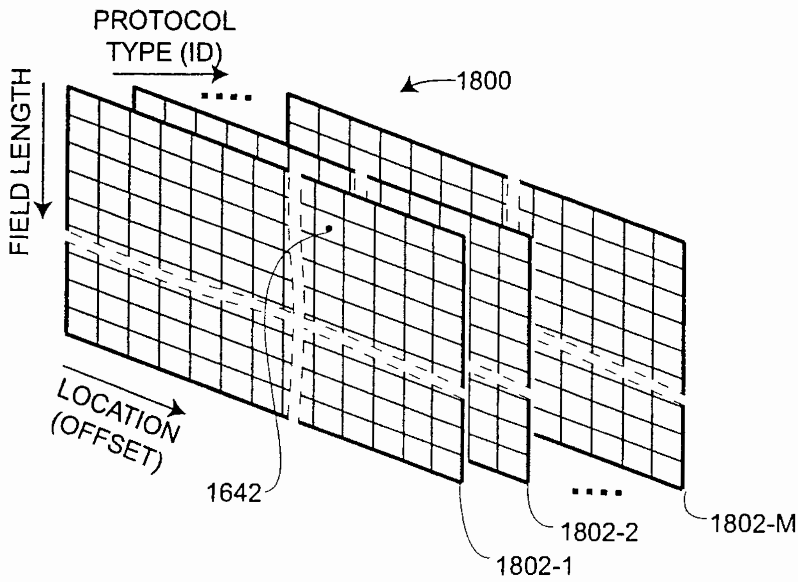


FIG. 18A

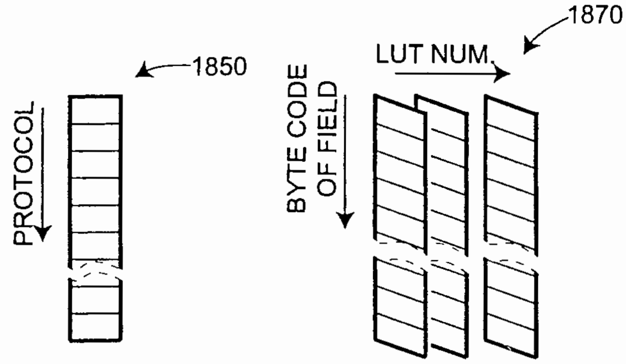


FIG. 18B

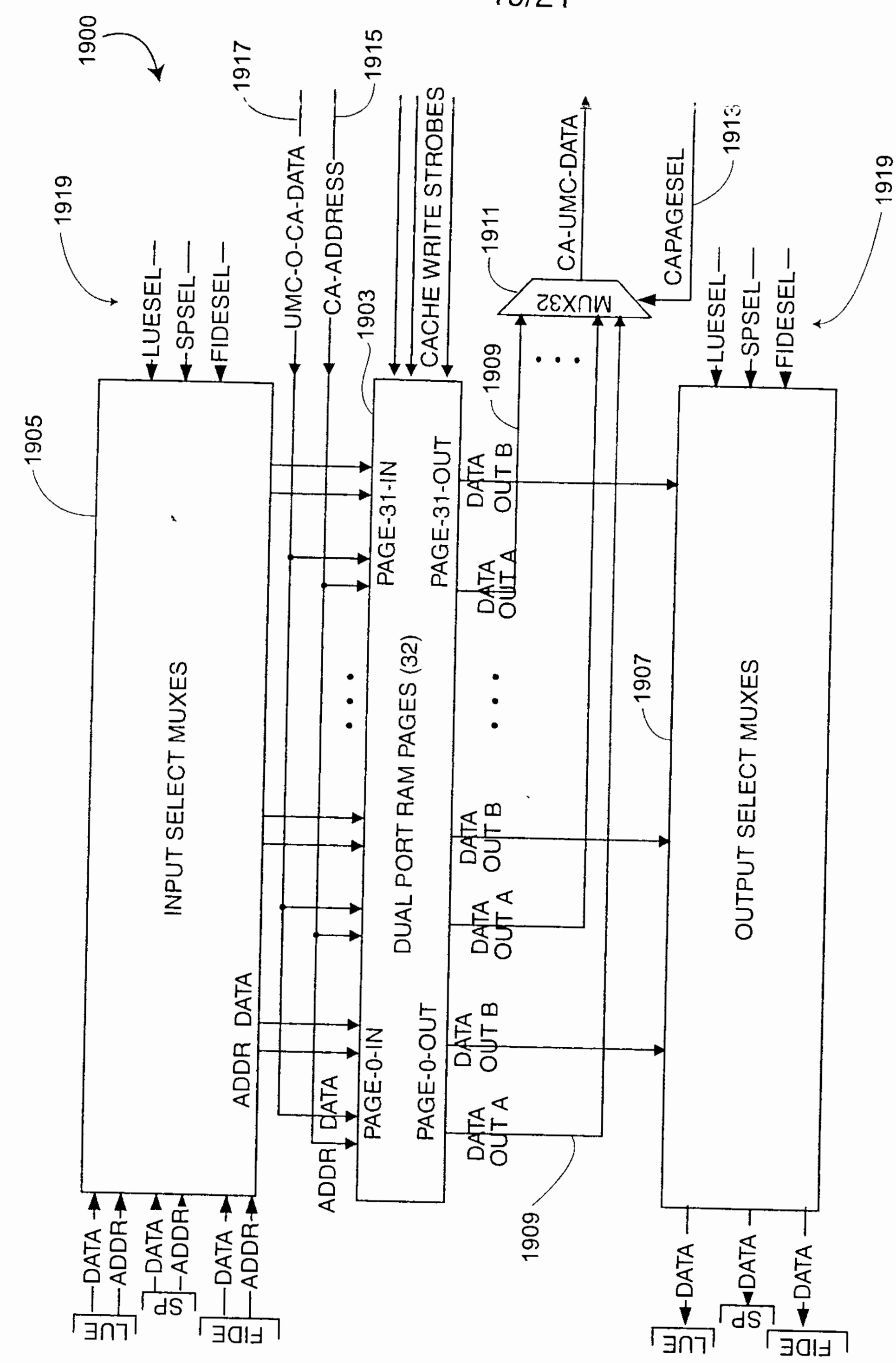

1900
1919
LUESEL
SPSEL
FIDESEL
1917
UMC-O-CA-DATA
1915
CA-ADDRESS
CACHE WRITE STROBES
1903
1911
CA-UMC-DATA
MUX32
1913
CAPAGESEL
1919
LUESEL
SPSEL
FIDESEL
1905
INPUT SELECT MUXES
ADDR DATA
PAGE-31-IN
PAGE-31-OUT
DUAL PORT RAM PAGES (32)
PAGE-0-IN
PAGE-0-OUT
ADDR DATA
1909
DATA OUT B
DATA OUT A
1907
OUTPUT SELECT MUXES
1909
DATA
ADDR
LUE
SP
FIDE

FIG. 19

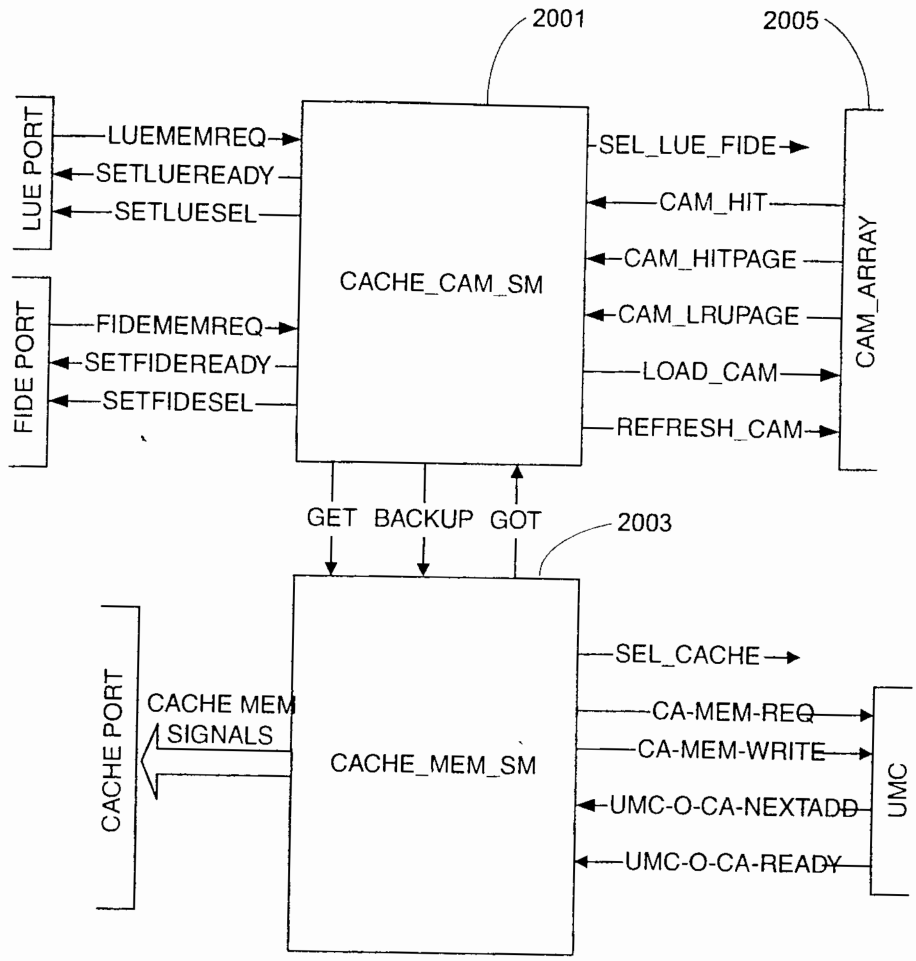

2001
2005
LUE PORT
LUEMEMREQ
SETLUEREADY
SETLUESEL
FIDE PORT
FIDEMEMREQ
SETFIDEREADY
SETFIDESEL
CACHE_CAM_SM
SEL_LUE_FIDE
CAM_HIT
CAM_HITPAGE
CAM_LRUPAGE
LOAD_CAM
REFRESH_CAM
CAM_ARRAY
GET
BACKUP
GOT
2003
CACHE PORT
CACHE MEM
SIGNALS
CACHE_MEM_SM
SEL_CACHE
CA-MEM-REQ
CA-MEM-WRITE
UMC-O-CA-NEXTADD
UMC-O-CA-READY
UMC

FIG. 20

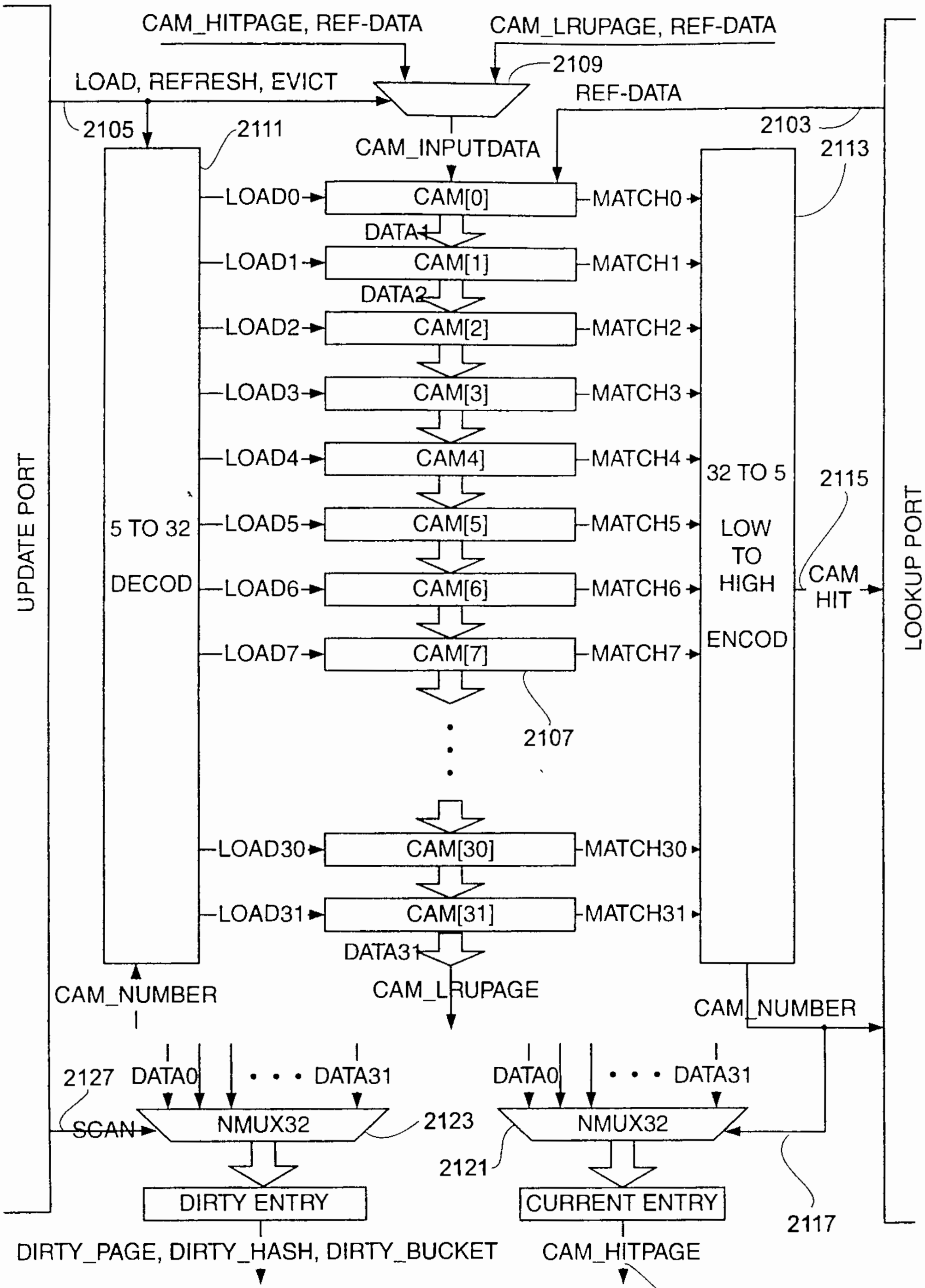

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
2109
LOAD, REFRESH, EVICT
REF-DATA
2105
2111
CAM_INPUTDATA
2103
2113
UPDATE PORT
5 TO 32
DECOD
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
LOAD30
LOAD31
CAM[0]
DATA1
CAM[1]
DATA2
CAM[2]
CAM[3]
CAM4]
CAM[5]
CAM[6]
CAM[7]
CAM[30]
CAM[31]
DATA31
MATCH0
MATCH1
MATCH2
MATCH3
MATCH4
MATCH5
MATCH6
MATCH7
MATCH30
MATCH31
32 TO 5
LOW
TO
HIGH
ENCOD
2115
CAM
HIT
LOOKUP PORT
2107
CAM_NUMBER
CAM_LRUPAGE
CAM_NUMBER
2127
DATA0
DATA31
DATA0
DATA31
SCAN
NMUX32
2123
NMUX32
2121
DIRTY ENTRY
CURRENT ENTRY
2117
DIRTY_PAGE, DIRTY_HASH, DIRTY_BUCKET
CAM_HITPAGE
2119

FIG. 21

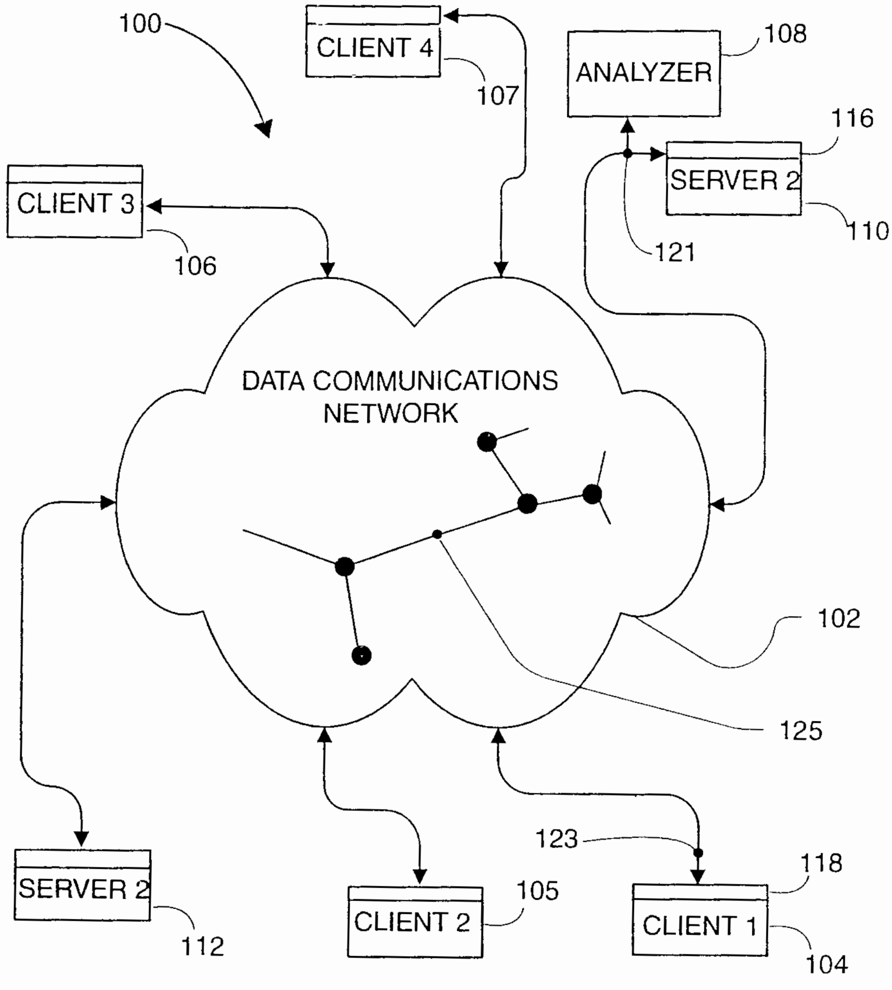

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS
NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

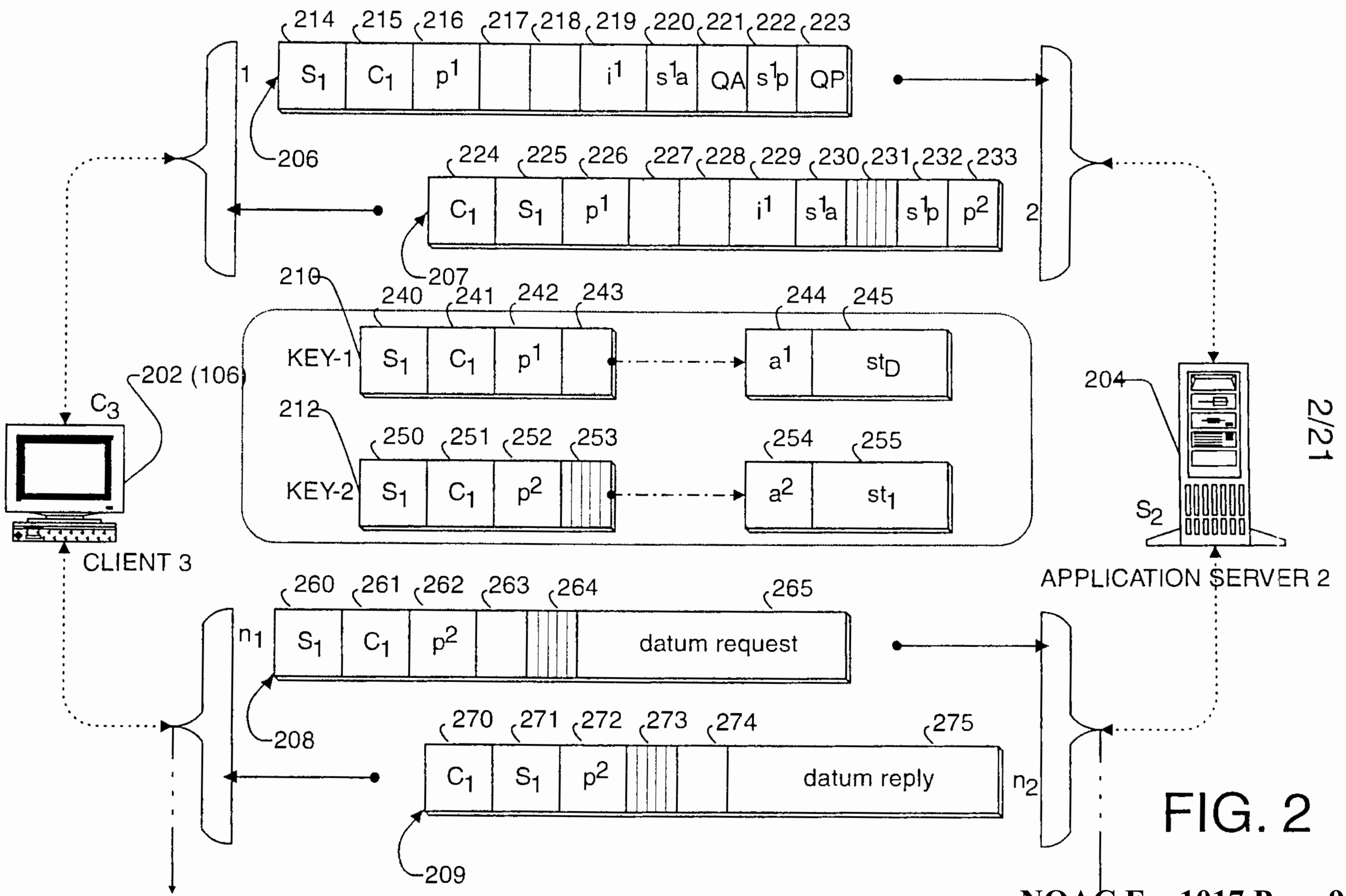

214 215 216 217 218 219 220 221 222 223
S1 C1 p1 i1 s1a QA s1p QP
1
206
224 225 226 227 228 229 230 231 232 233
C1 S1 p1 i1 s1a s1p p2
2
207
210
240 241 242 243
KEY-1 S1 C1 p1
244 245
a1 stD
212
250 251 252 253
KEY-2 S1 C1 p2
254 255
a2 st1
202 (106)
C3
CLIENT 3
204
S2
APPLICATION SERVER 2
260 261 262 263 264 265
n1 S1 C1 p2 datum request
208
270 271 272 273 274 275
C1 S1 p2 datum reply n2
209

FIG. 2

PRINT OF DRAWINGS AS ORIGINALLY FILED

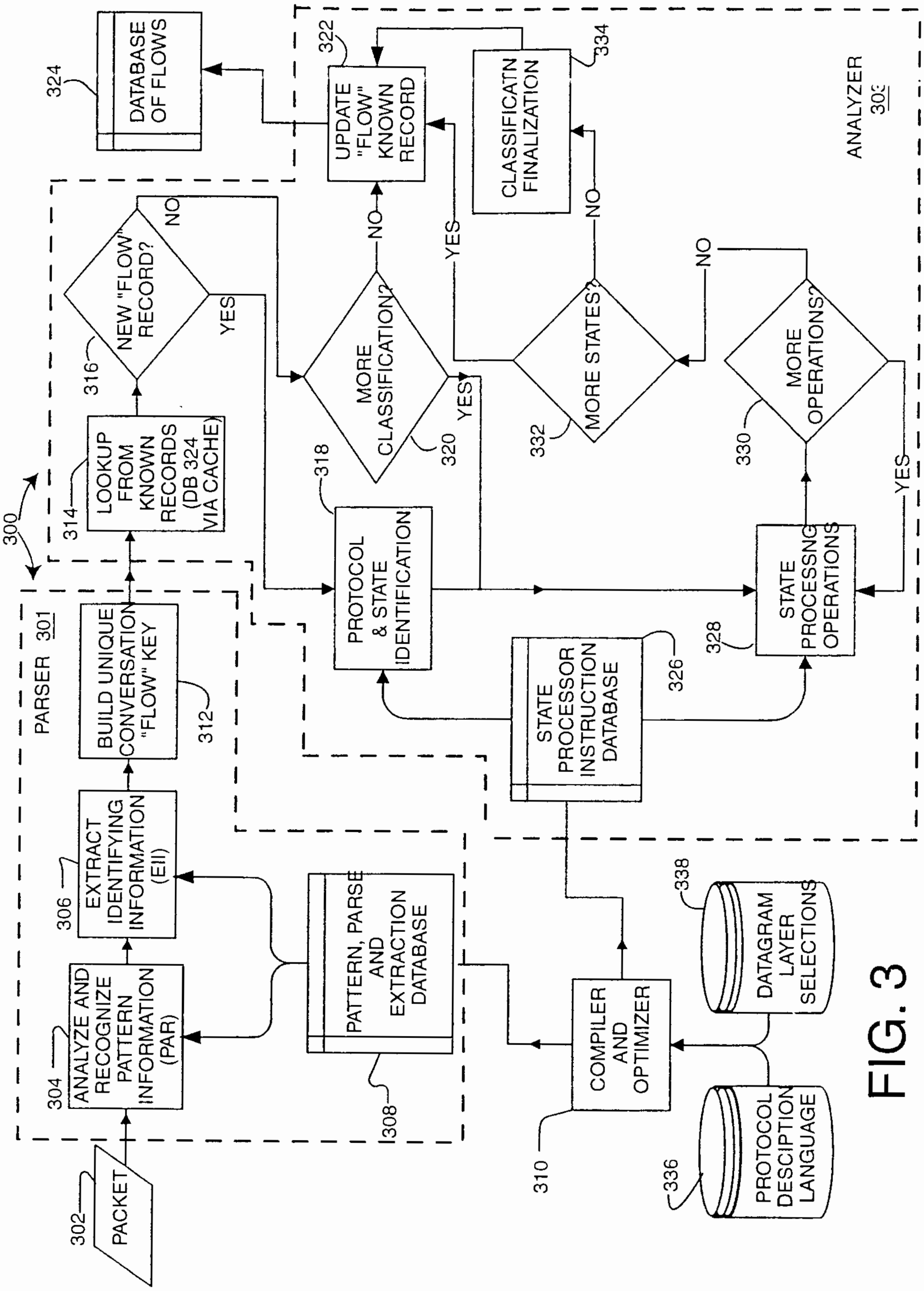


FIG. 3

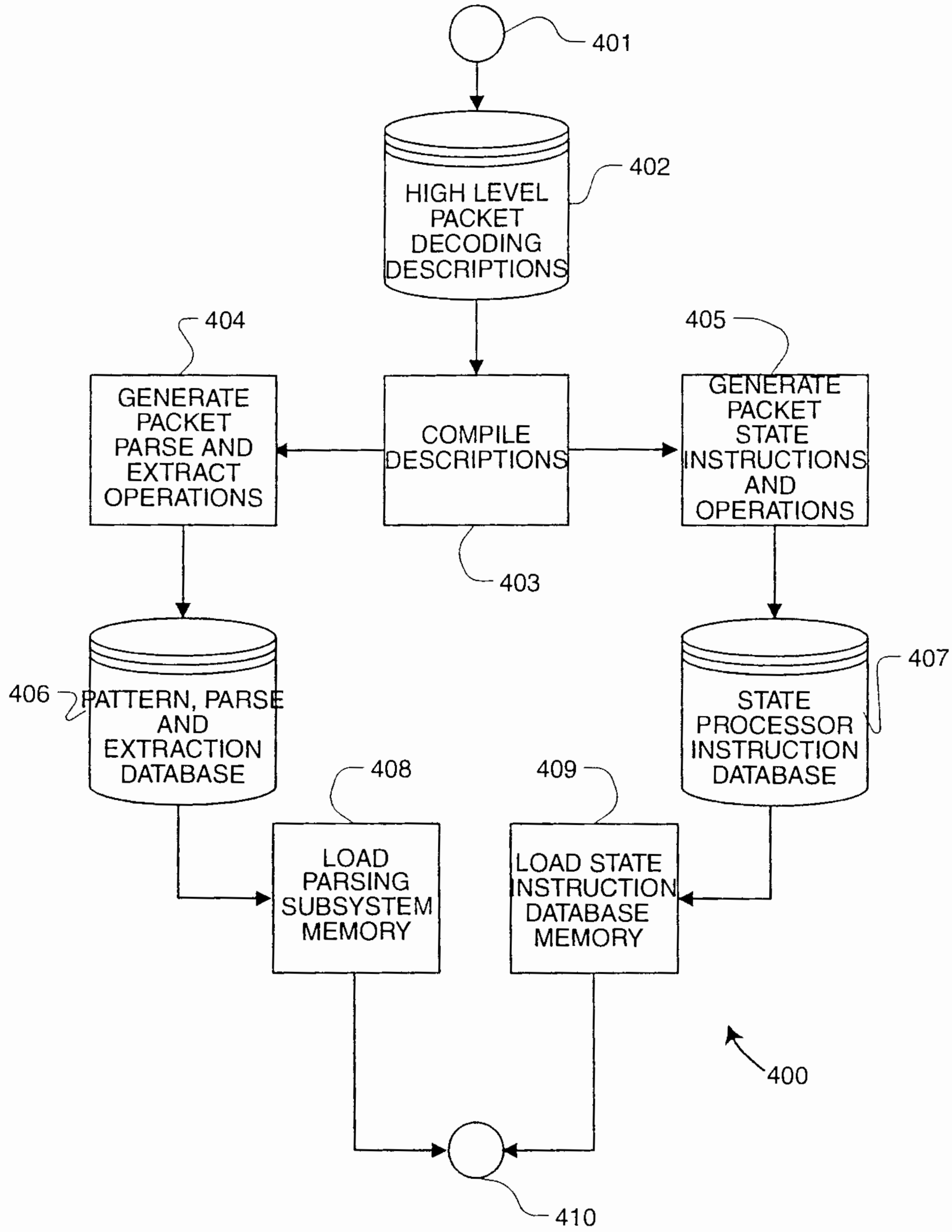

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
405
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
403
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
409
LOAD PARSING SUBSYSTEM MEMORY
LOAD STATE INSTRUCTION DATABASE MEMORY
400
410

FIG. 4

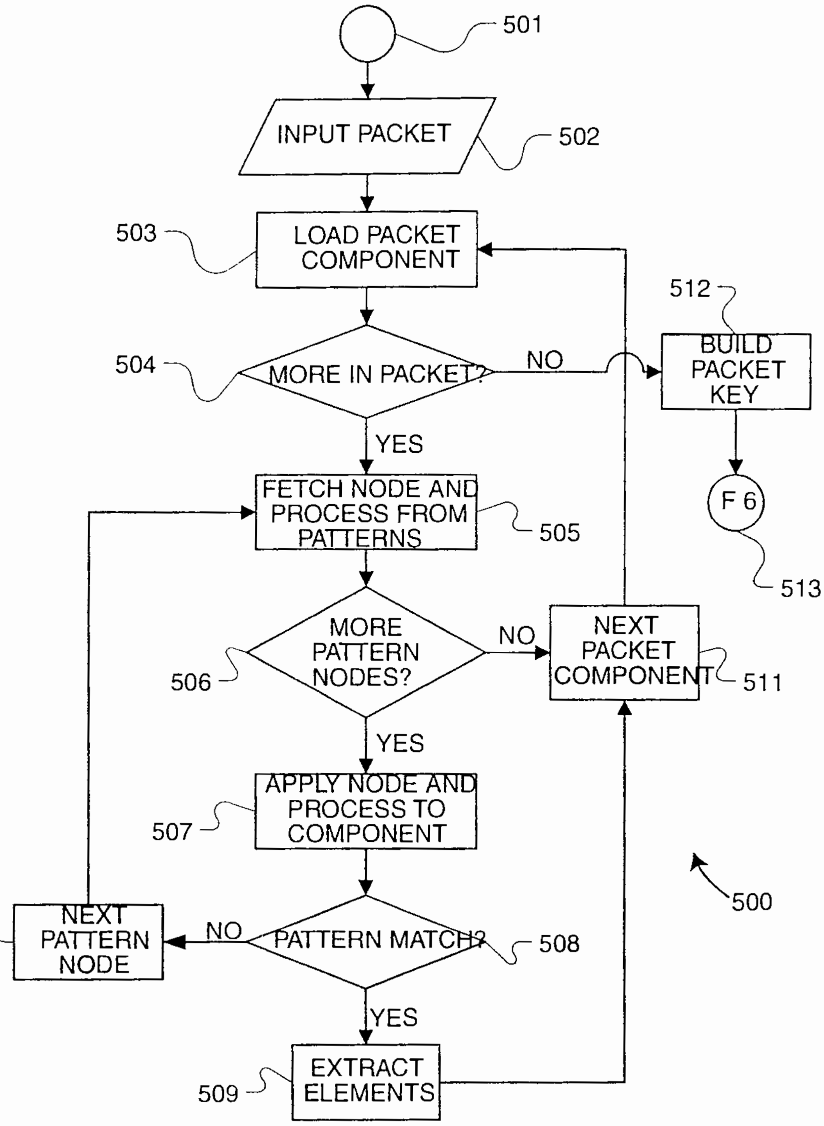

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
504
MORE IN PACKET?
NO
512
BUILD PACKET KEY
YES
FETCH NODE AND PROCESS FROM PATTERNS
505
F 6
513
MORE PATTERN NODES?
NO
NEXT PACKET COMPONENT
506
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
510
500
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT ELEMENTS
509

FIG. 5

PRINT OF DRAWING AS ORIGINALLY FILED

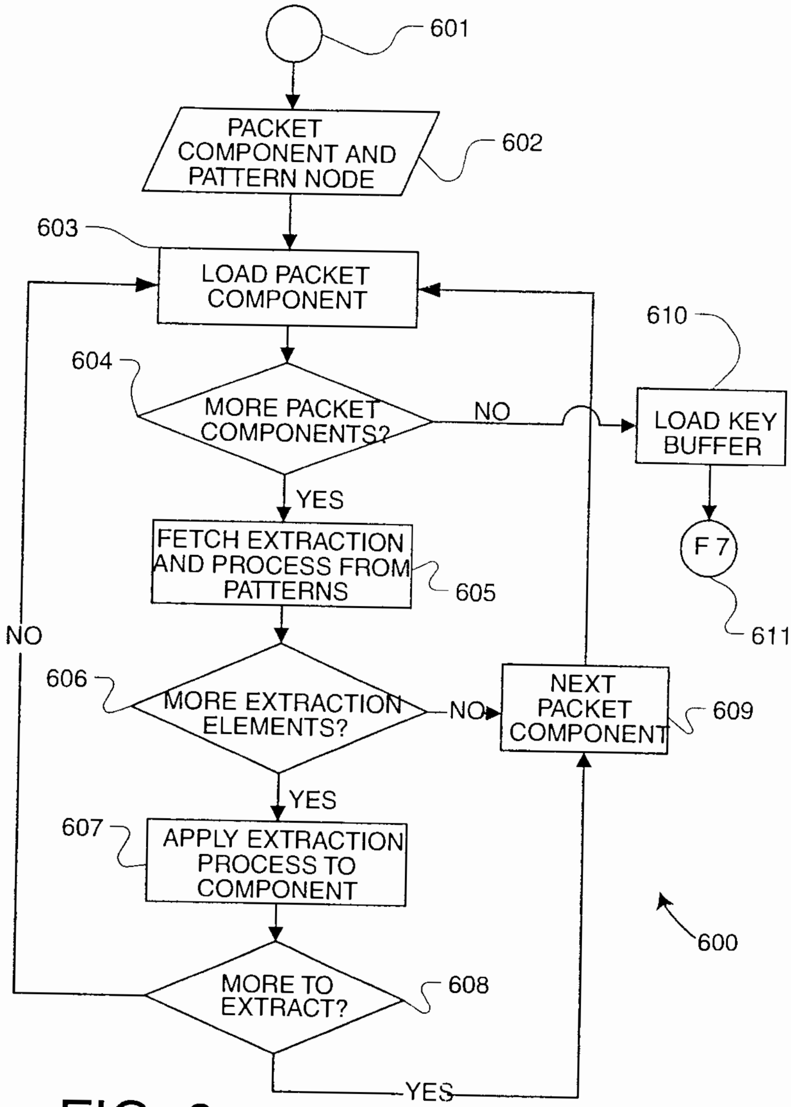


FIG. 6

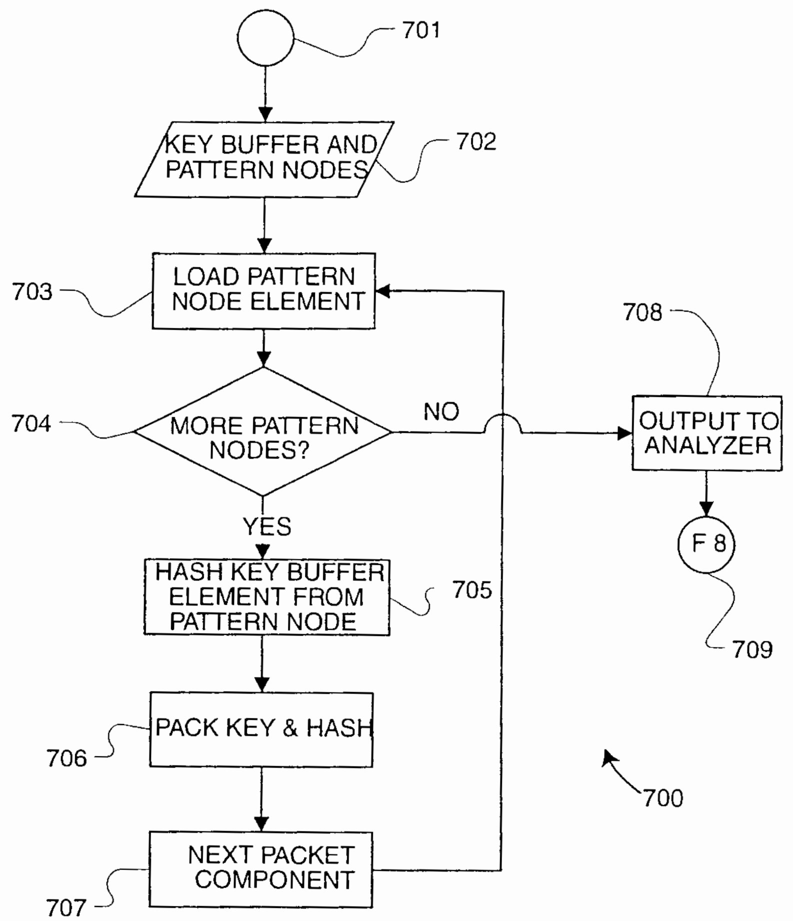

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
OUTPUT TO ANALYZER
708
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

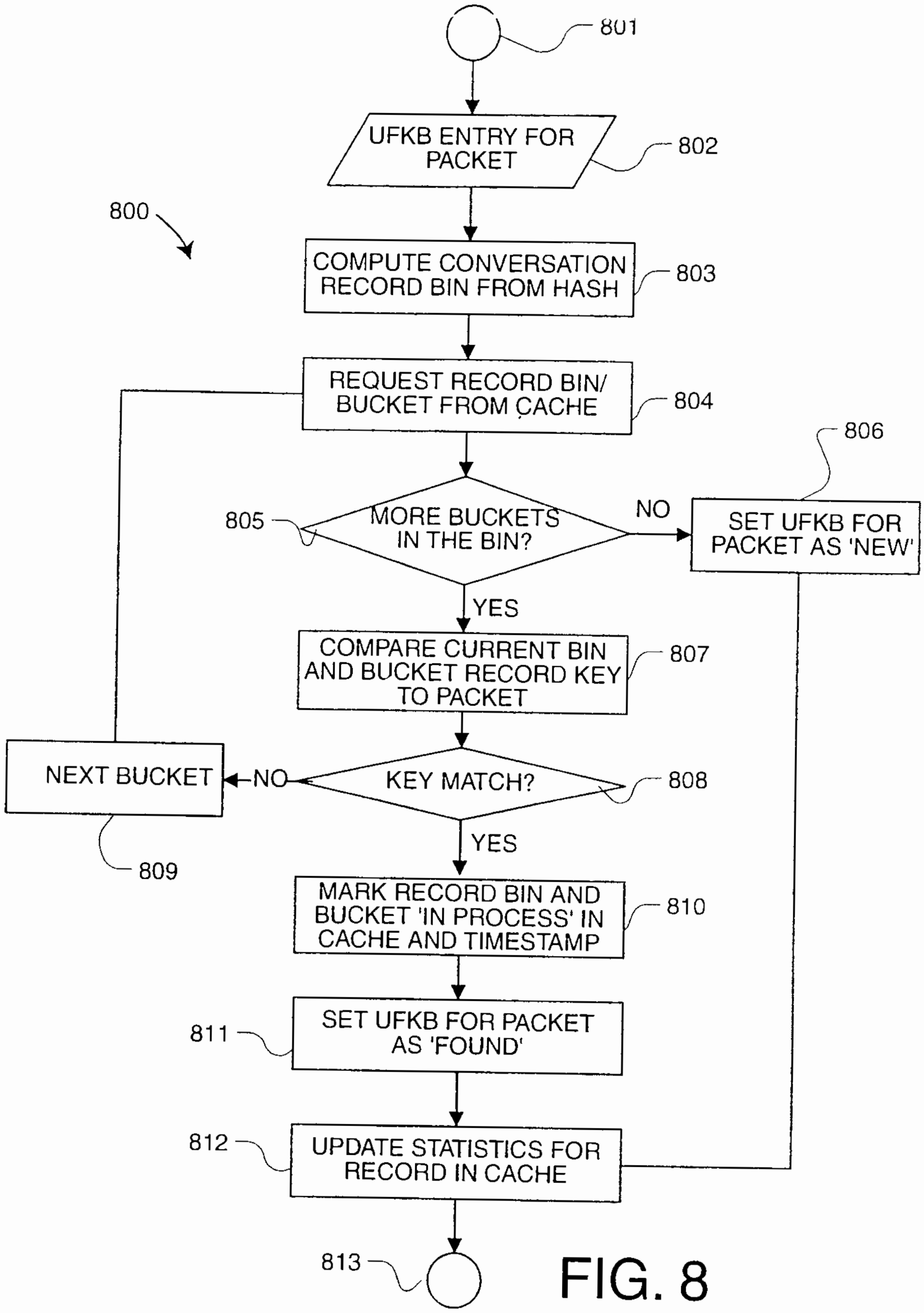

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
809
YES
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
811
SET UFKB FOR PACKET AS 'FOUND'
812
UPDATE STATISTICS FOR RECORD IN CACHE
813

FIG. 8

Sark

PRINT OF DRAWINGS AS ORIGINALLY FILED

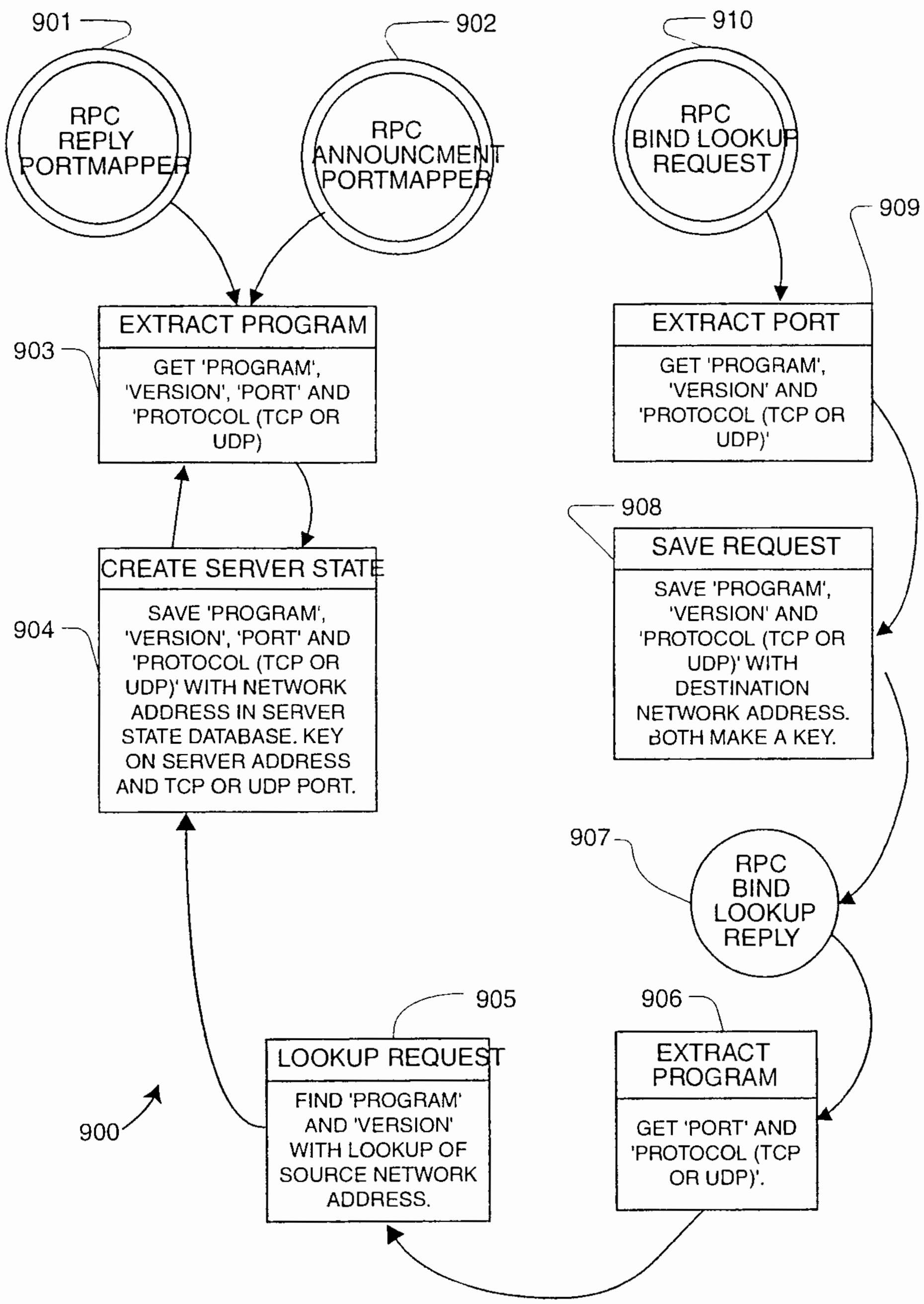


FIG. 9

Sarkissi…

PRINT OF DRAWINGS AS ORIGINALLY FILED

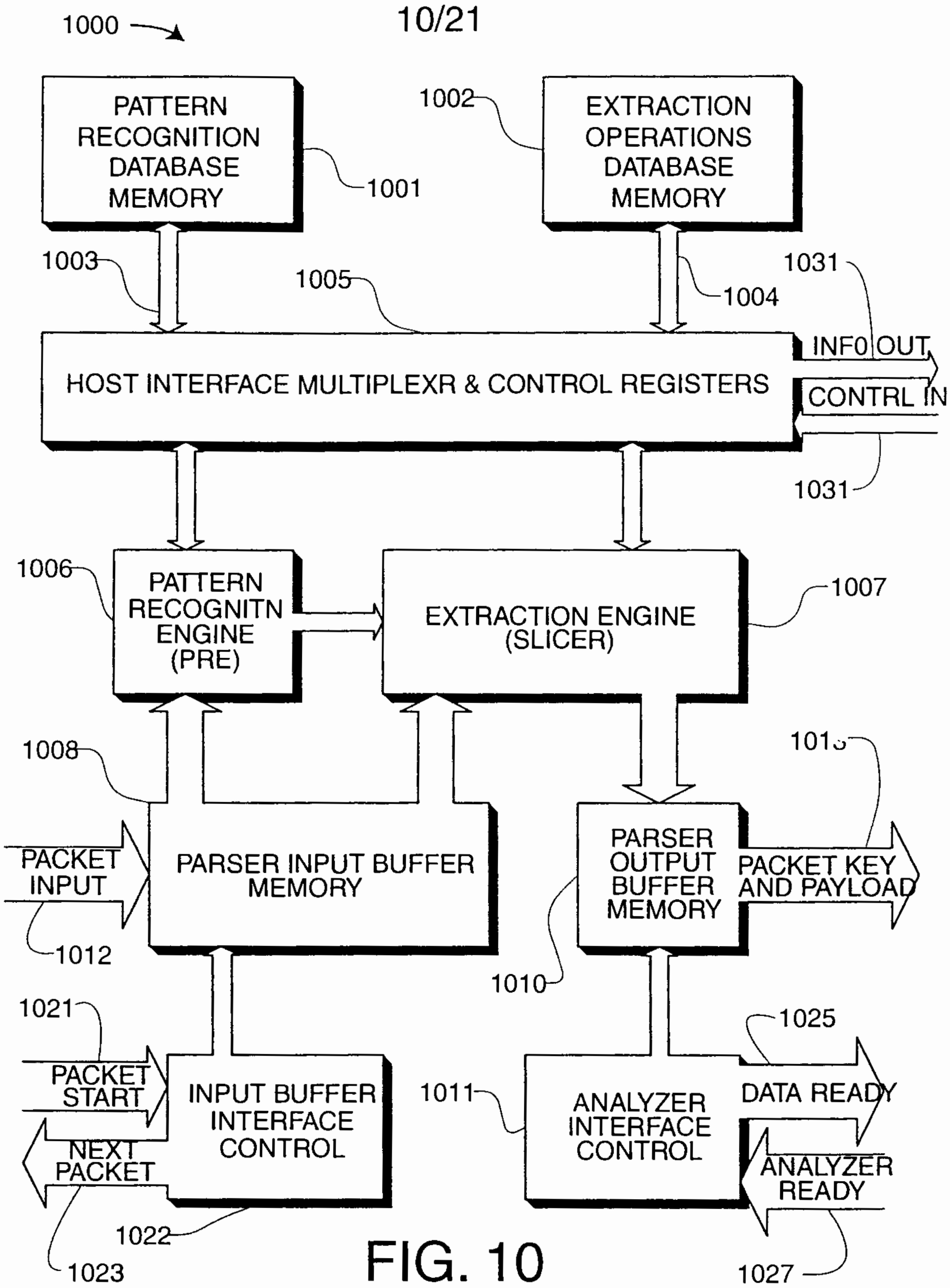


FIG. 10

Sarkis

PRINT OF DRAWINGS AS ORIGINALLY FILED

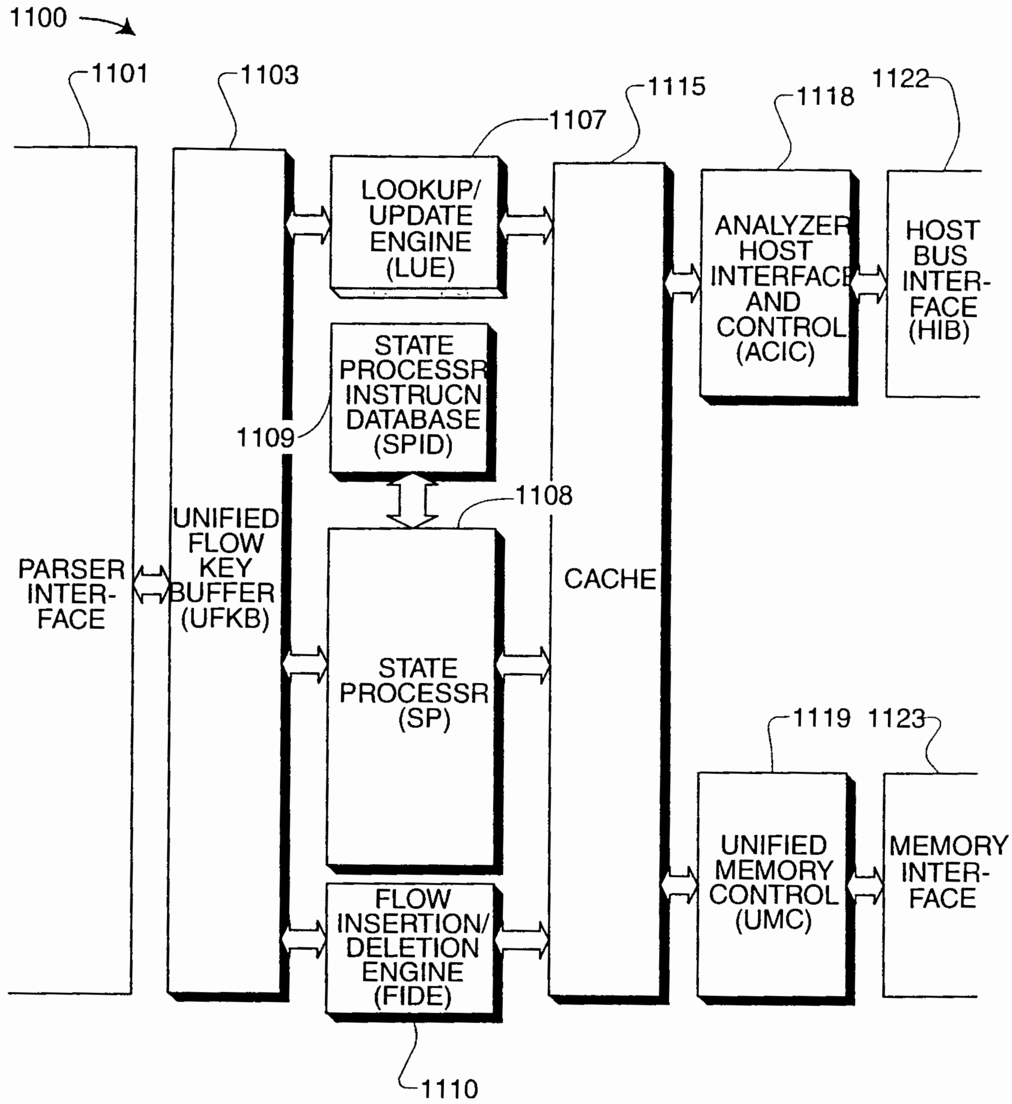


FIG. 11

Sarkiss

PRINT OF DRAWINGS
AS ORIGINALLY FILED

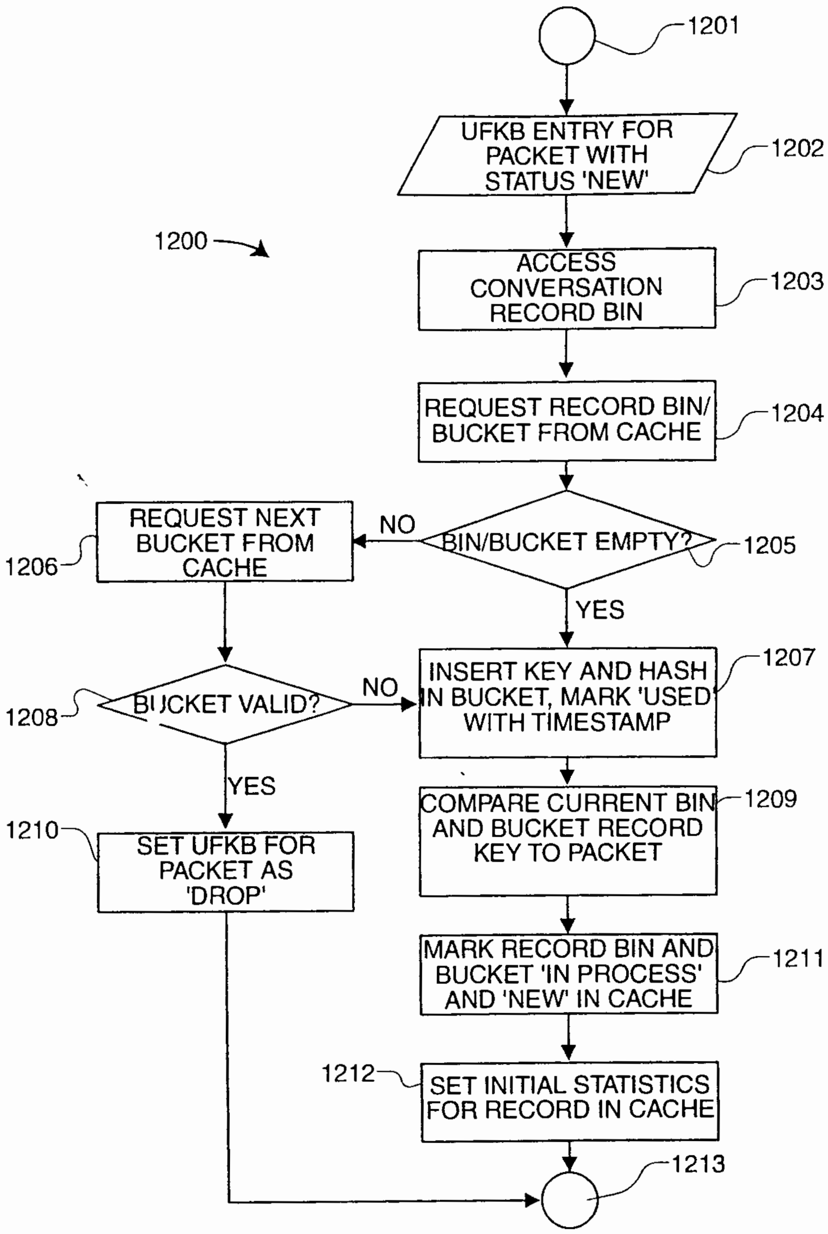


FIG. 12

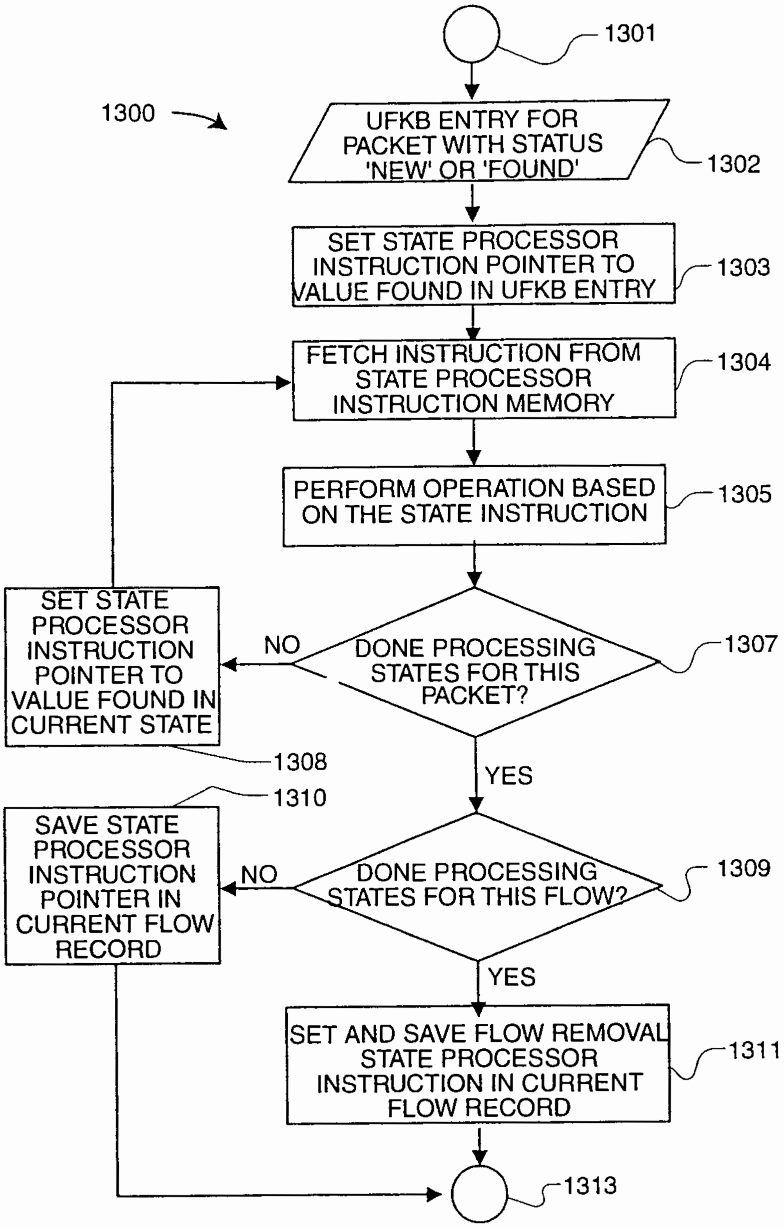

1300
1301
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307
NO
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
1308
YES
1310
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
NO
DONE PROCESSING STATES FOR THIS FLOW?
1309
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

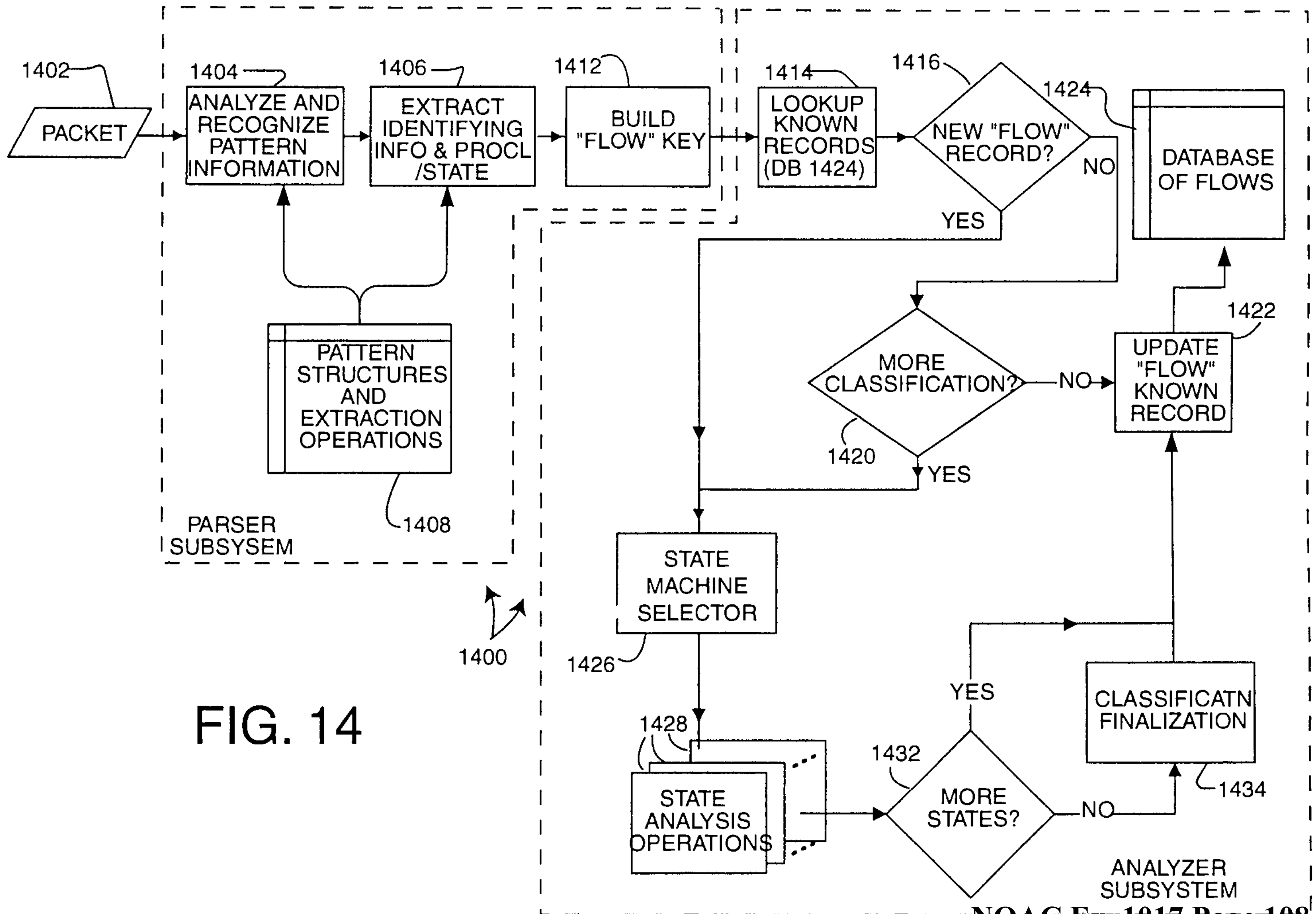

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1412
BUILD "FLOW" KEY
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
1424
NO
YES
DATABASE OF FLOWS
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
1400
MORE CLASSIFICATION?
1420
NO
YES
1422
UPDATE "FLOW" KNOWN RECORD
STATE MACHINE SELECTOR
1426
1428
STATE ANALYSIS OPERATIONS
1432
MORE STATES?
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM

FIG. 14

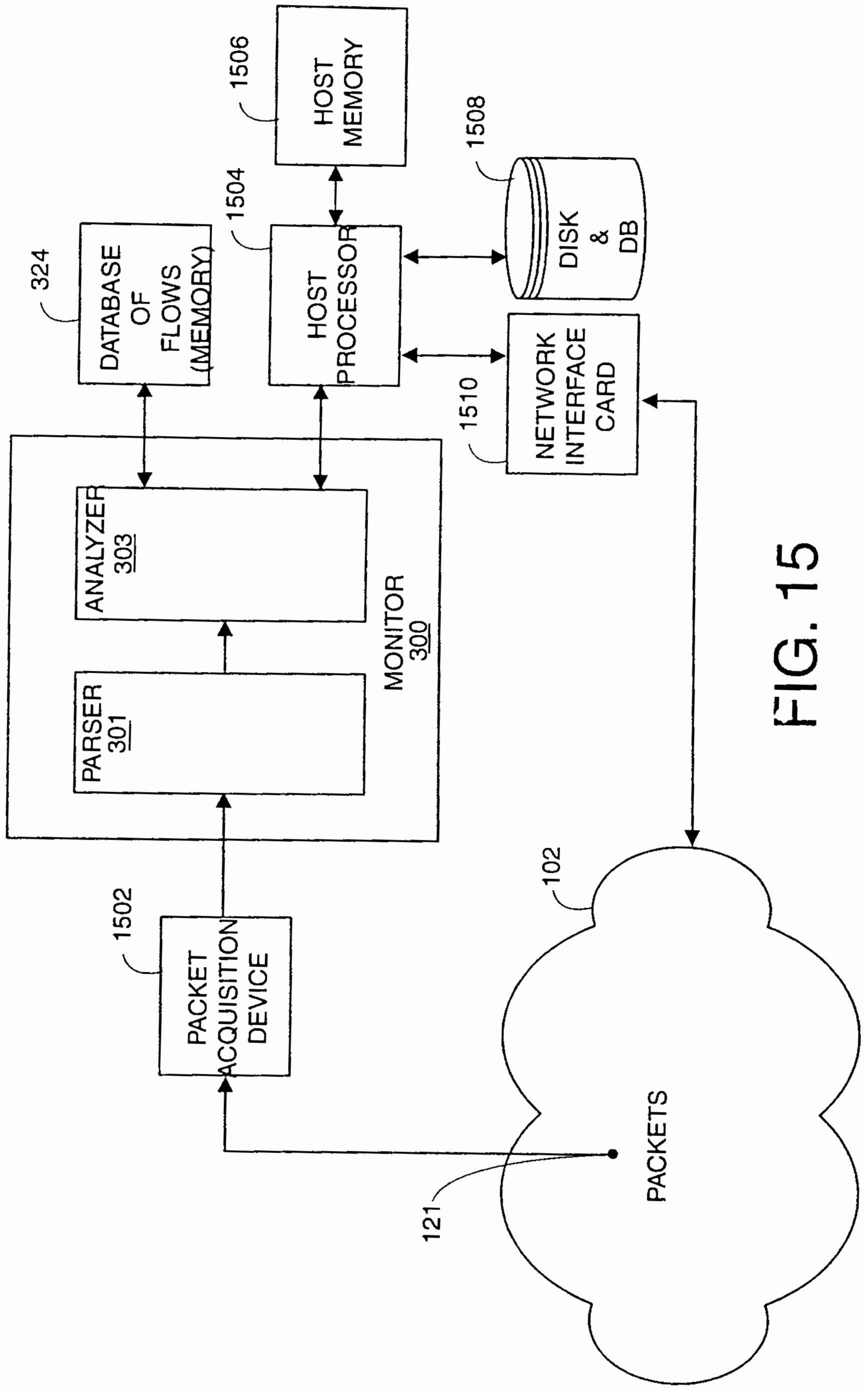

1506
HOST MEMORY
1508
DISK & DB
324
DATABASE OF FLOWS (MEMORY)
1504
HOST PROCESSOR
1510
NETWORK INTERFACE CARD
ANALYZER 303
PARSER 301
MONITOR 300
1502
PACKET ACQUISITION DEVICE
102
PACKETS
121

FIG. 15

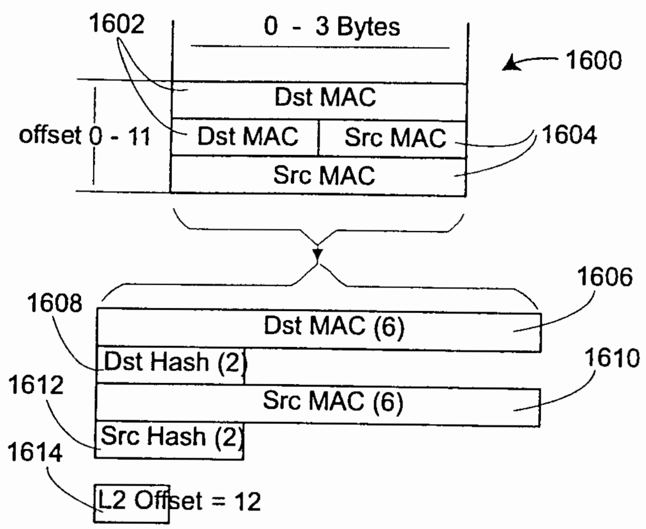

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

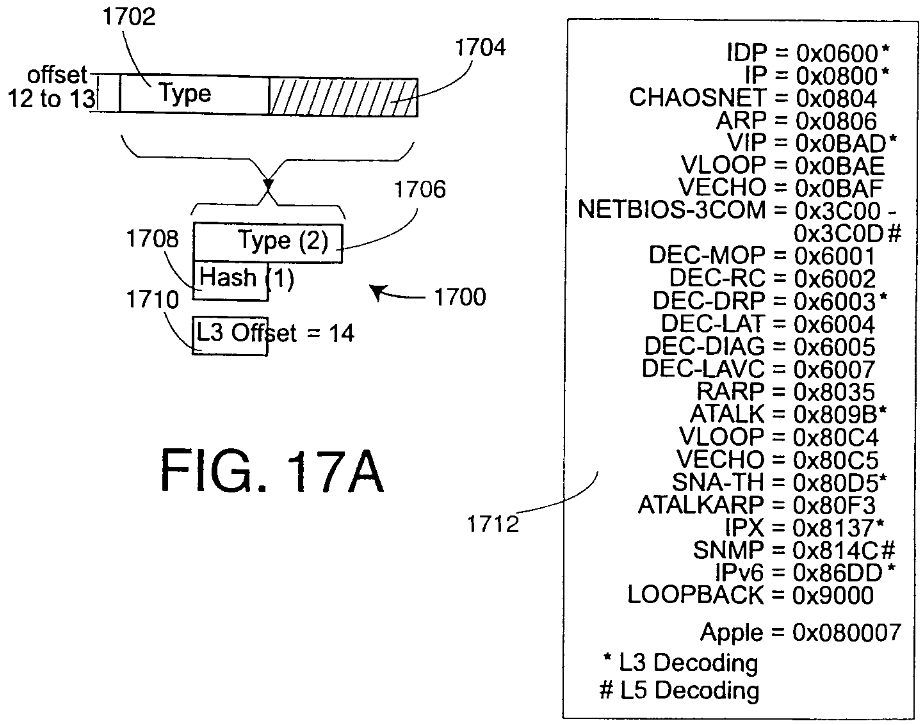


FIG. 17A

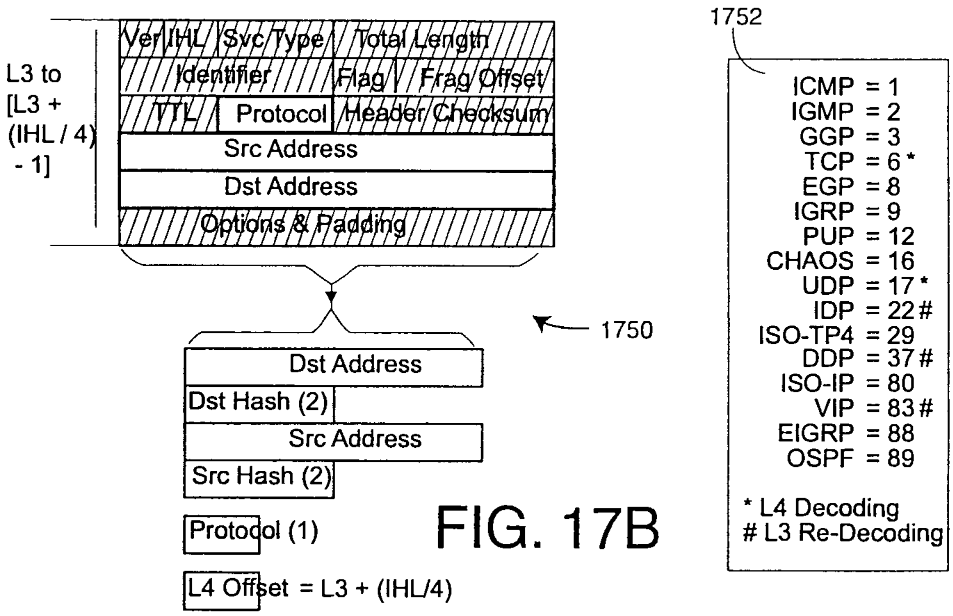


FIG. 17B

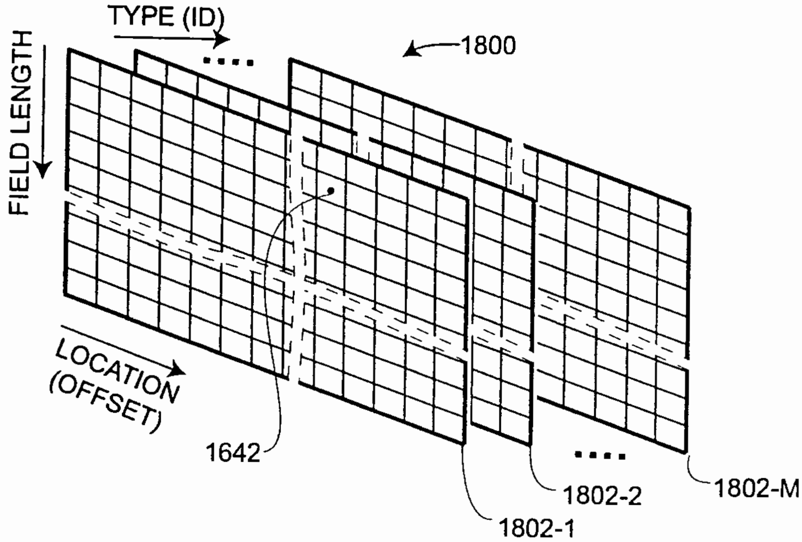


FIG. 18A

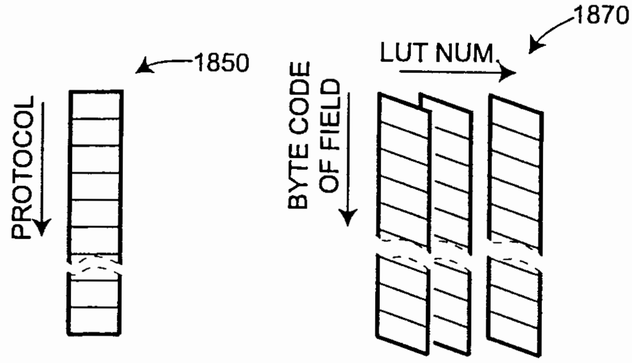


FIG. 18B

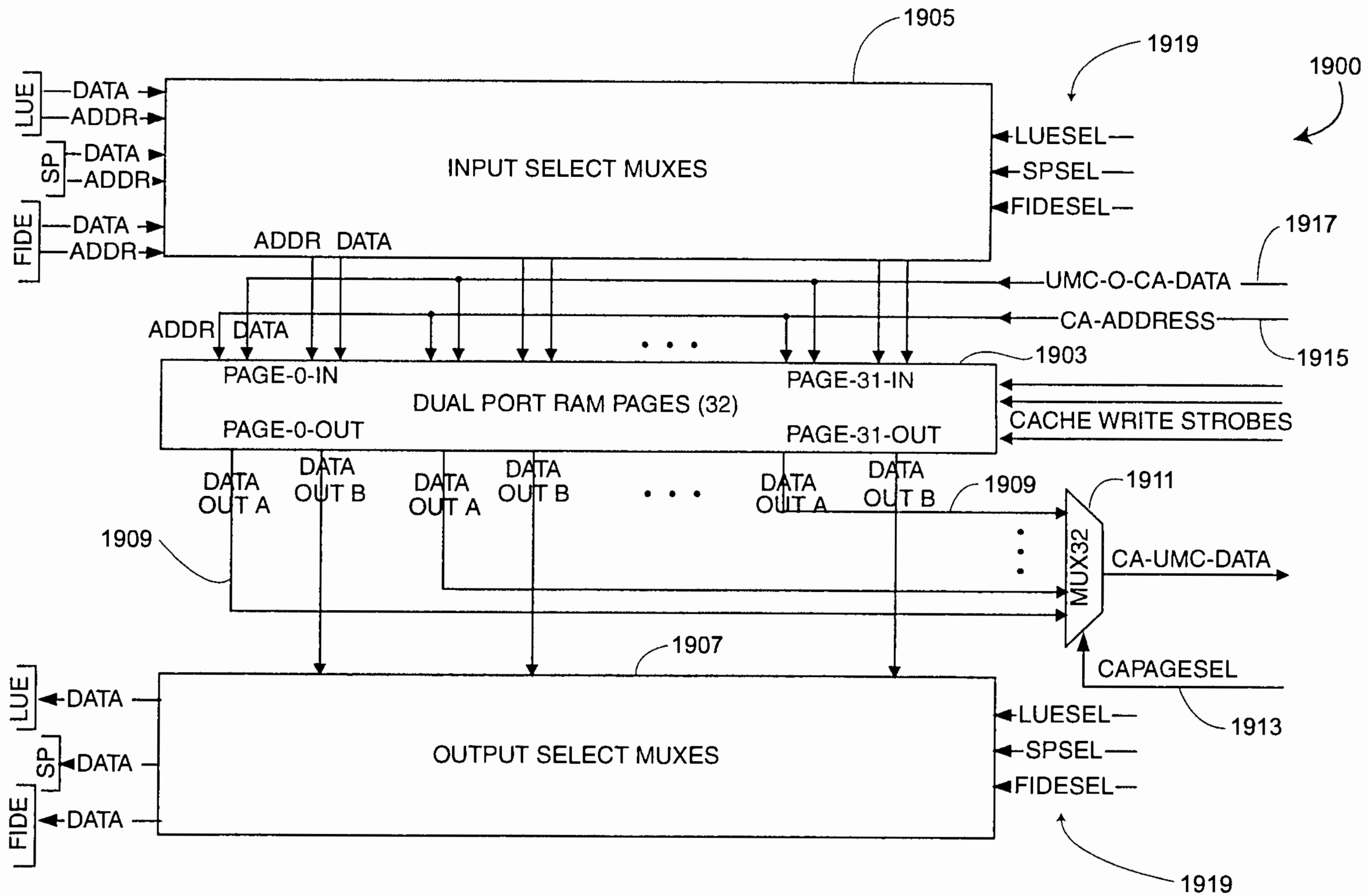


FIG. 19

19/21

PRINT OF DRAWINGS AS ORIGINALLY FILED

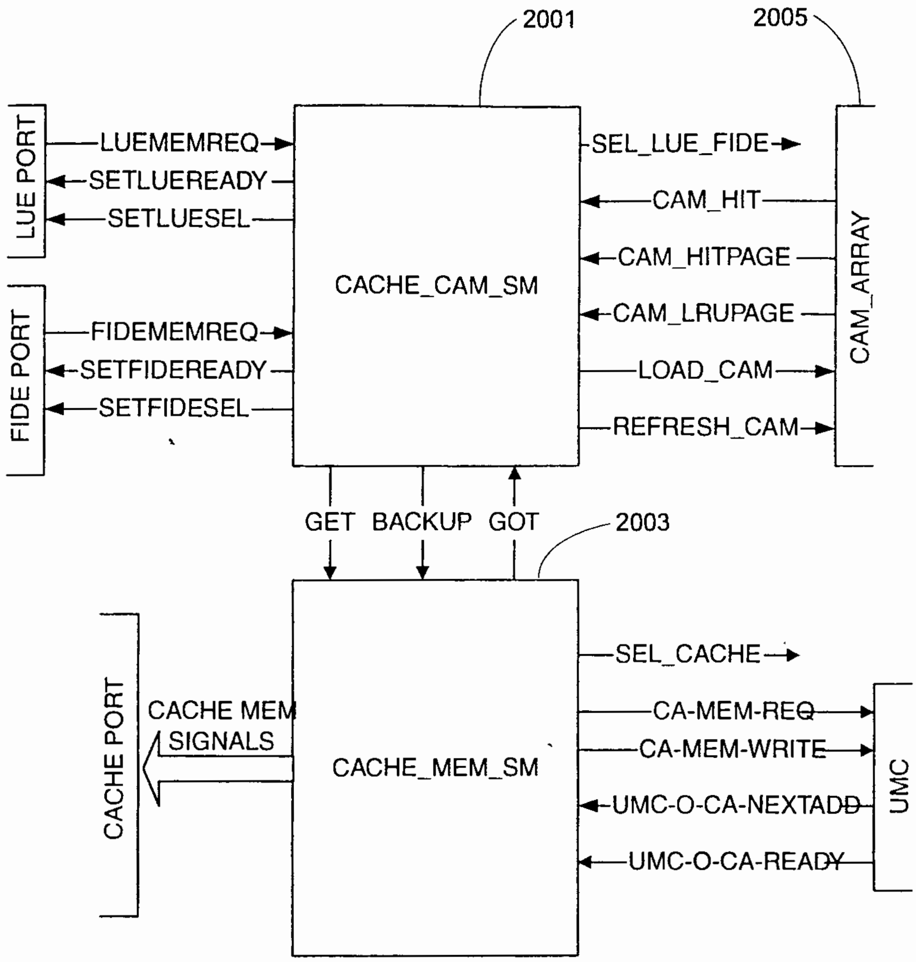

2001
2005
LUE PORT
LUEMEMREQ
SETLUEREADY
SETLUESEL
FIDE PORT
FIDEMEMREQ
SETFIDEREADY
SETFIDESEL
CACHE_CAM_SM
SEL_LUE_FIDE
CAM_HIT
CAM_HITPAGE
CAM_LRUPAGE
LOAD_CAM
REFRESH_CAM
CAM_ARRAY
GET
BACKUP
GOT
2003
CACHE PORT
CACHE MEM
SIGNALS
CACHE_MEM_SM
SEL_CACHE
CA-MEM-REQ
CA-MEM-WRITE
UMC-O-CA-NEXTADD
UMC-O-CA-READY
UMC

FIG. 20

PRINT OF DRAWINGS
AS ORIGINALLY FILED

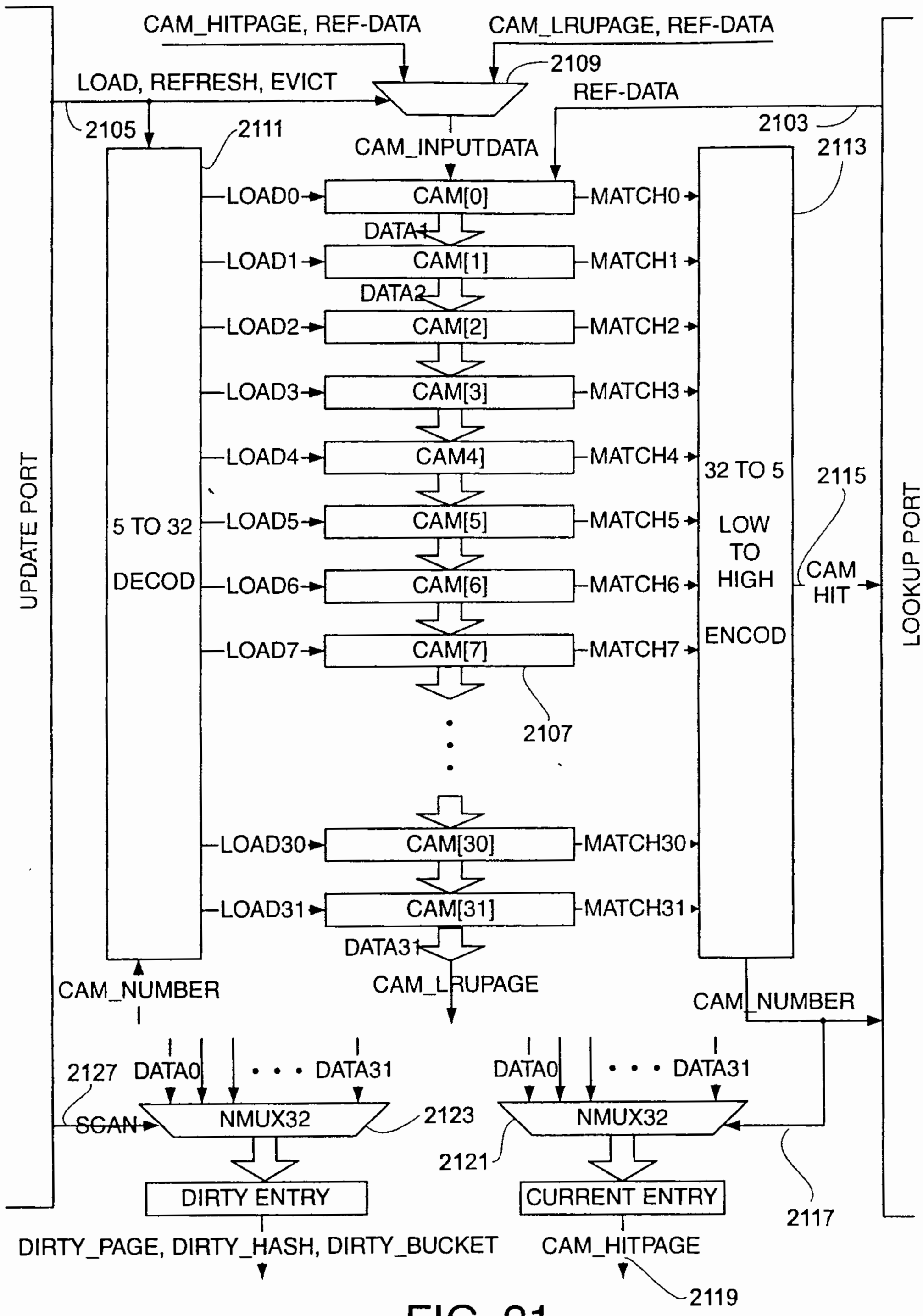


FIG. 21

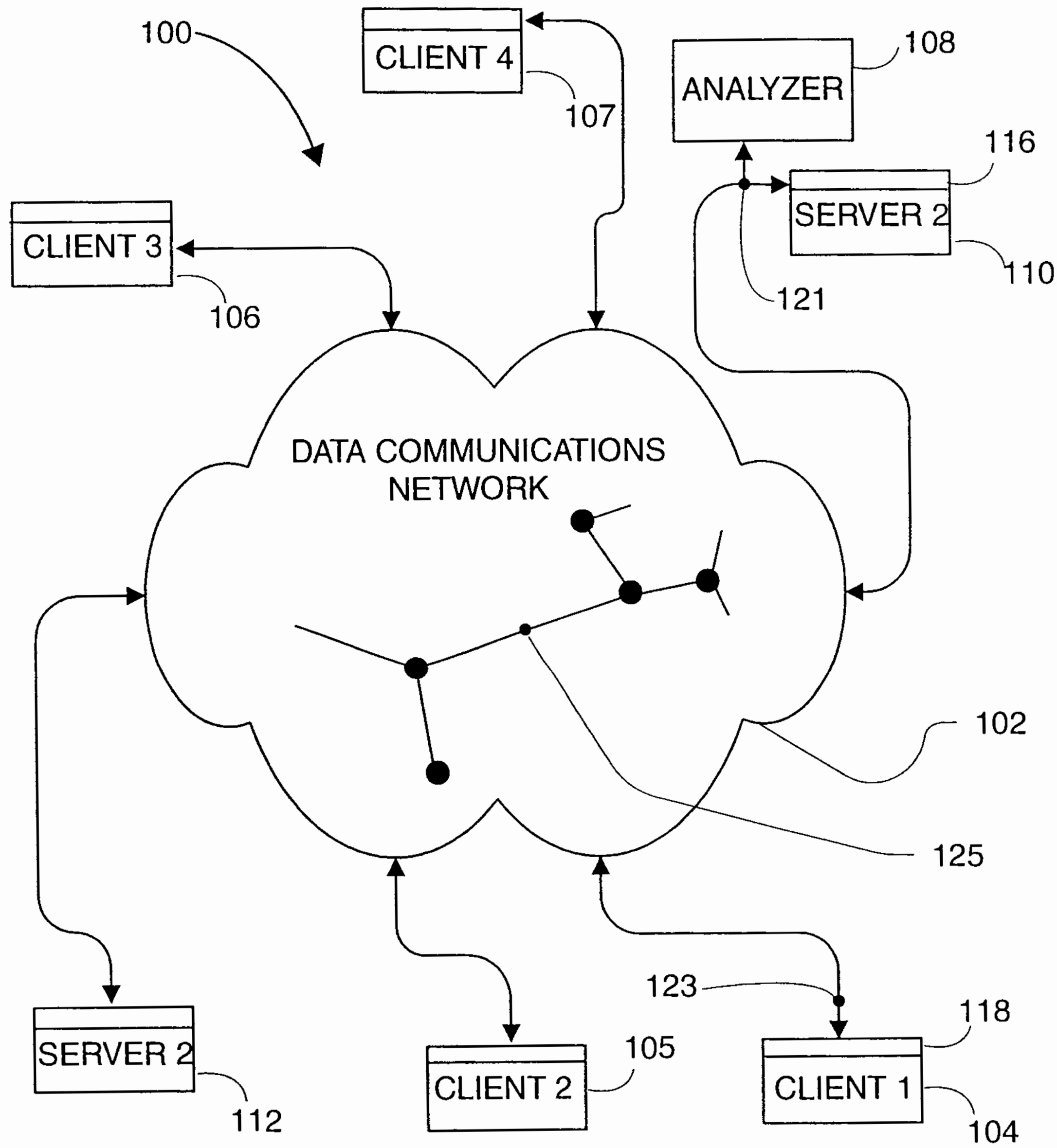

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

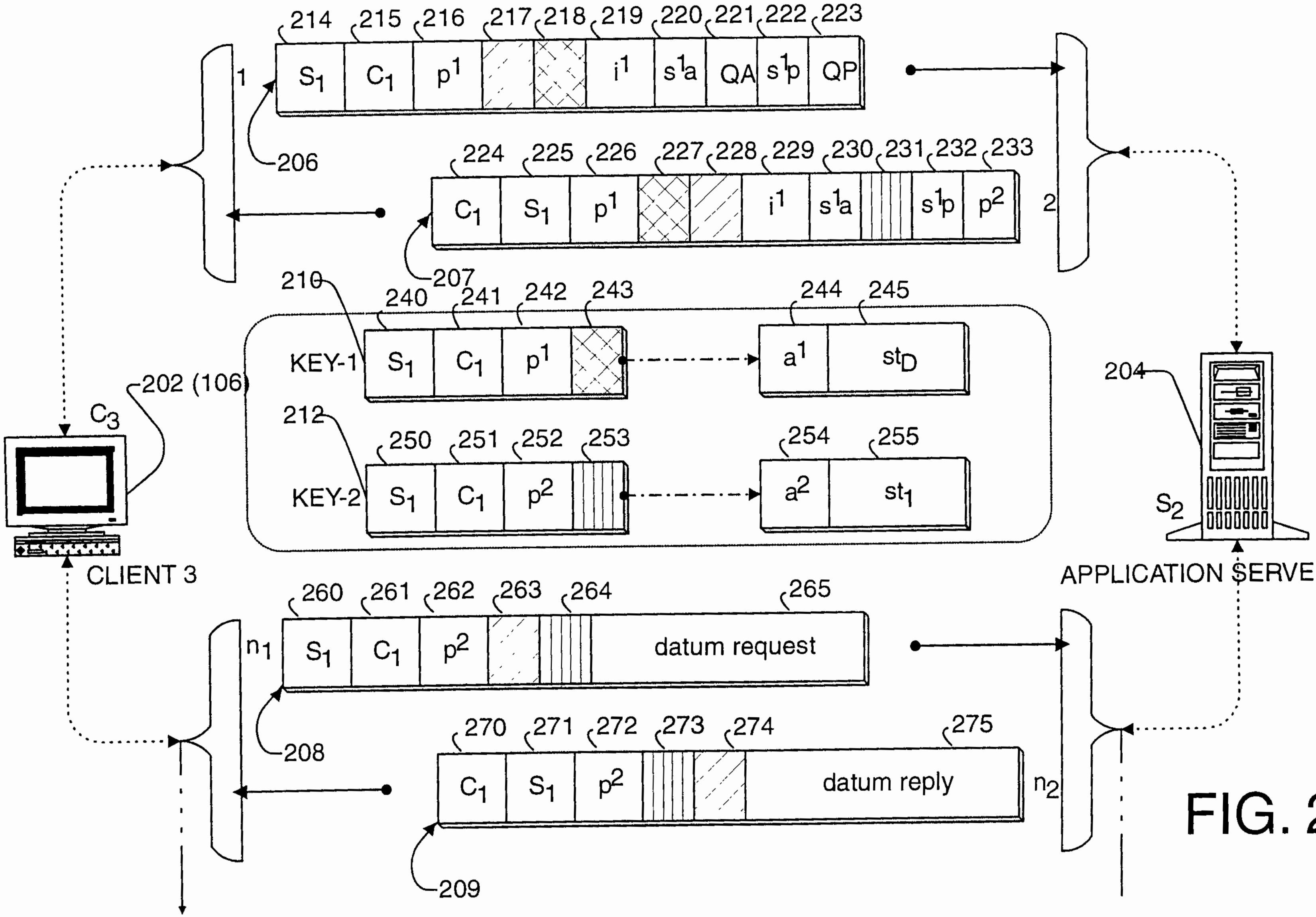

214 215 216 217 218 219 220 221 222 223
1
S1 C1 p1 i1 s1a QA s1p QP
206
224 225 226 227 228 229 230 231 232 233
C1 S1 p1 i1 s1a s1p p2
2
207
210
240 241 242 243
KEY-1 S1 C1 p1
244 245
a1 stD
212
250 251 252 253
KEY-2 S1 C1 p2
254 255
a2 st1
C3
202 (106)
CLIENT 3
204
S2
APPLICATION SERVER 2
260 261 262 263 264 265
n1
S1 C1 p2 datum request
208
270 271 272 273 274 275
C1 S1 p2 datum reply
n2
209

FIG. 2

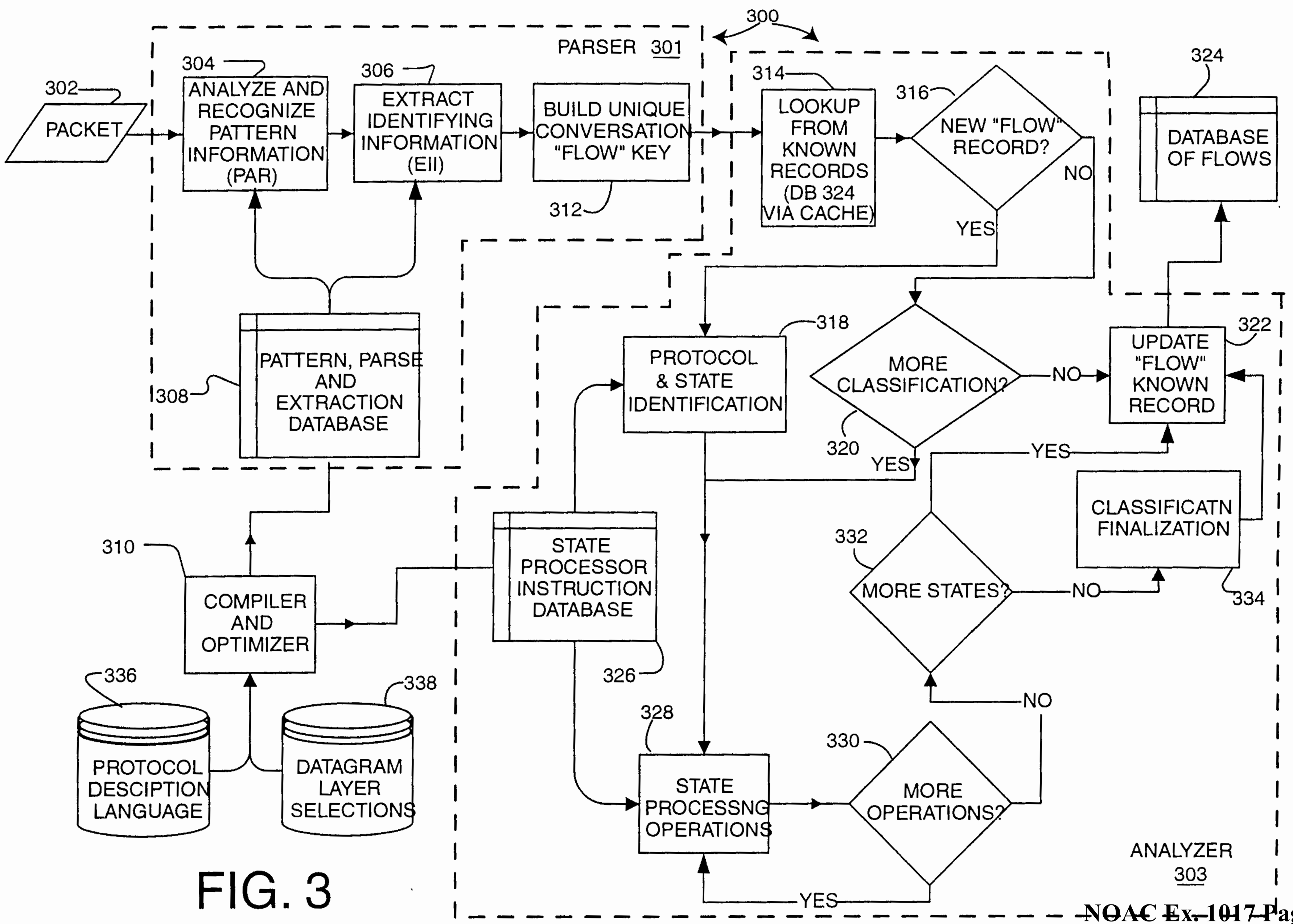


FIG. 3

Sarkissian et al. APPT-001-4

3/21

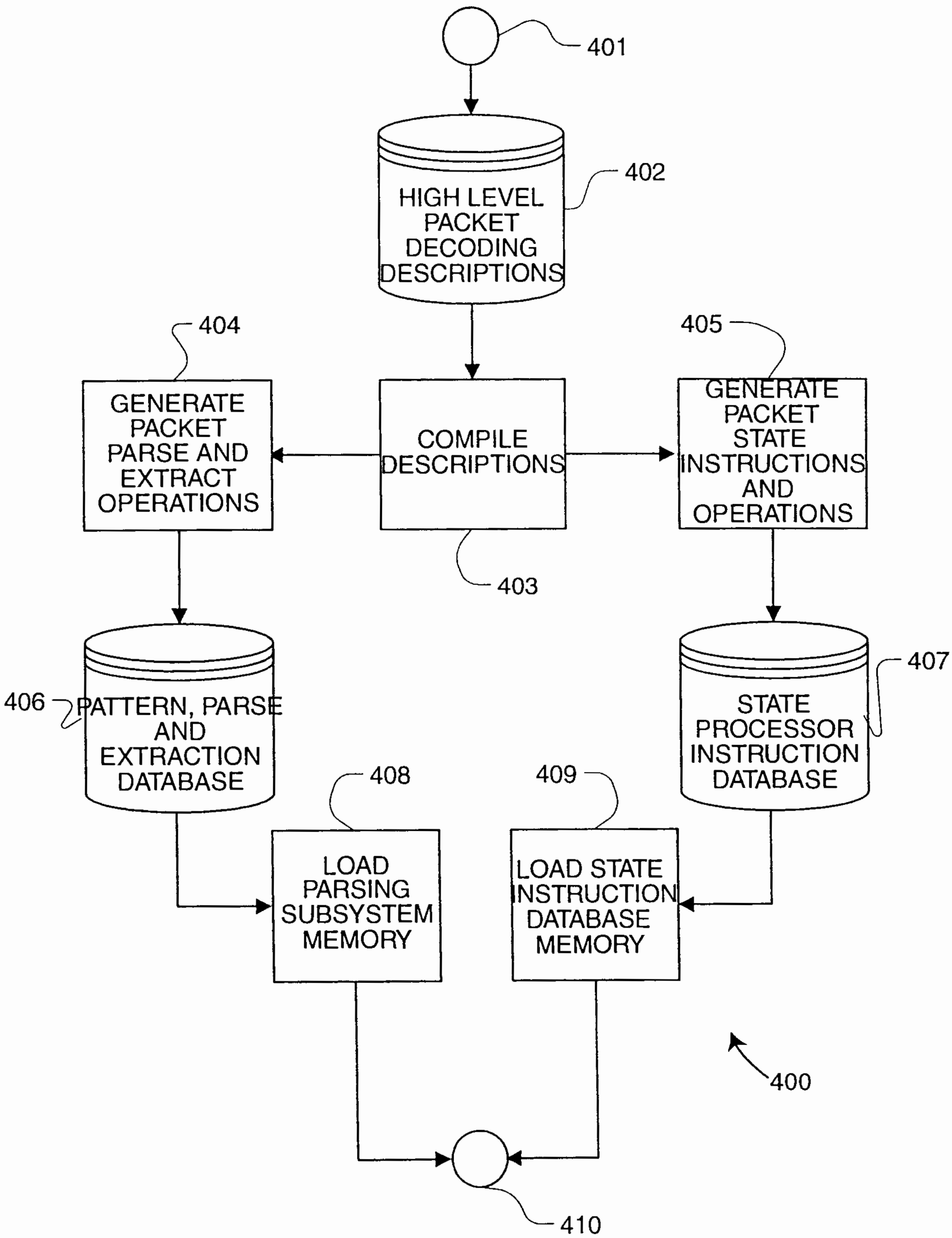

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
405
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
403
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
409
LOAD PARSING SUBSYSTEM MEMORY
LOAD STATE INSTRUCTION DATABASE MEMORY
400
410

FIG. 4

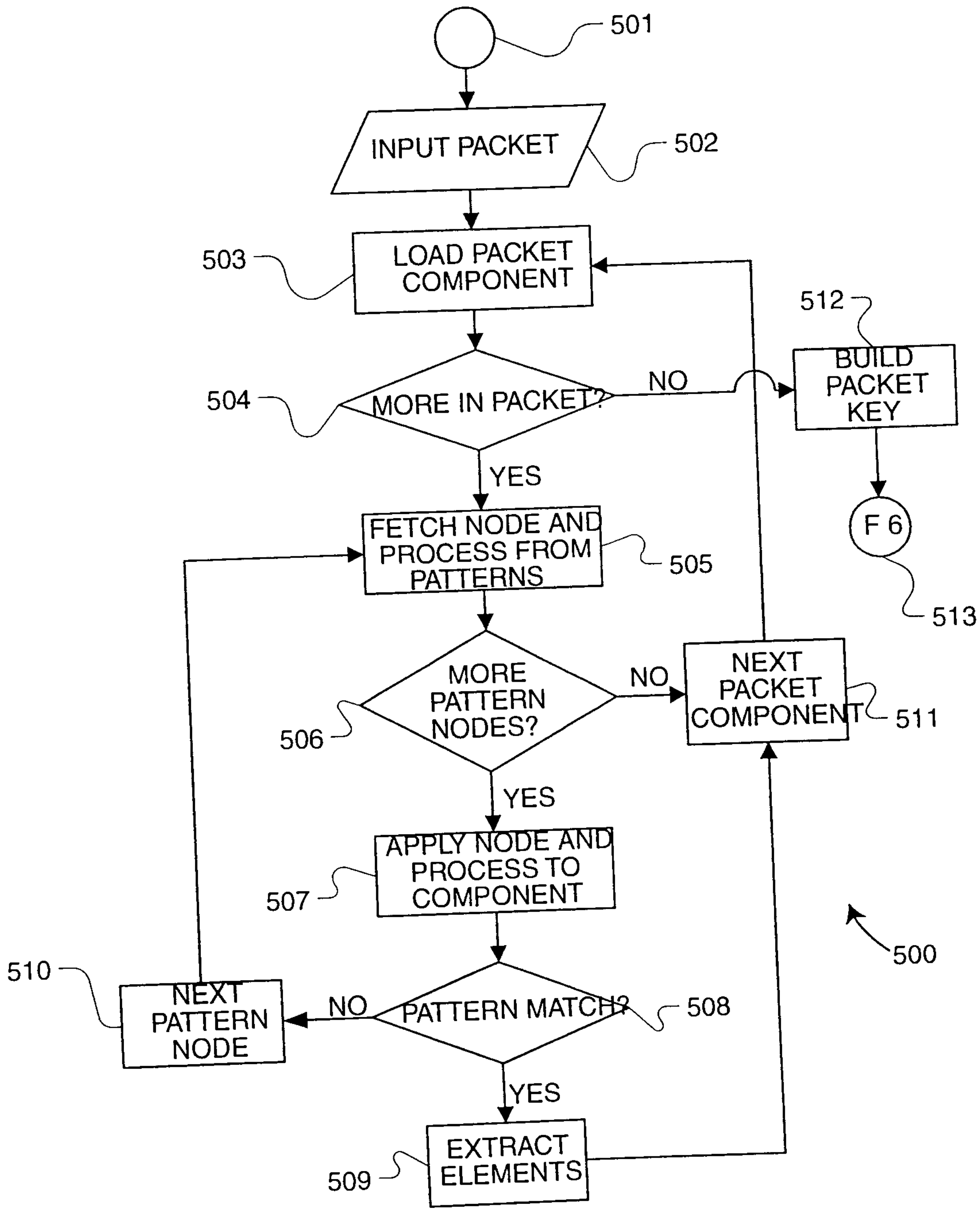

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
504
MORE IN PACKET?
NO
YES
512
BUILD PACKET KEY
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
NO
NEXT PACKET COMPONENT
511
506
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT ELEMENTS
509

FIG. 5

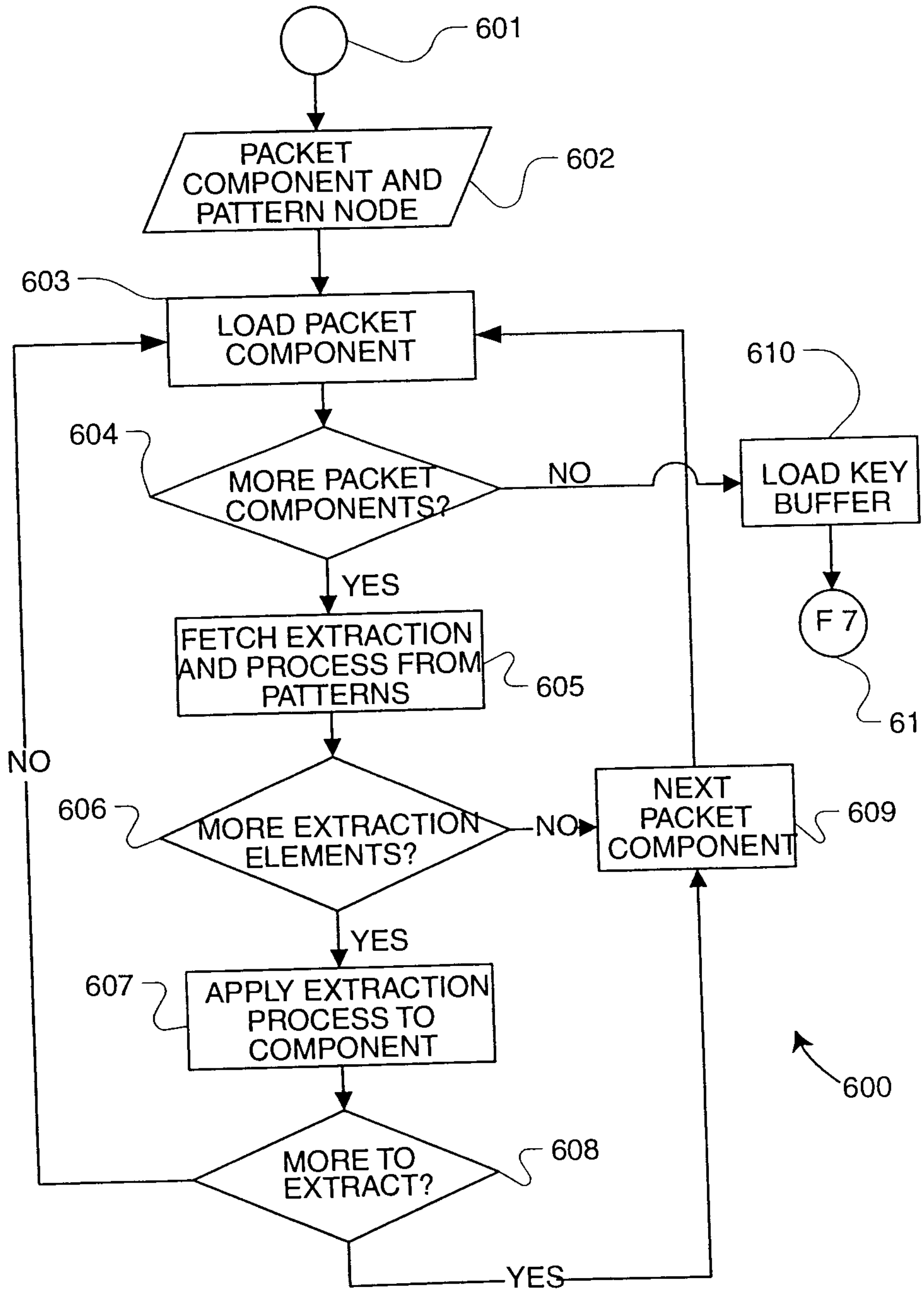

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

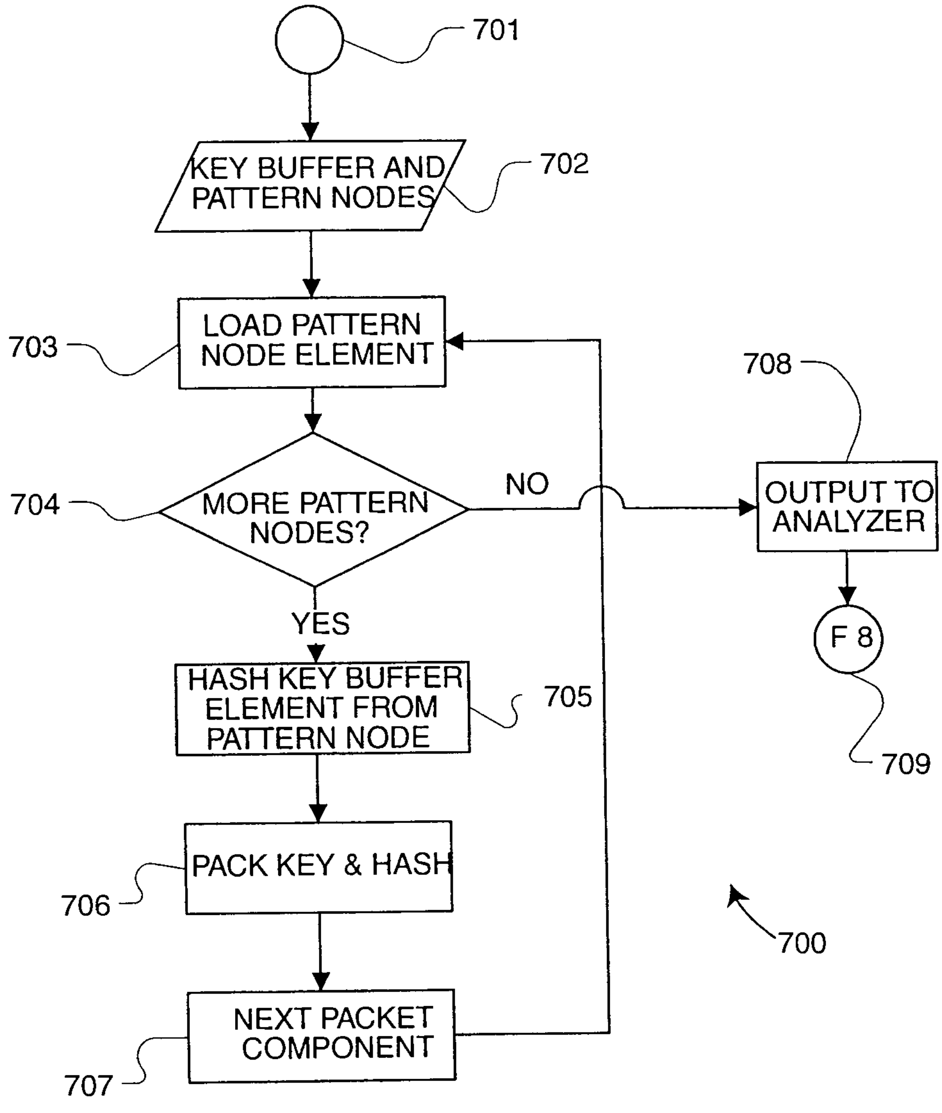

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
OUTPUT TO ANALYZER
708
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

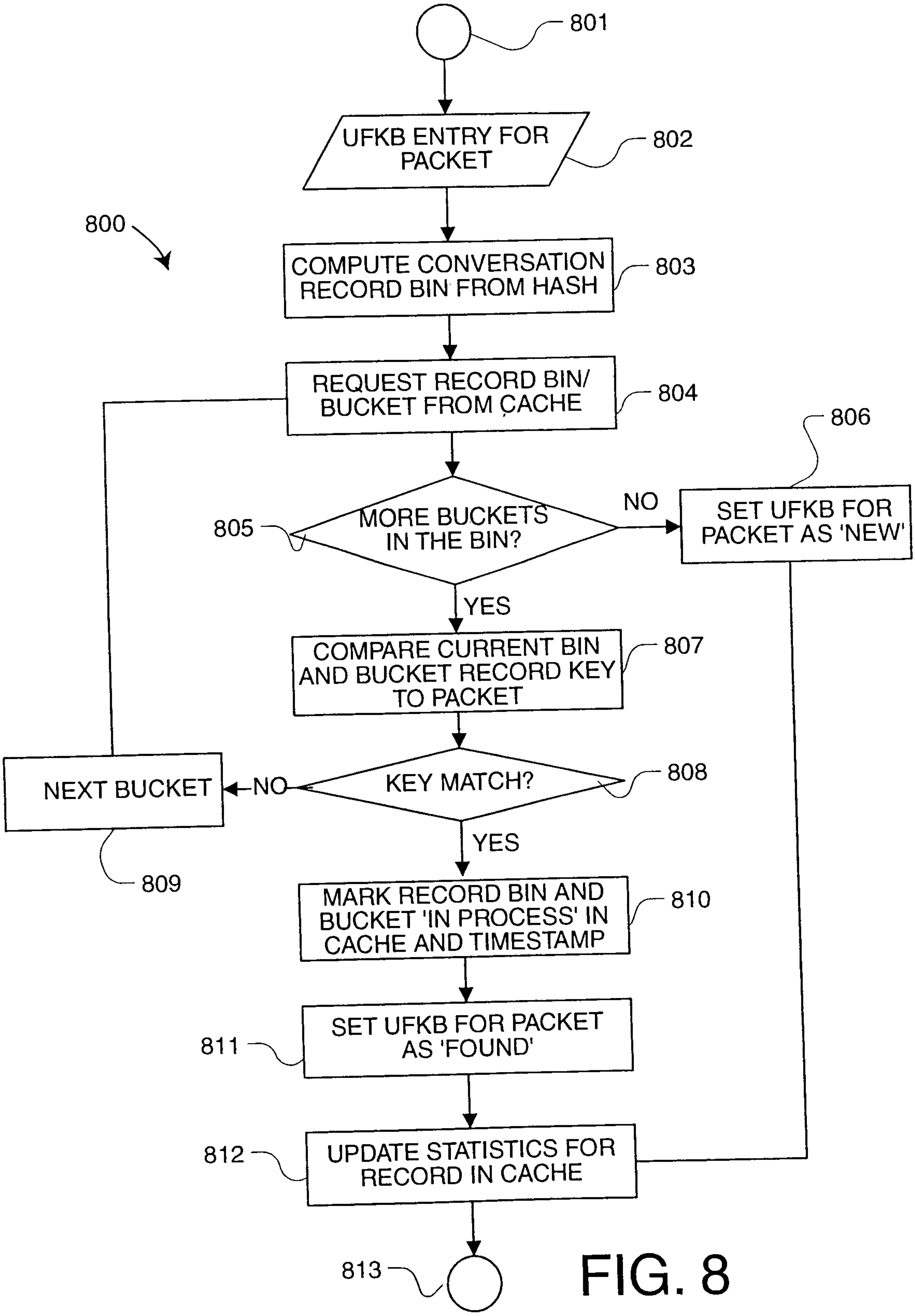

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
MORE BUCKETS IN THE BIN?
805
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
KEY MATCH?
808
NEXT BUCKET
NO
809
YES
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
SET UFKB FOR PACKET AS 'FOUND'
811
UPDATE STATISTICS FOR RECORD IN CACHE
812
813

FIG. 8

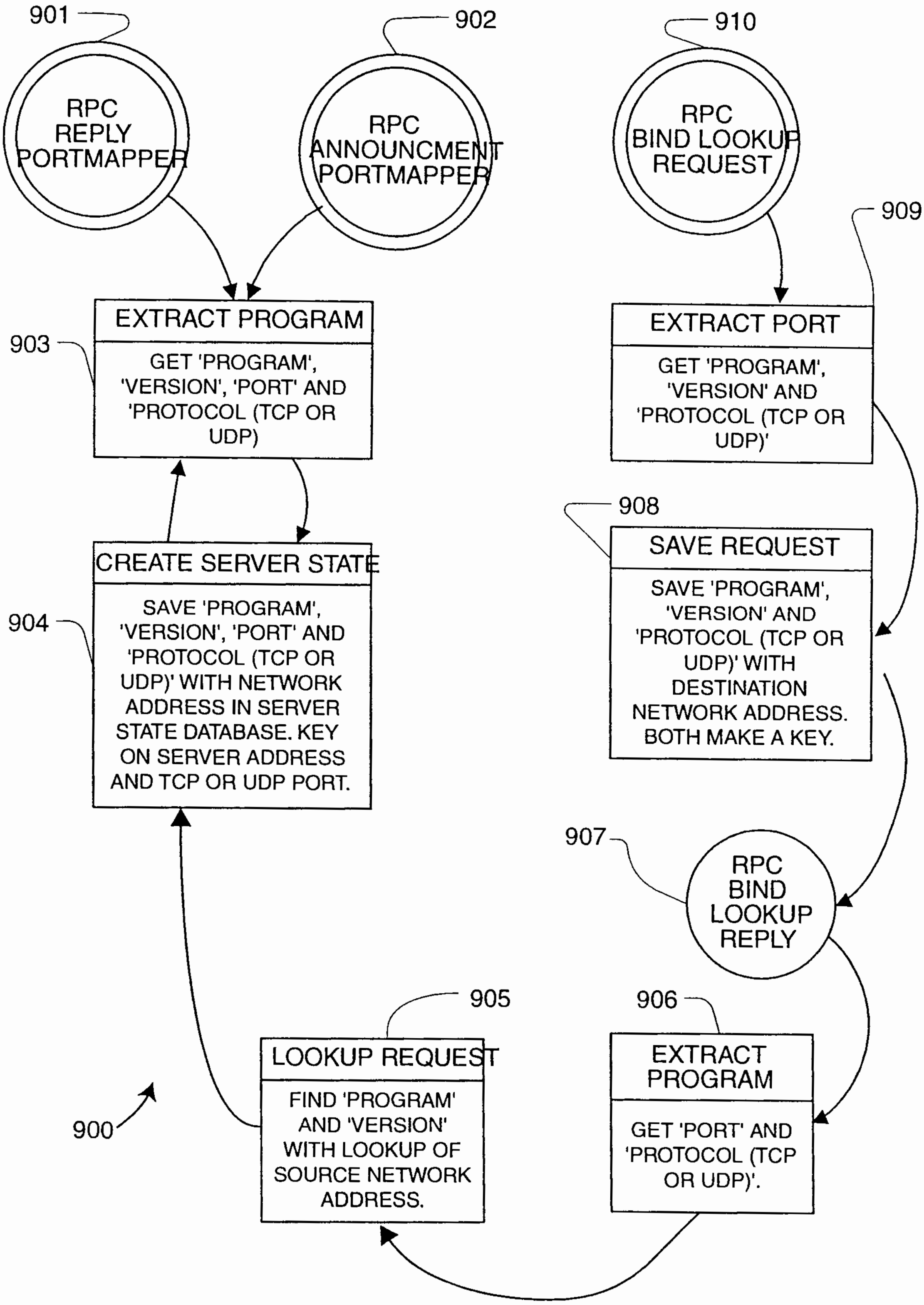

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
906
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

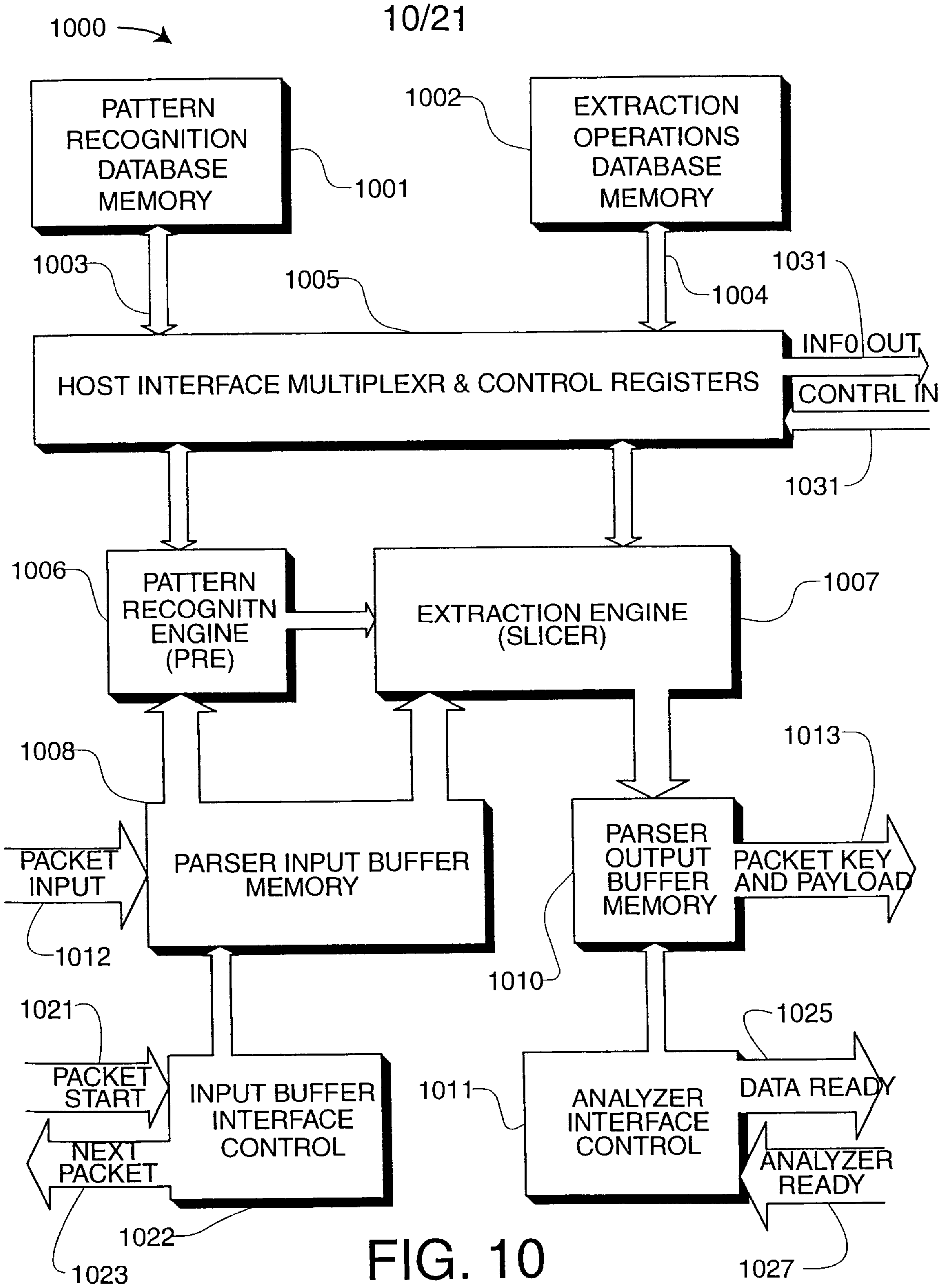

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
INFO OUT
CONTRL IN
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

FIG. 10

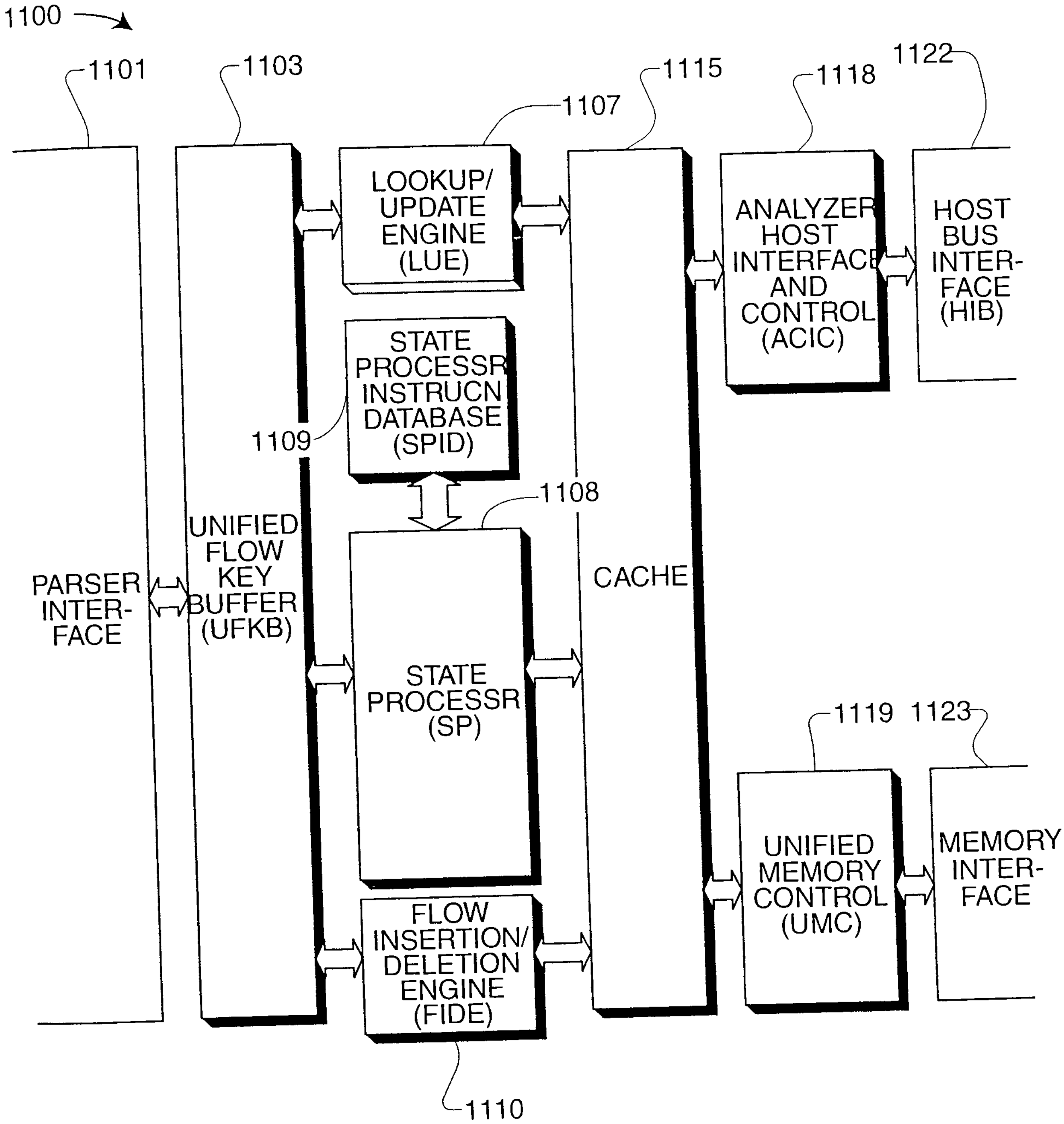

1100
1101
1103
1107
1115
1118
1122
PARSER INTER-FACE
UNIFIED FLOW KEY BUFFER (UFKB)
LOOKUP/ UPDATE ENGINE (LUE)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
STATE PROCESSR (SP)
FLOW INSERTION/ DELETION ENGINE (FIDE)
1110
CACHE
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY INTER-FACE

FIG. 11

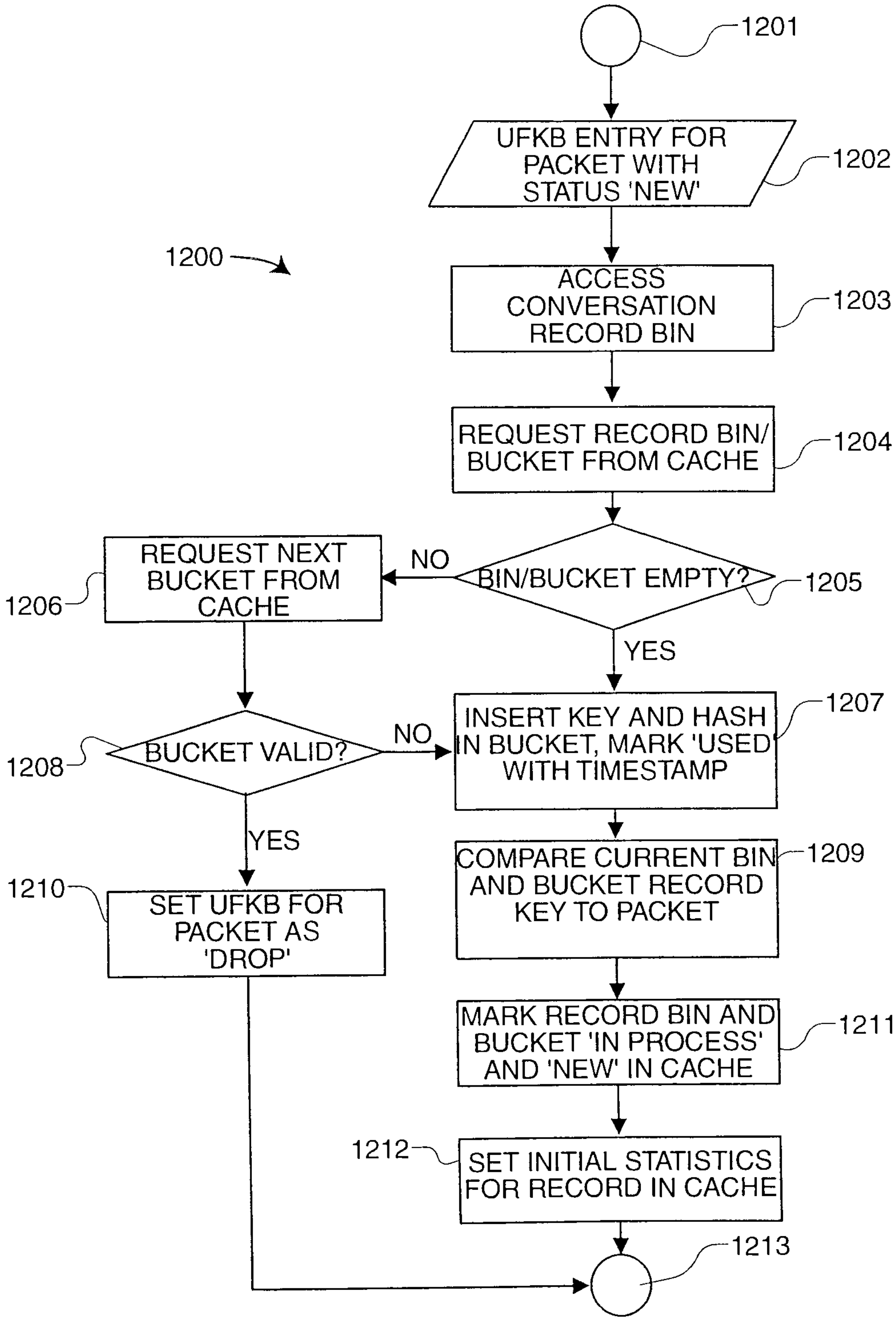

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
REQUEST NEXT BUCKET FROM CACHE
1206
NO
BIN/BUCKET EMPTY?
1205
YES
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
1208
BUCKET VALID?
NO
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
1210
SET UFKB FOR PACKET AS 'DROP'
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
1212
SET INITIAL STATISTICS FOR RECORD IN CACHE
1213

FIG. 12

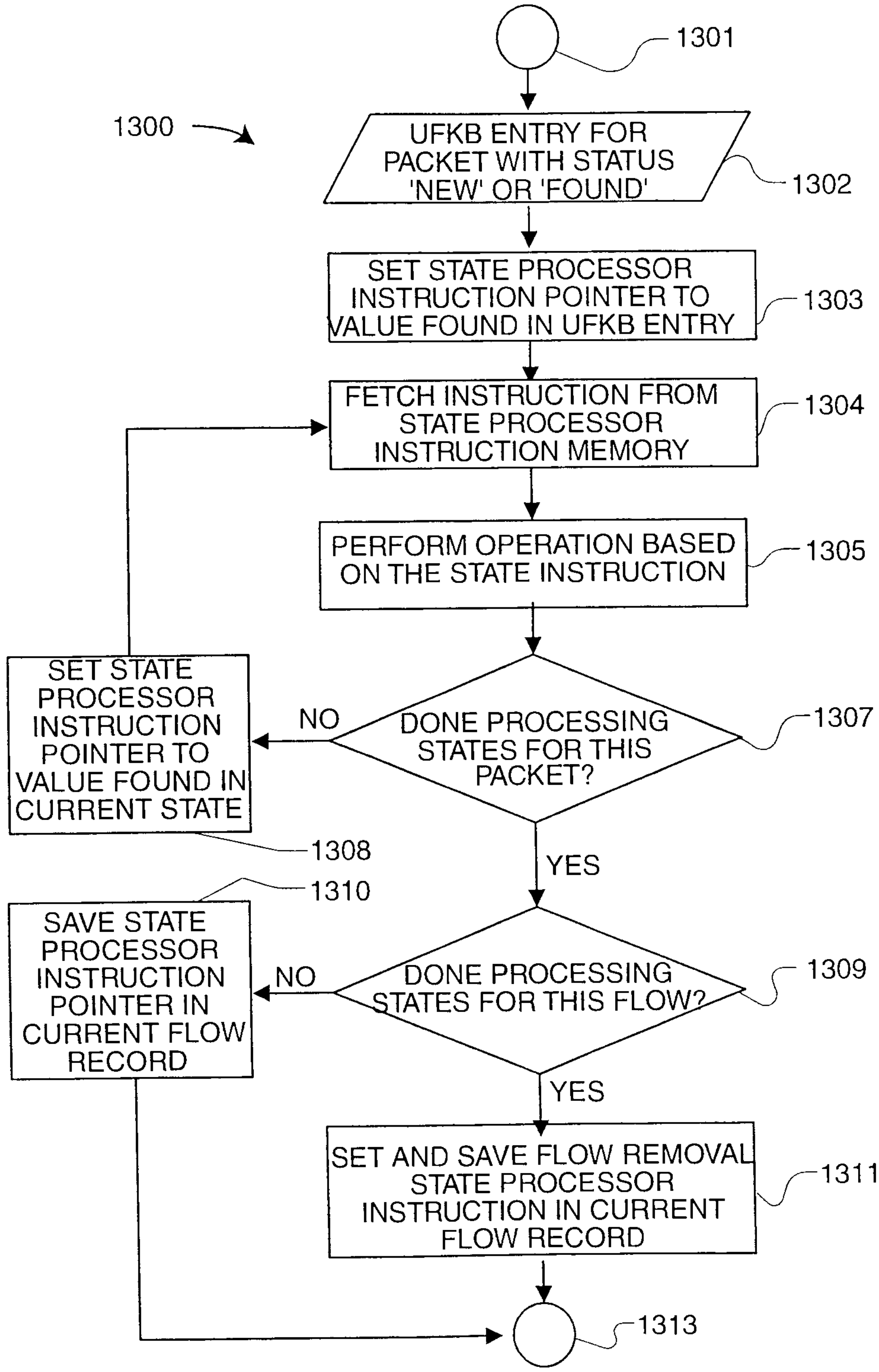

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307
NO
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
1308
YES
1310
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
NO
DONE PROCESSING STATES FOR THIS FLOW?
1309
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

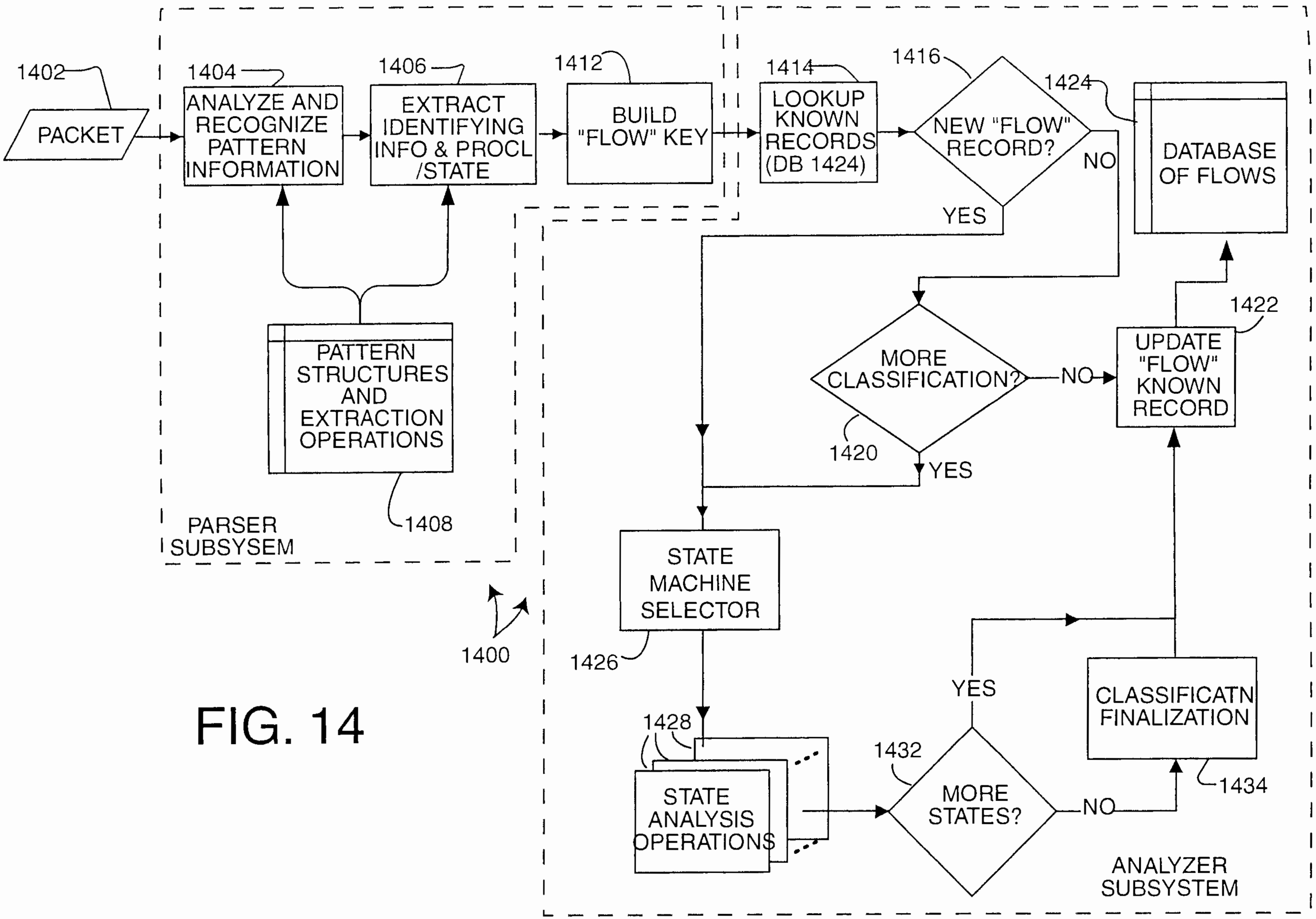

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1412
BUILD "FLOW" KEY
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
1424
DATABASE OF FLOWS
NO
YES
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
MORE CLASSIFICATION?
NO
1420
YES
UPDATE "FLOW" KNOWN RECORD
1422
STATE MACHINE SELECTOR
1426
1400
1428
STATE ANALYSIS OPERATIONS
1432
MORE STATES?
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM

FIG. 14

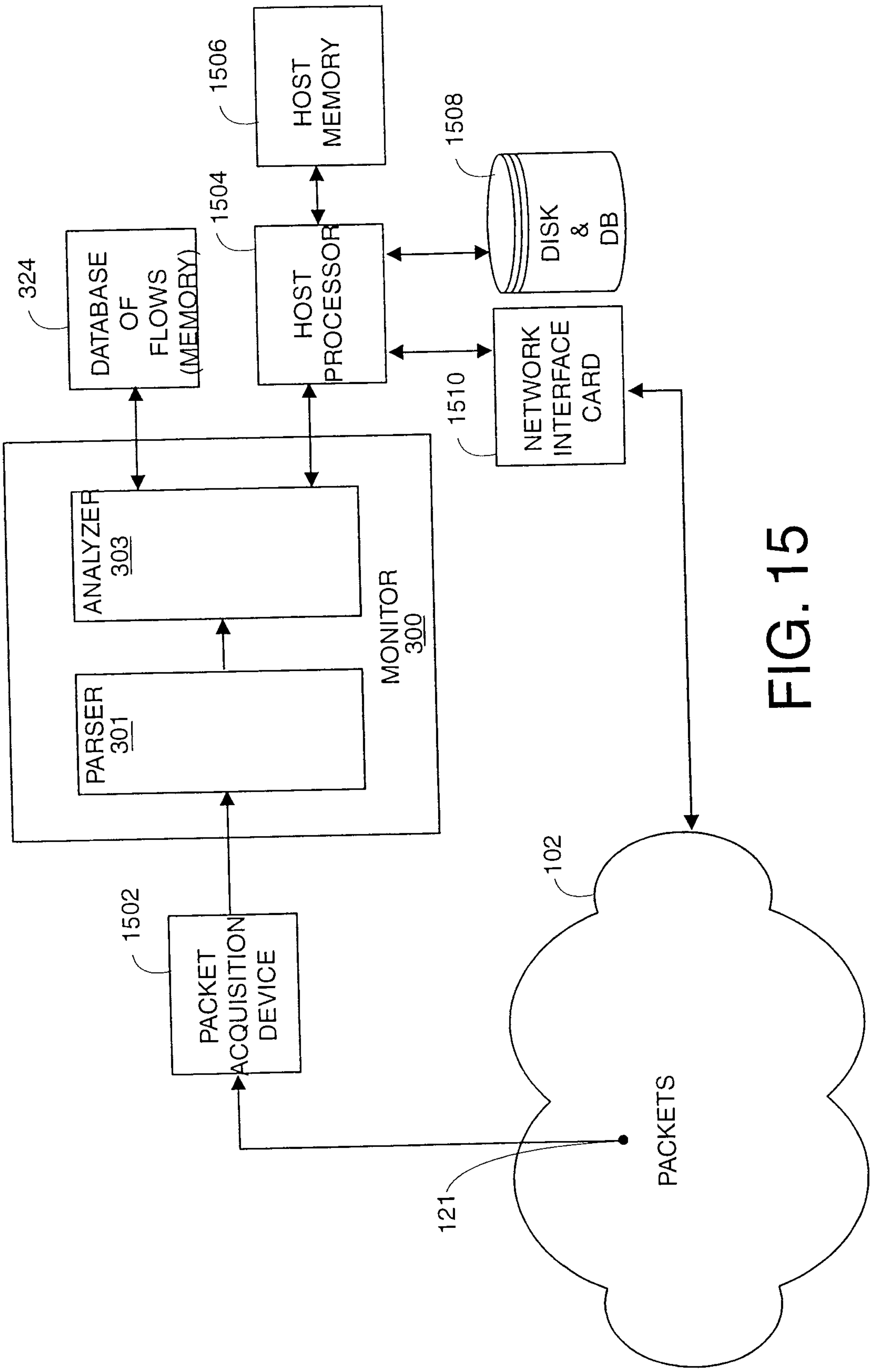

DATABASE OF FLOWS (MEMORY)
324
HOST MEMORY
1506
HOST PROCESSOR
1504
DISK & DB
1508
NETWORK INTERFACE CARD
1510
ANALYZER 303
PARSER 301
MONITOR 300
PACKET ACQUISITION DEVICE
1502
102
PACKETS
121

FIG. 15

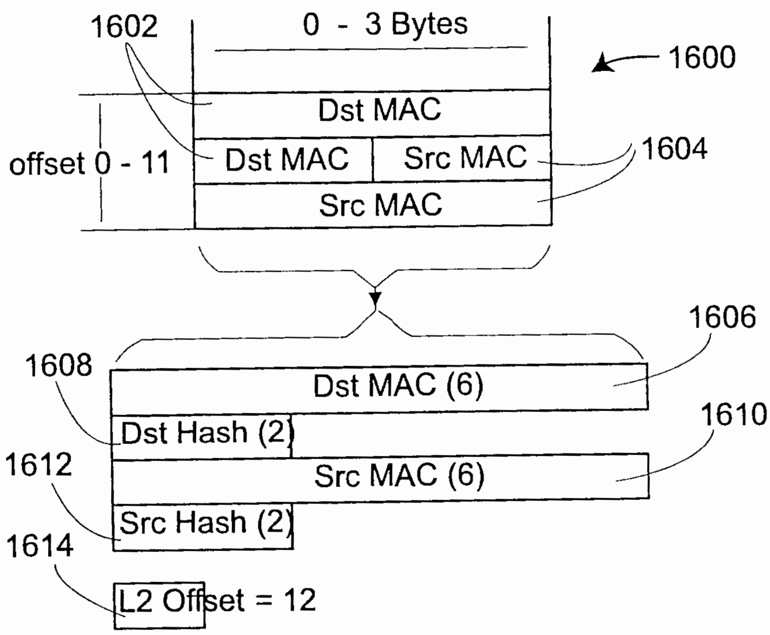

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

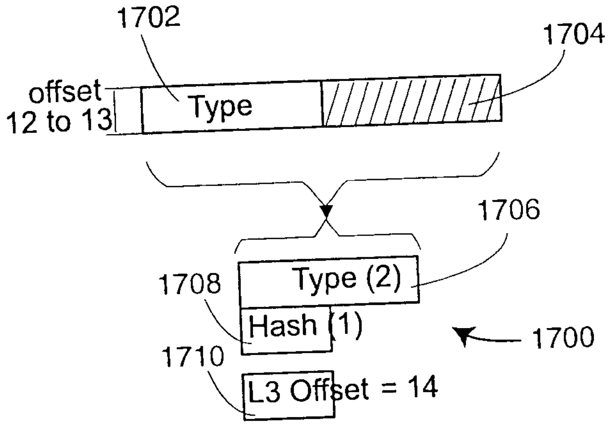


FIG. 17A

1712

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

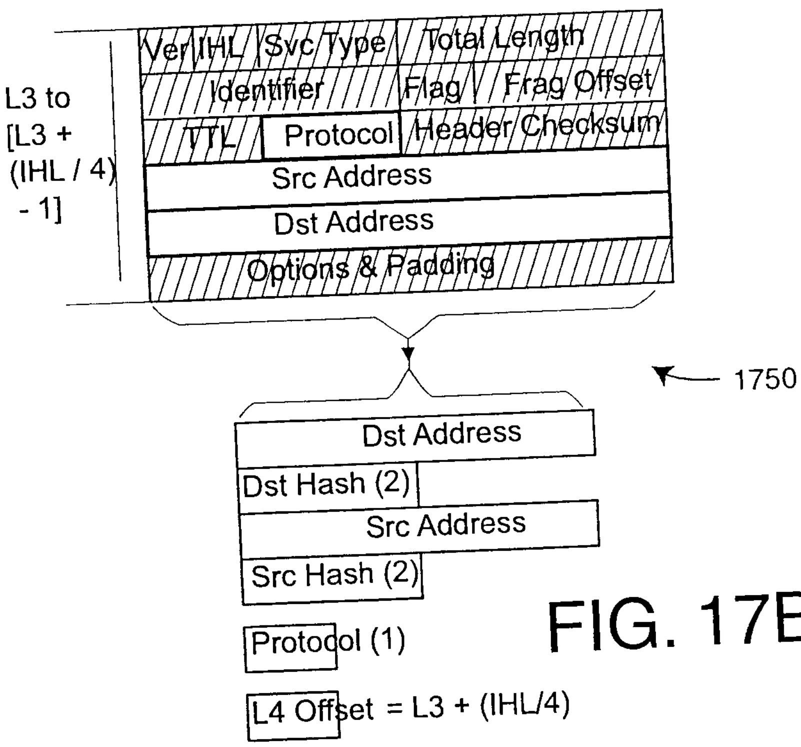


FIG. 17B

1752

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

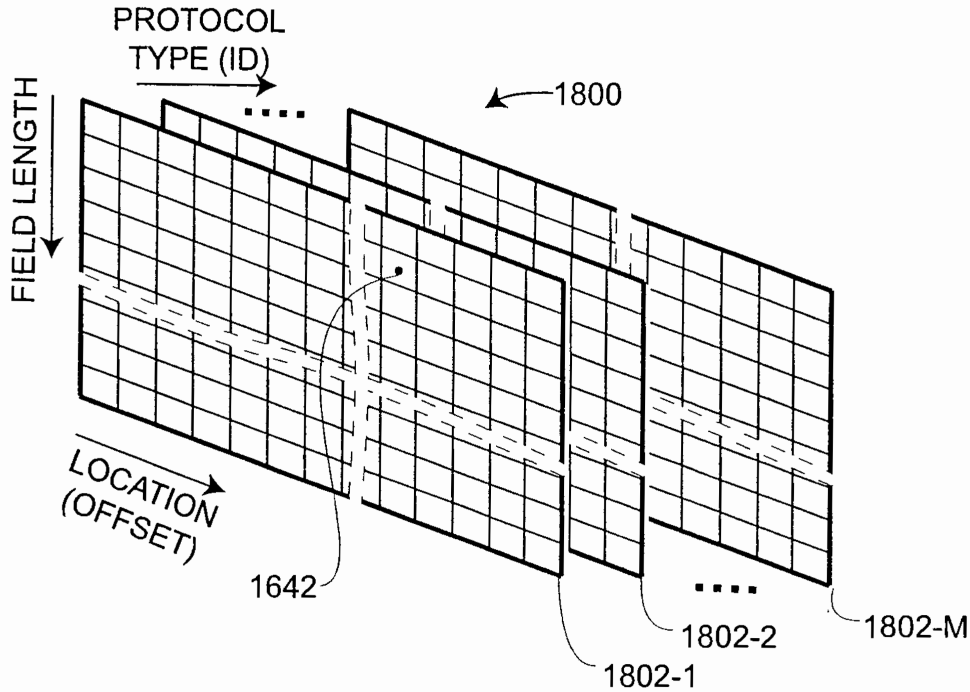


FIG. 18A

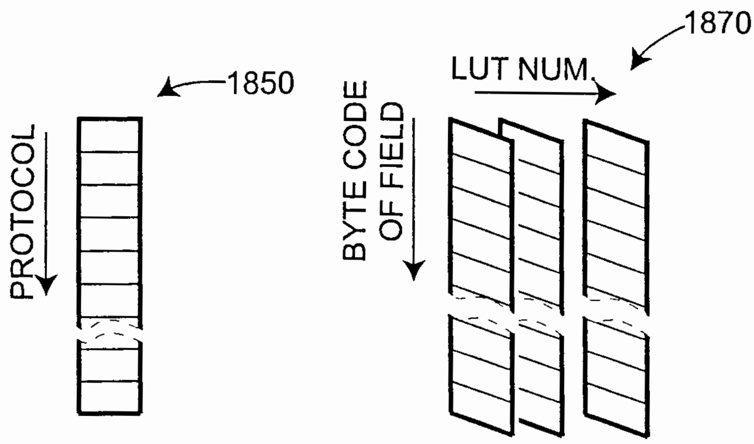


FIG. 18B

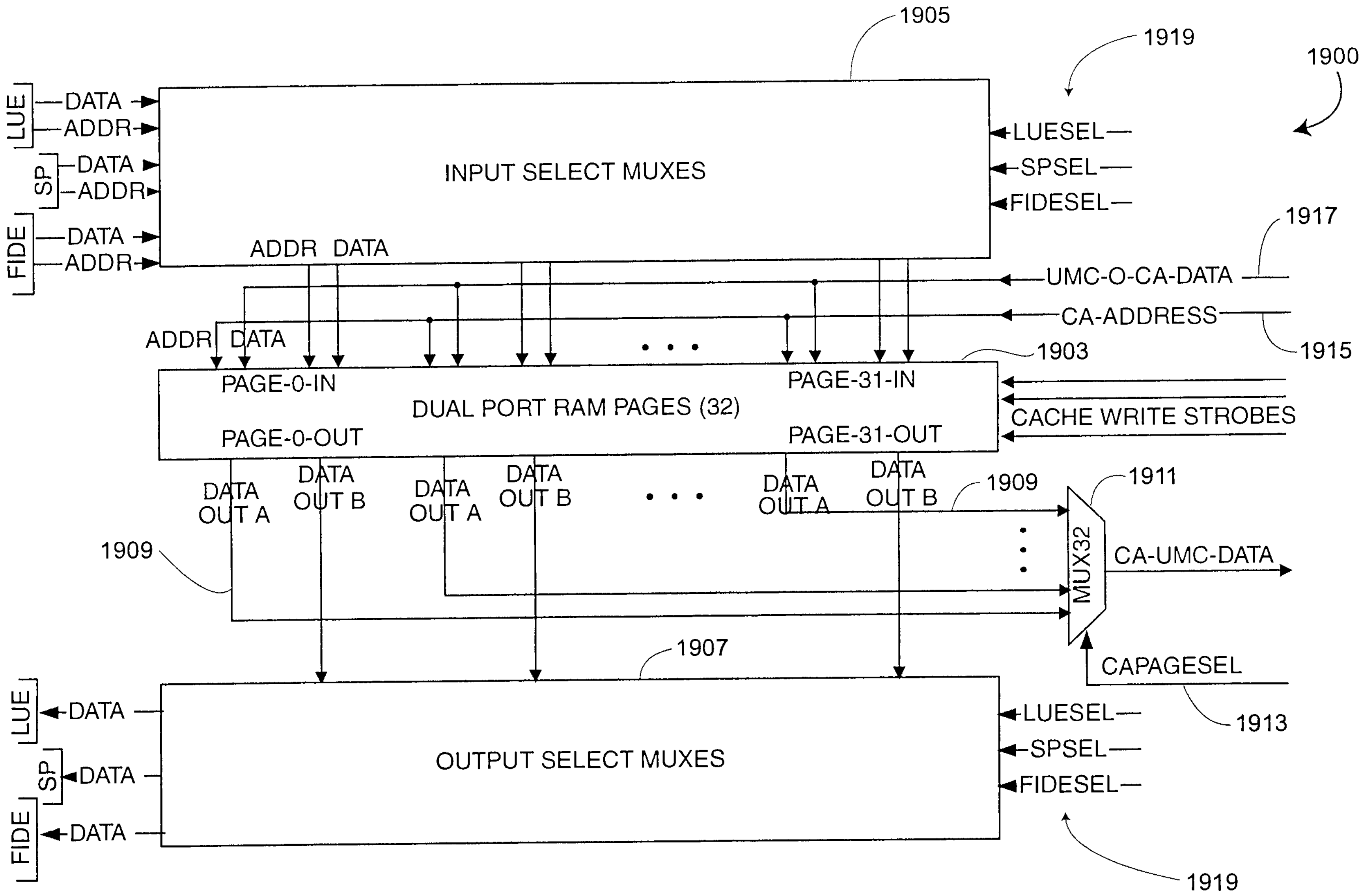

1905
1919
1900
LUE
DATA
ADDR
SP
DATA
ADDR
FIDE
DATA
ADDR
INPUT SELECT MUXES
LUESEL
SPSEL
FIDESEL
1917
ADDR DATA
UMC-O-CA-DATA
CA-ADDRESS
ADDR DATA
1903
1915
PAGE-0-IN
PAGE-31-IN
DUAL PORT RAM PAGES (32)
CACHE WRITE STROBES
PAGE-0-OUT
PAGE-31-OUT
DATA OUT A
DATA OUT B
DATA OUT A
DATA OUT B
DATA OUT A
DATA OUT B
1909
1911
1909
MUX32
CA-UMC-DATA
1907
CAPAGESEL
LUE
DATA
LUESEL
1913
SP
DATA
OUTPUT SELECT MUXES
SPSEL
FIDE
DATA
FIDESEL
1919

FIG. 19

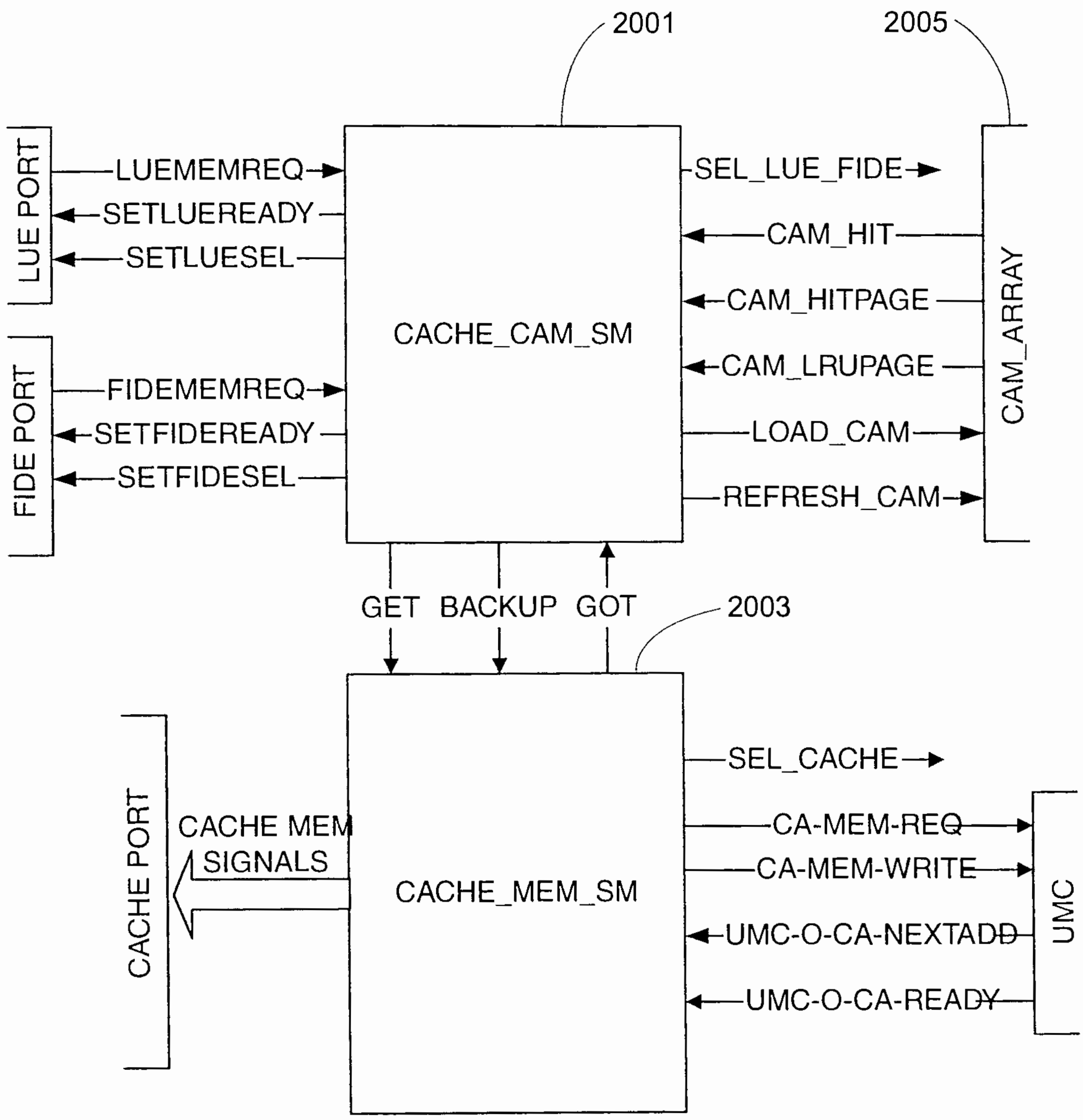

2001
2005
LUE PORT
LUEMEMREQ
SETLUEREADY
SETLUESEL
FIDE PORT
FIDEMEMREQ
SETFIDEREADY
SETFIDESEL
CACHE_CAM_SM
SEL_LUE_FIDE
CAM_HIT
CAM_HITPAGE
CAM_LRUPAGE
LOAD_CAM
REFRESH_CAM
CAM_ARRAY
GET
BACKUP
GOT
2003
CACHE PORT
CACHE MEM
SIGNALS
CACHE_MEM_SM
SEL_CACHE
CA-MEM-REQ
CA-MEM-WRITE
UMC-O-CA-NEXTADD
UMC-O-CA-READY
UMC

FIG. 20

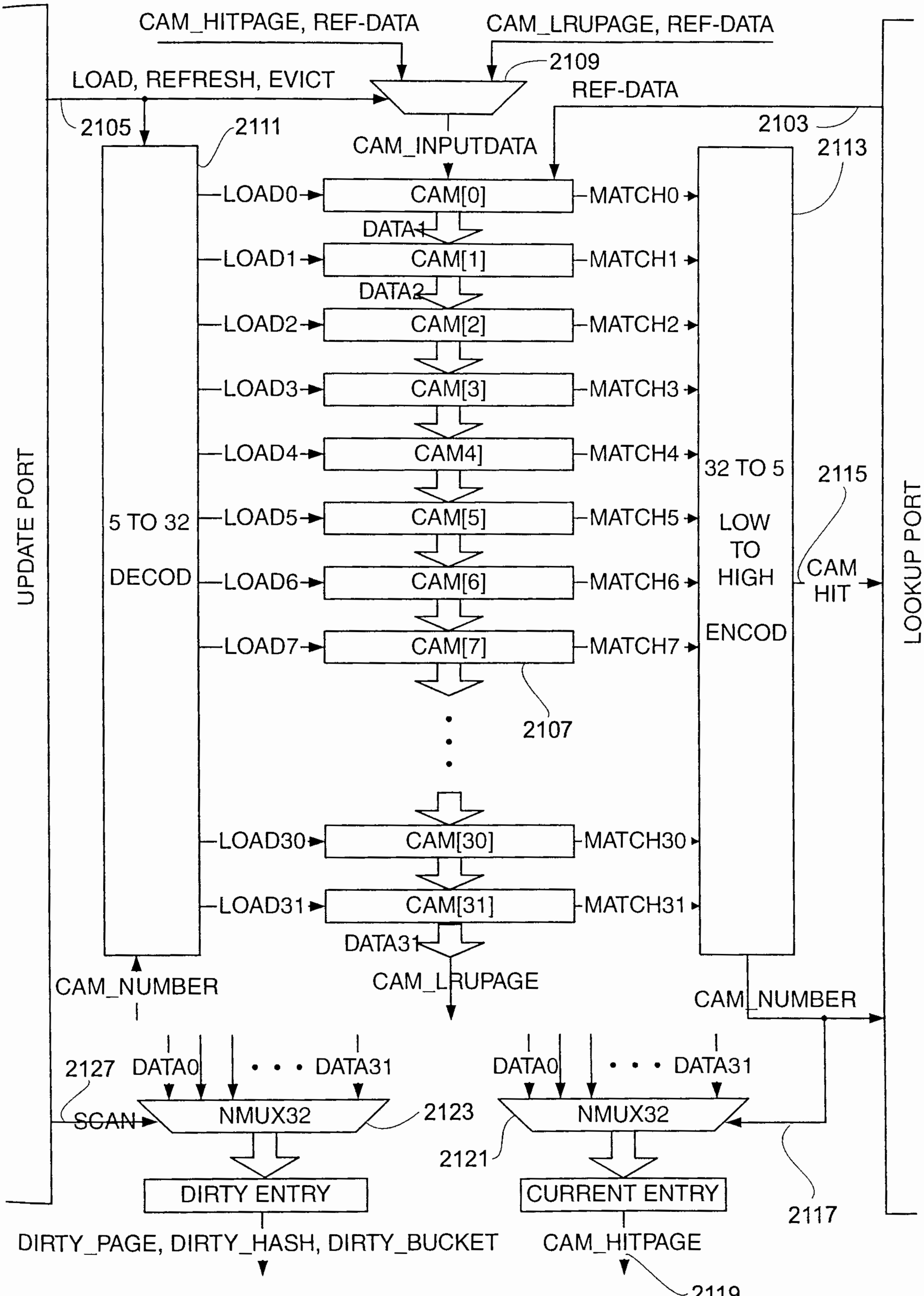

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
2109
REF-DATA
LOAD, REFRESH, EVICT
2105
2111
2103
2113
CAM_INPUTDATA
UPDATE PORT
5 TO 32
DECOD
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
LOAD30
LOAD31
CAM[0]
DATA1
CAM[1]
DATA2
CAM[2]
CAM[3]
CAM4]
CAM[5]
CAM[6]
CAM[7]
CAM[30]
CAM[31]
DATA31
MATCH0
MATCH1
MATCH2
MATCH3
MATCH4
MATCH5
MATCH6
MATCH7
MATCH30
MATCH31
32 TO 5
LOW
TO
HIGH
ENCOD
2115
CAM
HIT
LOOKUP PORT
2107
CAM_NUMBER
CAM_LRUPAGE
CAM_NUMBER
2127
DATA0
DATA31
SCAN
NMUX32
2123
DATA0
DATA31
NMUX32
2121
DIRTY ENTRY
CURRENT ENTRY
2117
DIRTY_PAGE, DIRTY_HASH, DIRTY_BUCKET
CAM_HITPAGE
2119

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

FORMALITIES LETTER

OC000000005373402

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

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

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

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	

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:

_X_ Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration;
_X_ Executed Assignment with assignment cover sheet.

_X_ A credit card payment form in the amount of $ 860.00 is attached, being for:
- _X_ Statutory basic filing fee: $ 690
- _X_ Additional claim fee of $ 0
- _X_ Assignment recordation fee of $ 40
- _X_ Missing Parts Surcharge $130

_X_ 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: [signature]

Name. Dov Rosenfeld, Reg. No 38687

_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

OIPE JC16 OCT 2 4 2000 PATENT & TRADEMARK OFFICE

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:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35
			YES: ____ NO: ____
			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:

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:

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

Send Correspondence to:	Direct Telephone Calls To:
Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618	Dov Rosenfeld, Reg. No. 38,687 Tel: (510) 547-3378

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

Haig A. Sarkissian

First Inventor's Signature

Sept 21, 2000

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 **Date** 10/9/00

OIPE OCT 24 2000 PATENT & TRADEMARK OFFICE

Our Ref./Docket No: APPT-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: Oct 20, 2000

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

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

FORMALITIES LETTER

OC000000005373402

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

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

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

Sector/# #3

OIPE OCT 24 2000 PATENT & TRADEMARK OFFICE

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	

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:

_X_ Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration;

_X_ Executed Assignment with assignment cover sheet.

_X_ A credit card payment form in the amount of $ 860.00 is attached, being for:

- _X_ Statutory basic filing fee: $ 690
- _X_ Additional claim fee of $ 0
- _X_ Assignment recordation fee of $ 40
- _X_ Missing Parts Surcharge $130

_X_ 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: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

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-0[illegible]-4 Patent 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:

OIPE APR 11 2001 JC33 PATENT & TRADEMARK OFFICE

RECEIVED APR 12 2001 TC 2600 MAILROOM

#4 4-12-01

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

_X_ 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.

_X_ Return postcard.

_X_ 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: Apr 9 2001

Signature:

Dov Rosenfeld, Reg. No. 38,687

Our Docket/Ref. No.: APPT-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: OIPE JC33 APR 11 2001 PATENT & TRADEMARK OFFICE RECEIVED APR 12 2001 TC 2600 MAILROOM

Commissioner for Patents
Washington, D.C. 20231

INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

This Information Disclosure Statement is submitted:

_X_ 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)

_X_ 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: Apr 9, 2001

Signature: [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.

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

Et al.FORM - 1449 SHEET 1 OF 5.

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~~ 2662

OIPE JC33 APR 11 2001 PATENT & TRADEMARK OFFICE

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
A~	AA	4736320	Apr. 5, 1988	Bristol	364	300	Oct. 8, 1985
A~	AB	4891639	Jan. 2, 1990	Nakamura	340	825.500	Jun. 23, 1988
A~	AC	5101402	Mar. 31, 1992	Chui et al.	~~370~~	~~17~~	May 24, 1988
A~	AD	5247517	Sep. 21, 1993	Ross et al.	370	85.5	Sep. 2, 1992
A~	AE	5247693	Sep. 21, 1993	Bristol	395	800	Nov. 17, 1992
A~	AF	5315580	May 24, 1994	Phaal	~~370~~	~~13~~	Aug. 26, 1991
A~	AG	5339268	Aug. 16, 1994	Machida	365	49	Nov. 24, 1992
A~	AH	5351243	Sep. 27, 1994	Kalkunte et. al.	~~370~~	~~92~~	Dec. 27, 1991
A~	AI	5365514	Nov. 15, 1994	Hershey et al.	~~370~~	~~17~~	Mar. 1, 1993
A~	AJ	5375070	Dec. 20, 1994	Hershey at al.	364	550	Mar. 1, 1993
A~	AK	5394394	Feb. 28, 1995	Crowther et al.	~~370~~	~~60~~	Jun. 24, 1993

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANS-LATION YES	NO
	AM							
	AN							

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

A~	AR	"Technical Note: the Narus System," Downloaded April 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.
	AS	

RECEIVED APR 12 2001 TC 2600 MAILROOM

EXAMINER	DATE CONSIDERED
[signature]	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~~ 2662

U.S. PATENT DOCUMENTS

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

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANS-LATION YES	NO
	BM							
	BN							

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

	BR	
	BS	

EXAMINER	DATE CONSIDERED
Al Yz	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

Et al. FORM - 1449 SHEET 3 OF 5.

APR 11 2001 JC33 PATENT & TRADEMARK OFFICE

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~~ 2662

U.S. PATENT DOCUMENTS

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

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANS-LATION YES	NO
	CM							
	CN							

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

	CR	
	CS	

RECEIVED APR 12 2001 TC 2600 MAILROOM

EXAMINER [signature]	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.

Et al.FORM - 1449 SHEET 4 OF 5.

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~~ 2662

OIPE APR 11 2001 PATENT & TRADEMARK OFFICE

U.S. PATENT DOCUMENTS

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

FOREIGN PATENT DOCUMENTS

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

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

	DR	
	DS	

RECEIVED APR 12 2001 TC 2600 MAILROOM

EXAMINER	DATE CONSIDERED
[signature]	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.	SERIAL NO.
APPT-001-4	09/608266
APPLICANT	
Sarkissian *et al.*	
FILING DATE	GROUP
6/30/2000	~~2731~~ 3762

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
AV	EA	5917821	Jun. 29, 1999	Gobuyan et al.	370	392	Aug. 16, 1996
AV	EB	5414704	May 9, 1995	Spinney	~~370~~	~~60~~	Apr. 5, 1994
AV	EC	6014380	Jan 11, 2000	Hendel et al.	370	392	Jun. 30, 1997
AV	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	TRANS-LATION YES	NO
	DM							
	DN							

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

	DR	
	DS	

EXAMINER	DATE CONSIDERED
Ale Jr	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.

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

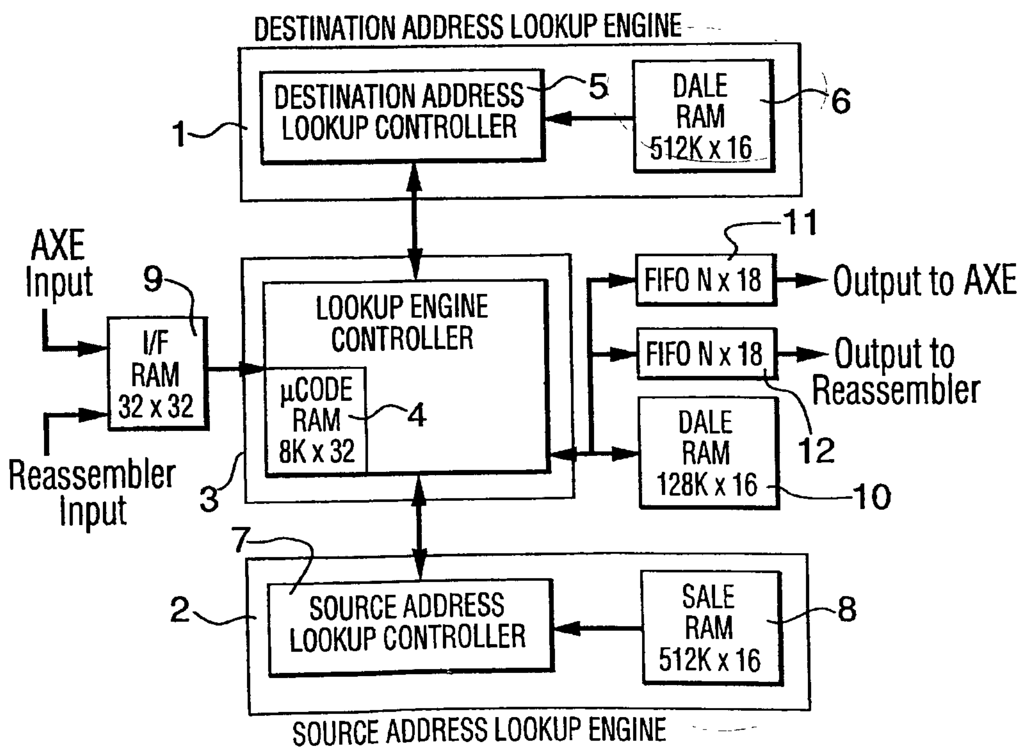
5,095,480 3/1992 Fenner 370/238
5,463,777 10/1995 Bialkowski et al. 370/256

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



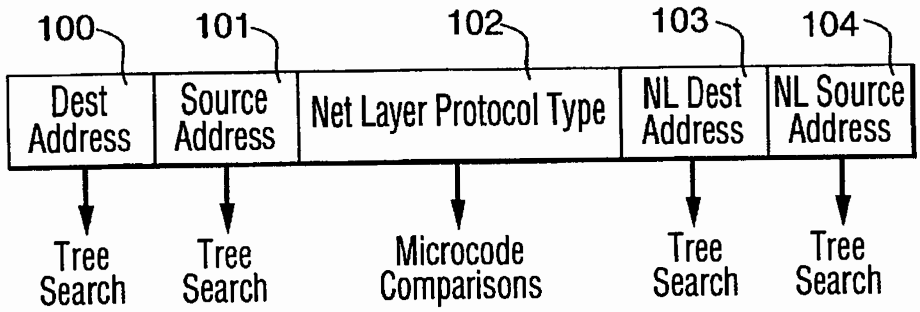


FIG. 1

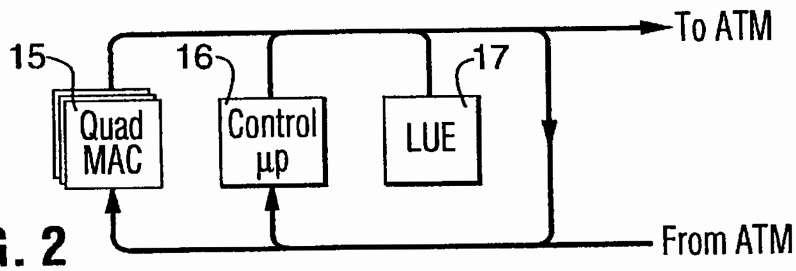


FIG. 2

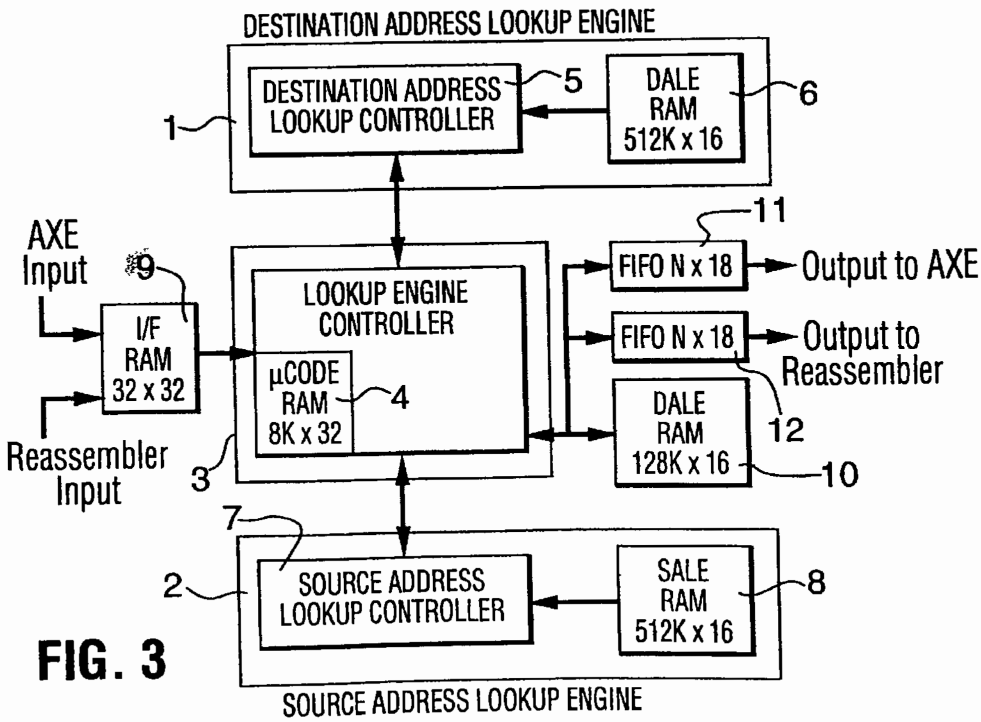
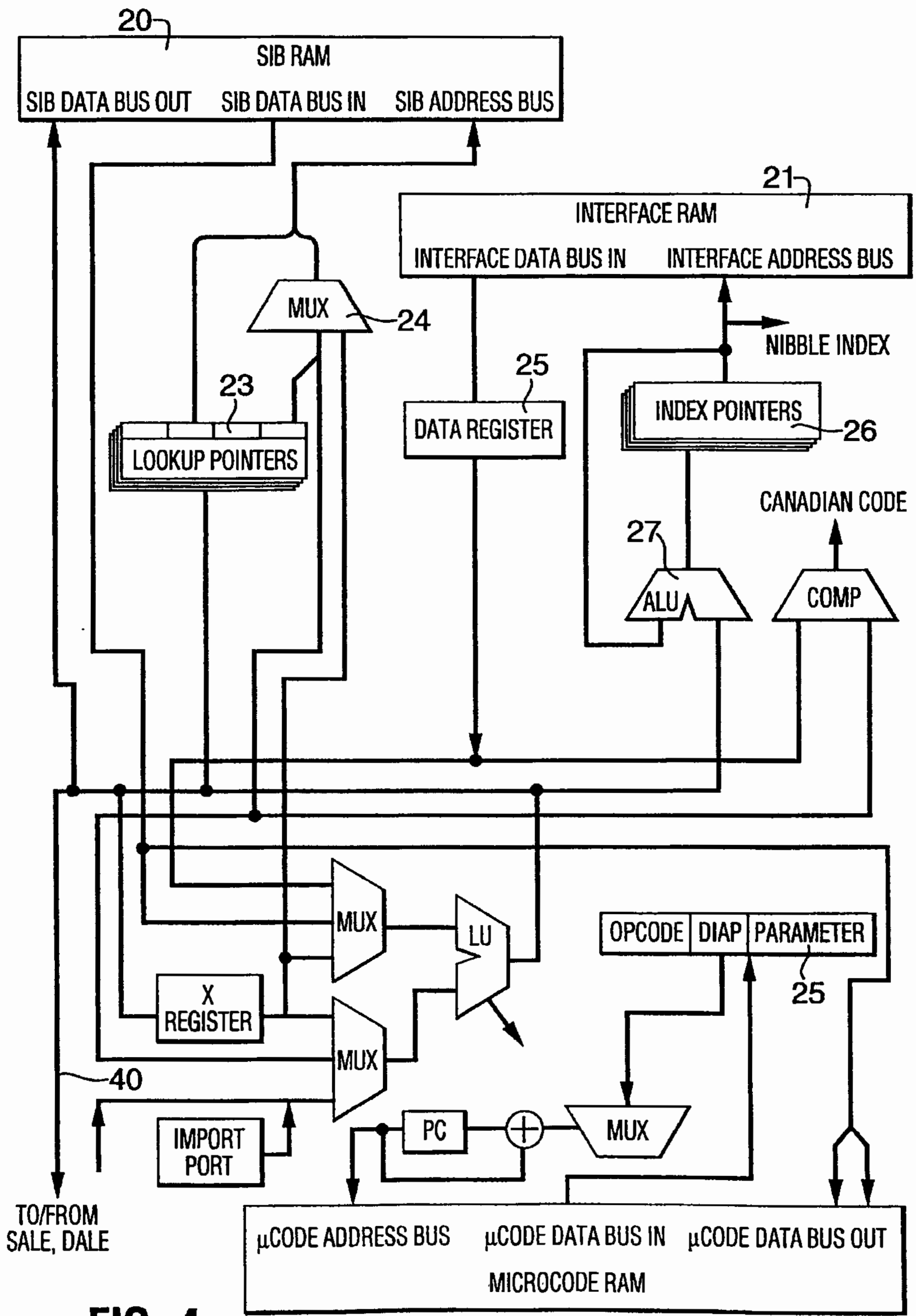


FIG. 3



20
SIB RAM
SIB DATA BUS OUT
SIB DATA BUS IN
SIB ADDRESS BUS
21
INTERFACE RAM
INTERFACE DATA BUS IN
INTERFACE ADDRESS BUS
MUX
24
NIBBLE INDEX
25
23
DATA REGISTER
INDEX POINTERS
26
LOOKUP POINTERS
CANADIAN CODE
27
ALU
COMP
MUX
LU
OPCODE
DIAP
PARAMETER
25
X REGISTER
MUX
40
IMPORT PORT
PC
MUX
TO/FROM SALE, DALE
µCODE ADDRESS BUS
µCODE DATA BUS IN
µCODE DATA BUS OUT
MICROCODE RAM

FIG. 4

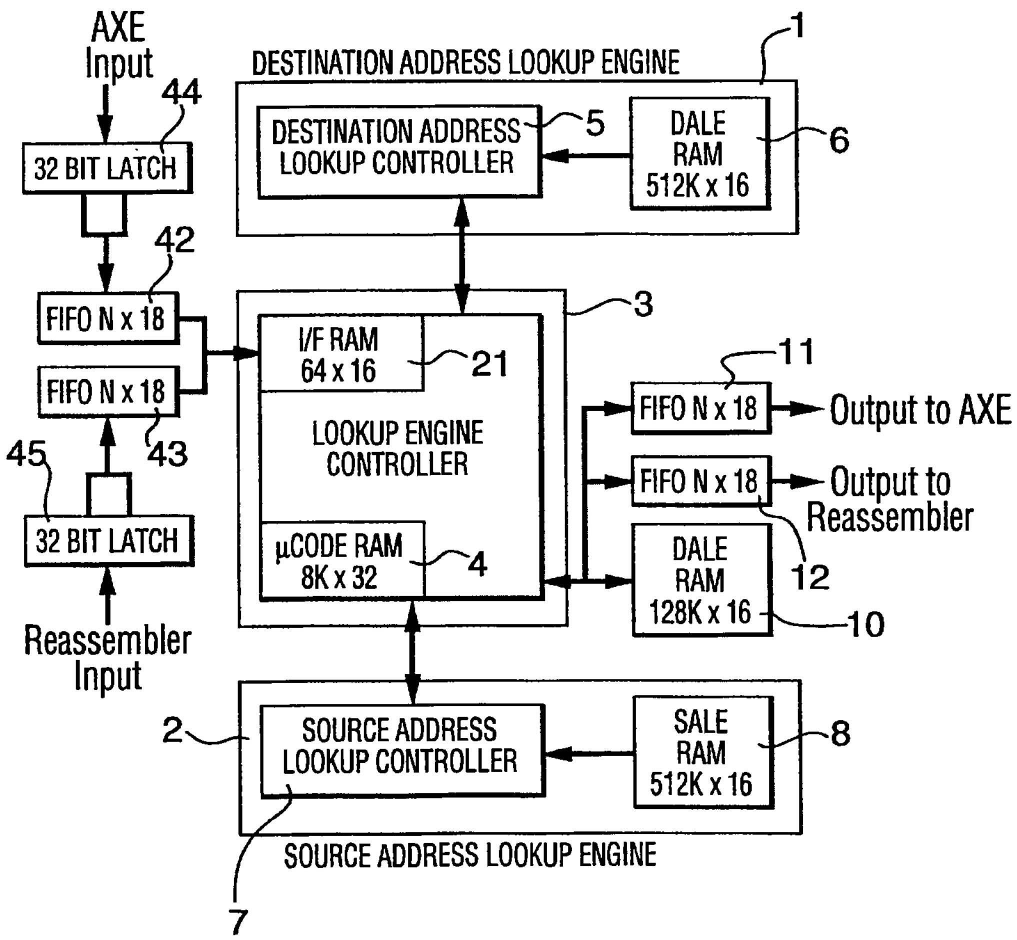
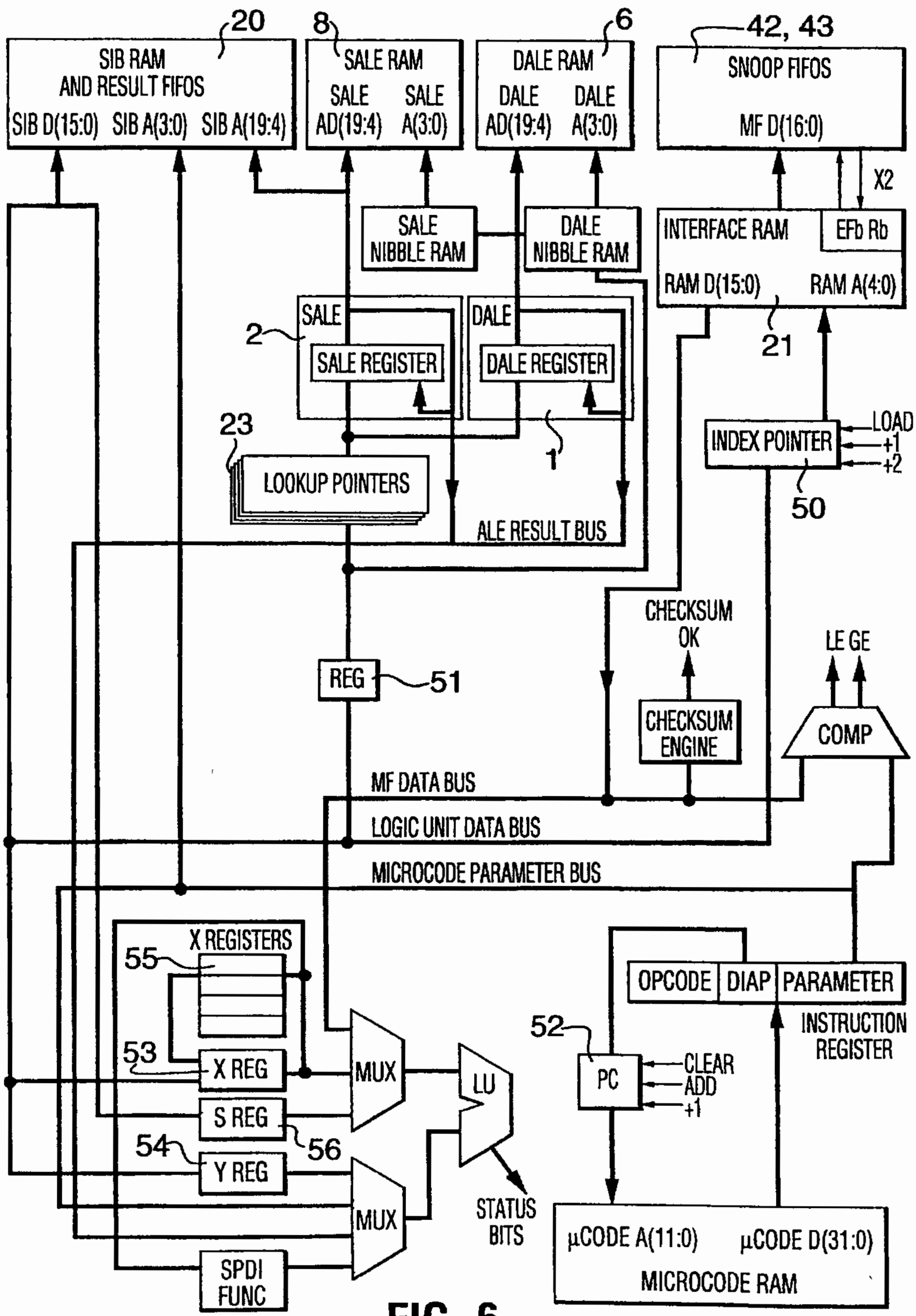

AXE Input
44
32 BIT LATCH
42
FIFO N x 18
FIFO N x 18
43
45
32 BIT LATCH
Reassembler Input
DESTINATION ADDRESS LOOKUP ENGINE
1
DESTINATION ADDRESS LOOKUP CONTROLLER
5
DALE RAM 512K x 16
6
3
I/F RAM 64 x 16
21
LOOKUP ENGINE CONTROLLER
µCODE RAM 8K x 32
4
11
FIFO N x 18
Output to AXE
FIFO N x 18
Output to Reassembler
12
DALE RAM 128K x 16
10
2
SOURCE ADDRESS LOOKUP CONTROLLER
SALE RAM 512K x 16
8
SOURCE ADDRESS LOOKUP ENGINE
7

FIG. 5



20
SIB RAM
AND RESULT FIFOS
SIB D(15:0) SIB A(3:0) SIB A(19:4)
8
SALE RAM
SALE AD(19:4)
SALE A(3:0)
6
DALE RAM
DALE AD(19:4)
DALE A(3:0)
42, 43
SNOOP FIFOS
MF D(16:0)
X2
SALE NIBBLE RAM
DALE NIBBLE RAM
INTERFACE RAM
EFb Rb
RAM D(15:0)
RAM A(4:0)
21
2
SALE
SALE REGISTER
DALE
DALE REGISTER
1
23
LOOKUP POINTERS
ALE RESULT BUS
INDEX POINTER
LOAD
+1
+2
50
CHECKSUM OK
CHECKSUM ENGINE
LE GE
COMP
REG
51
MF DATA BUS
LOGIC UNIT DATA BUS
MICROCODE PARAMETER BUS
X REGISTERS
55
53
X REG
S REG
56
54
Y REG
SPDI FUNC
MUX
MUX
LU
STATUS BITS
52
PC
CLEAR
ADD
+1
OPCODE
DIAP
PARAMETER
INSTRUCTION REGISTER
µCODE A(11:0)
µCODE D(31:0)
MICROCODE RAM

FIG. 6

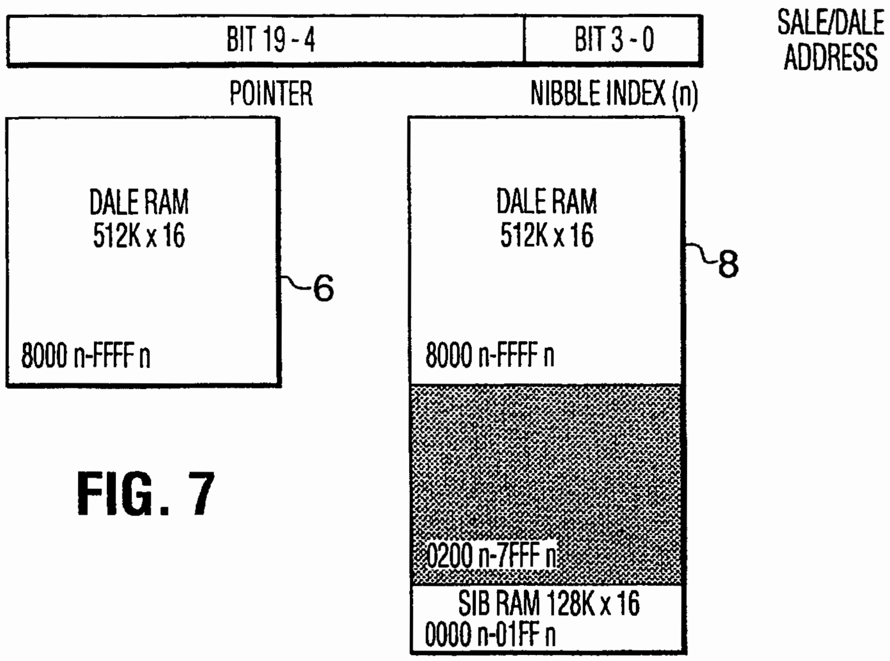

BIT 19 - 4
BIT 3 - 0
SALE/DALE ADDRESS
POINTER
NIBBLE INDEX (n)
DALE RAM 512K x 16
8000 n-FFFF n
6
DALE RAM 512K x 16
8000 n-FFFF n
8
0200 n-7FFF n
SIB RAM 128K x 16
0000 n-01FF n

FIG. 7

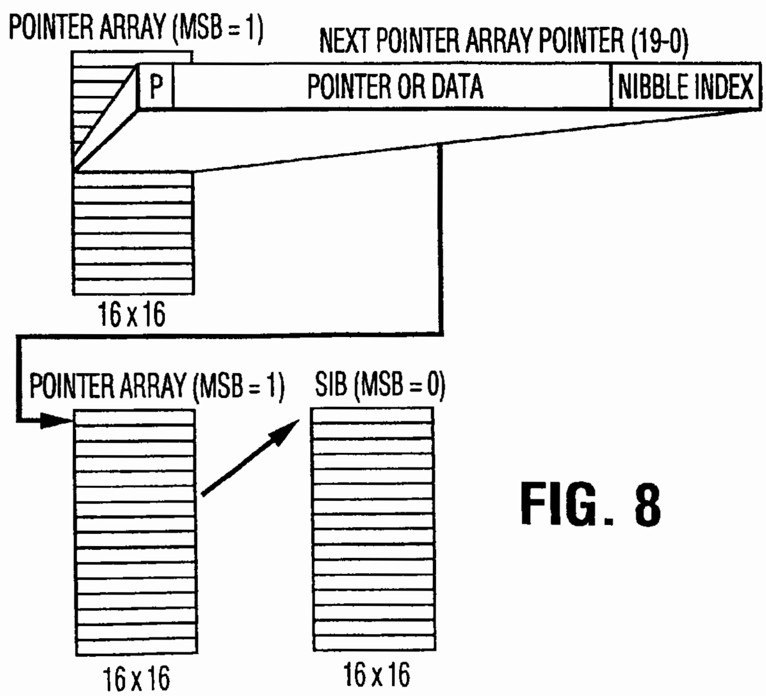

POINTER ARRAY (MSB = 1)
NEXT POINTER ARRAY POINTER (19-0)
P
POINTER OR DATA
NIBBLE INDEX
16 x 16
POINTER ARRAY (MSB = 1)
SIB (MSB = 0)
16 x 16
16 x 16

FIG. 8

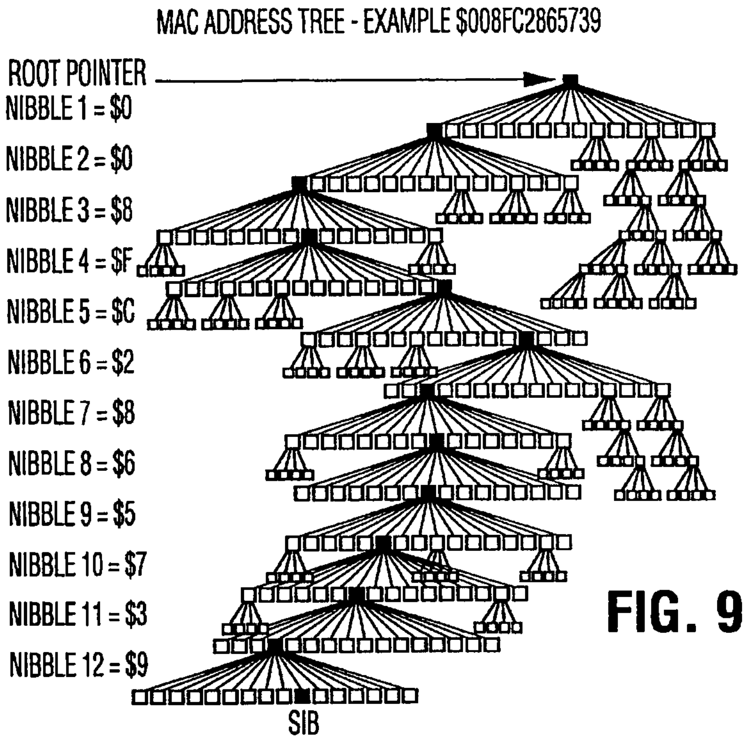

MAC ADDRESS TREE - EXAMPLE $008FC2865739
ROOT POINTER
NIBBLE 1 = $0
NIBBLE 2 = $0
NIBBLE 3 = $8
NIBBLE 4 = $F
NIBBLE 5 = $C
NIBBLE 6 = $2
NIBBLE 7 = $8
NIBBLE 8 = $6
NIBBLE 9 = $5
NIBBLE 10 = $7
NIBBLE 11 = $3
NIBBLE 12 = $9
SIB

FIG. 9

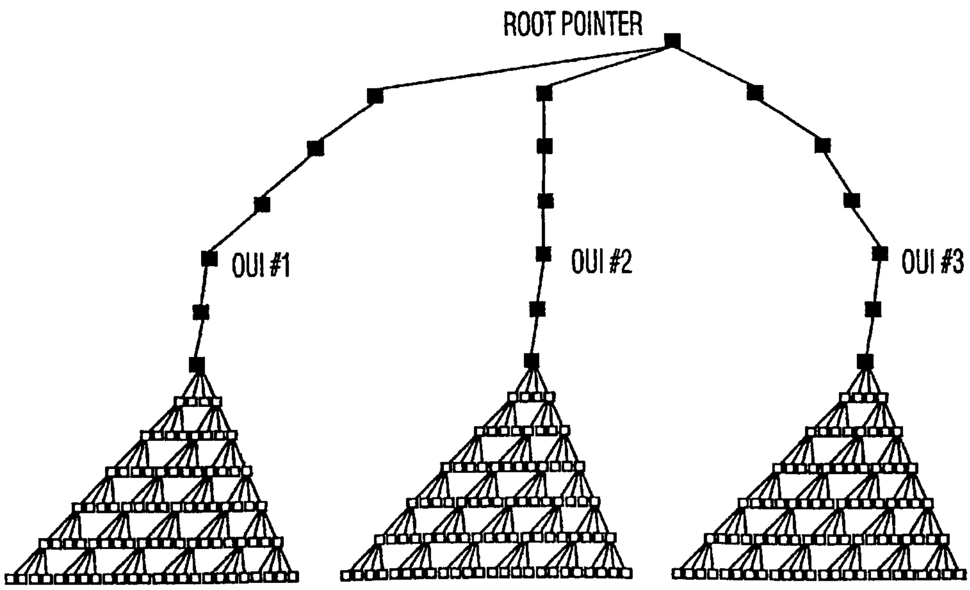

ROOT POINTER
OUI #1
OUI #2
OUI #3

FIG. 10

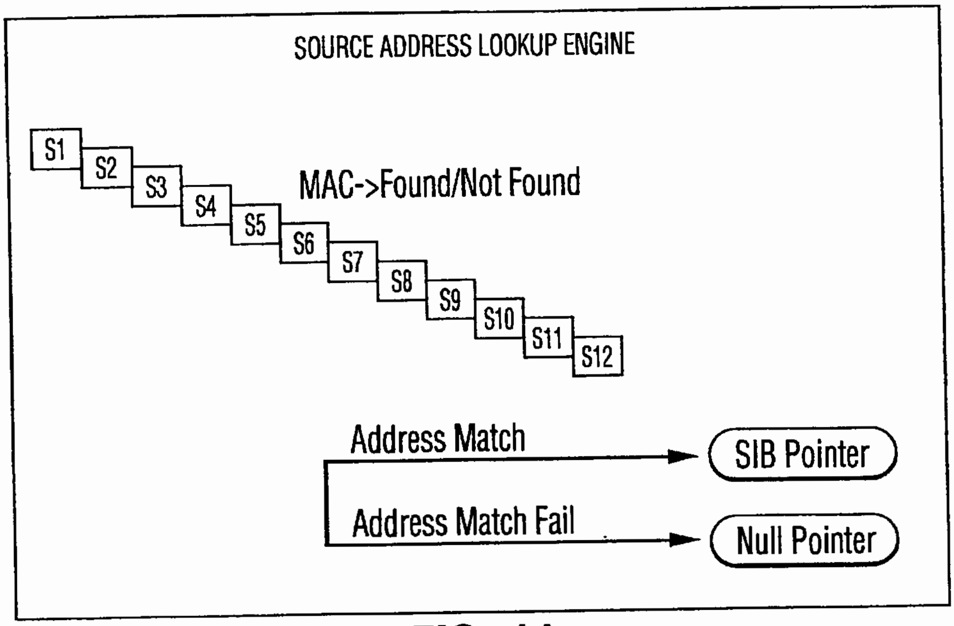

SOURCE ADDRESS LOOKUP ENGINE
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
MAC->Found/Not Found
Address Match
SIB Pointer
Address Match Fail
Null Pointer

FIG. 11

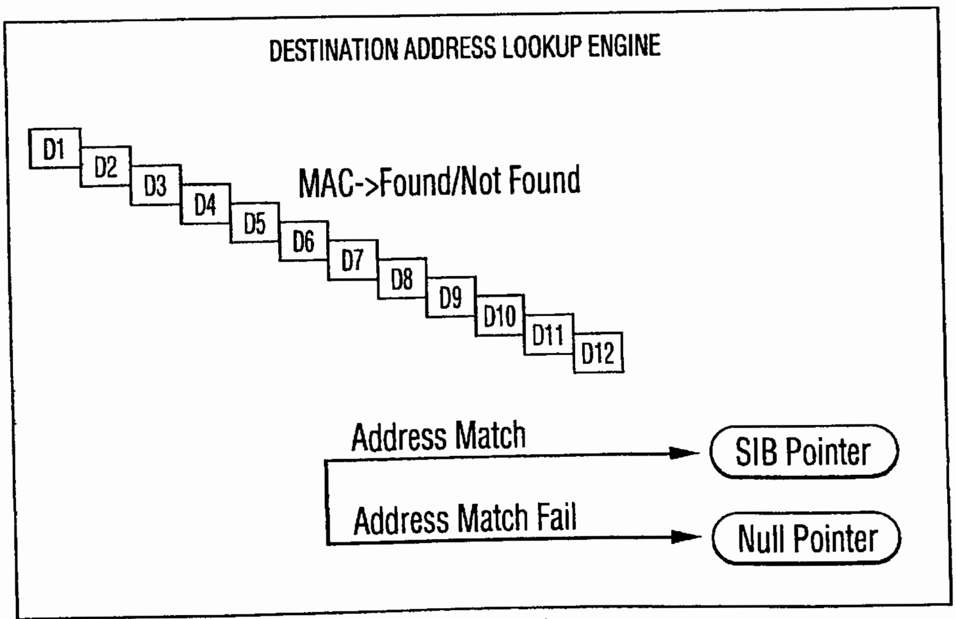

DESTINATION ADDRESS LOOKUP ENGINE
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
MAC->Found/Not Found
Address Match
SIB Pointer
Address Match Fail
Null Pointer

FIG. 12

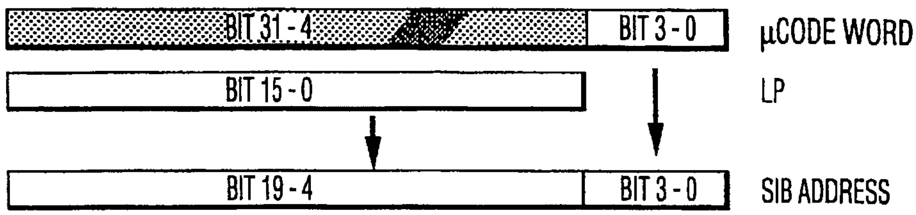

BIT 31 - 4
BIT 3 - 0
μCODE WORD
BIT 15 - 0
LP
BIT 19 - 4
BIT 3 - 0
SIB ADDRESS

FIG. 13

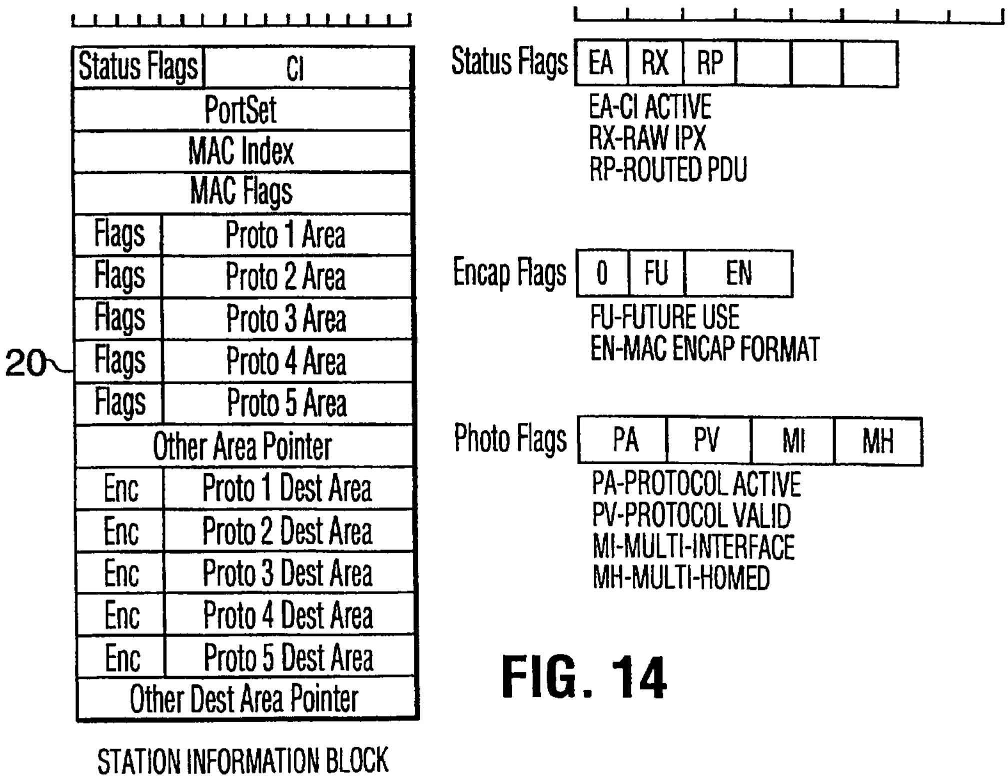

Status Flags
CI
PortSet
MAC Index
MAC Flags
Flags
Proto 1 Area
Flags
Proto 2 Area
Flags
Proto 3 Area
20
Flags
Proto 4 Area
Flags
Proto 5 Area
Other Area Pointer
Enc
Proto 1 Dest Area
Enc
Proto 2 Dest Area
Enc
Proto 3 Dest Area
Enc
Proto 4 Dest Area
Enc
Proto 5 Dest Area
Other Dest Area Pointer
STATION INFORMATION BLOCK
Status Flags
EA
RX
RP
EA-CI ACTIVE
RX-RAW IPX
RP-ROUTED PDU
Encap Flags
0
FU
EN
FU-FUTURE USE
EN-MAC ENCAP FORMAT
Photo Flags
PA
PV
MI
MH
PA-PROTOCOL ACTIVE
PV-PROTOCOL VALID
MI-MULTI-INTERFACE
MH-MULTI-HOMED

FIG. 14

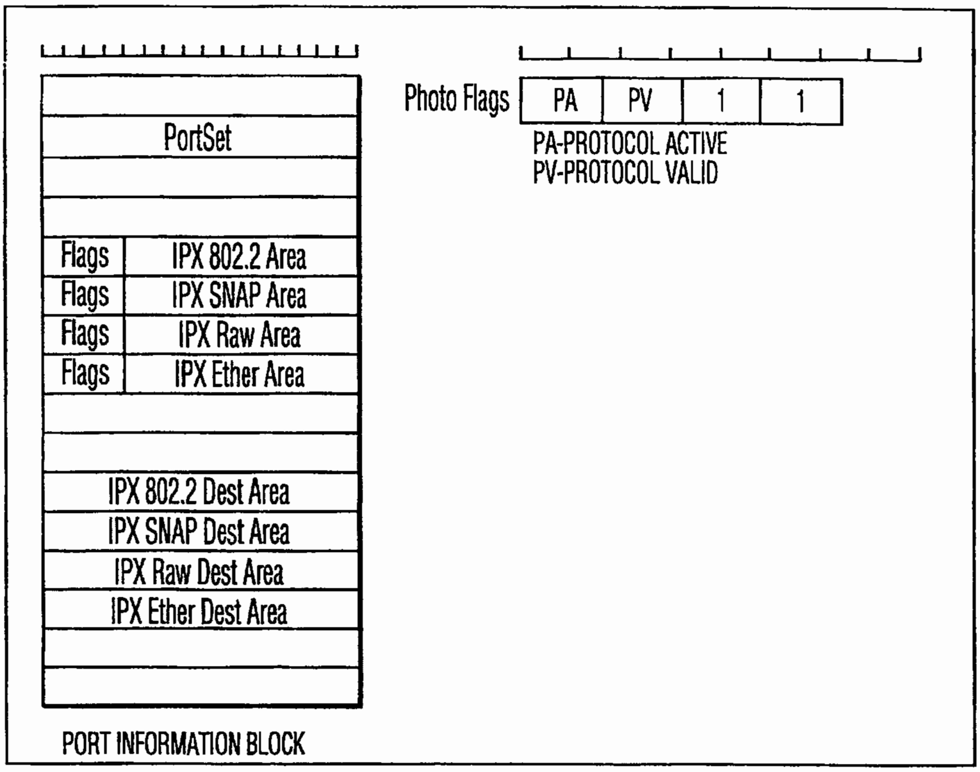

PortSet
Flags
IPX 802.2 Area
Flags
IPX SNAP Area
Flags
IPX Raw Area
Flags
IPX Ether Area
IPX 802.2 Dest Area
IPX SNAP Dest Area
IPX Raw Dest Area
IPX Ether Dest Area
PORT INFORMATION BLOCK
Photo Flags
PA
PV
1
1
PA-PROTOCOL ACTIVE
PV-PROTOCOL VALID

FIG. 15

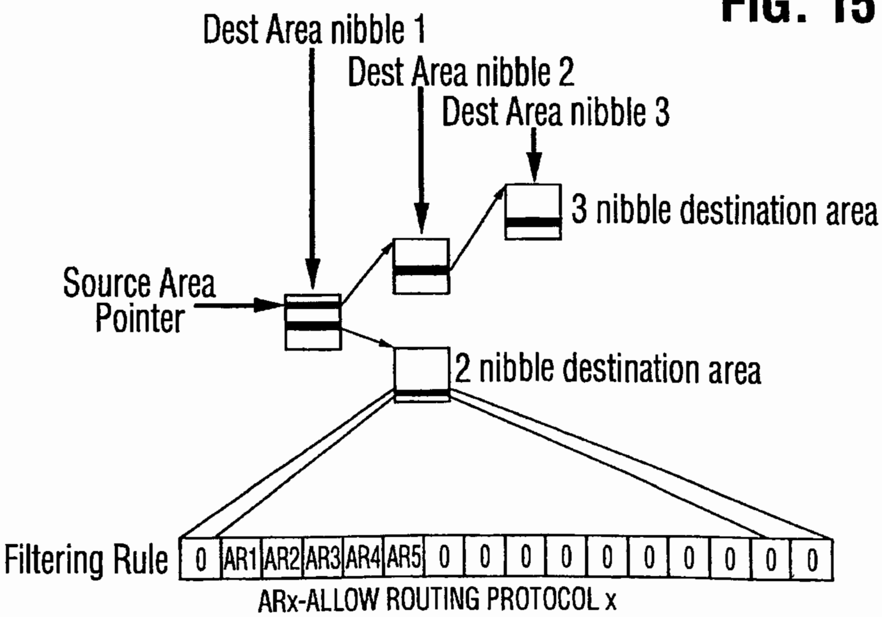

Dest Area nibble 1
Dest Area nibble 2
Dest Area nibble 3
3 nibble destination area
Source Area Pointer
2 nibble destination area
Filtering Rule
0 AR1 AR2 AR3 AR4 AR5 0 0 0 0 0 0 0 0 0 0
ARx-ALLOW ROUTING PROTOCOL x

FIG. 16

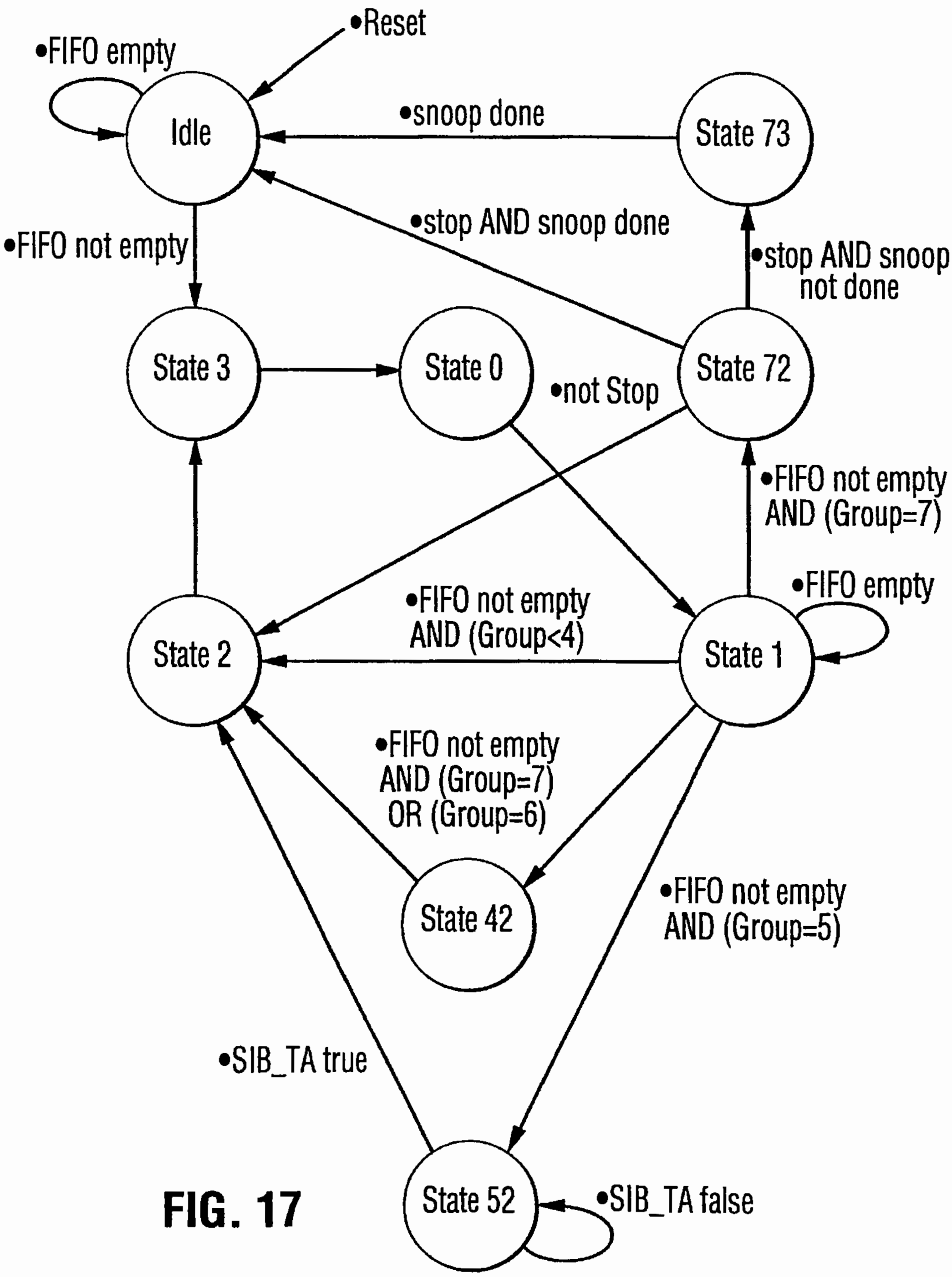

•Reset
•FIFO empty
Idle
•snoop done
State 73
•stop AND snoop done
•FIFO not empty
•stop AND snoop not done
State 3
State 0
•not Stop
State 72
•FIFO not empty AND (Group=7)
•FIFO empty
•FIFO not empty AND (Group<4)
State 2
State 1
•FIFO not empty AND (Group=7) OR (Group=6)
State 42
•FIFO not empty AND (Group=5)
•SIB_TA true
State 52
•SIB_TA false

FIG. 17

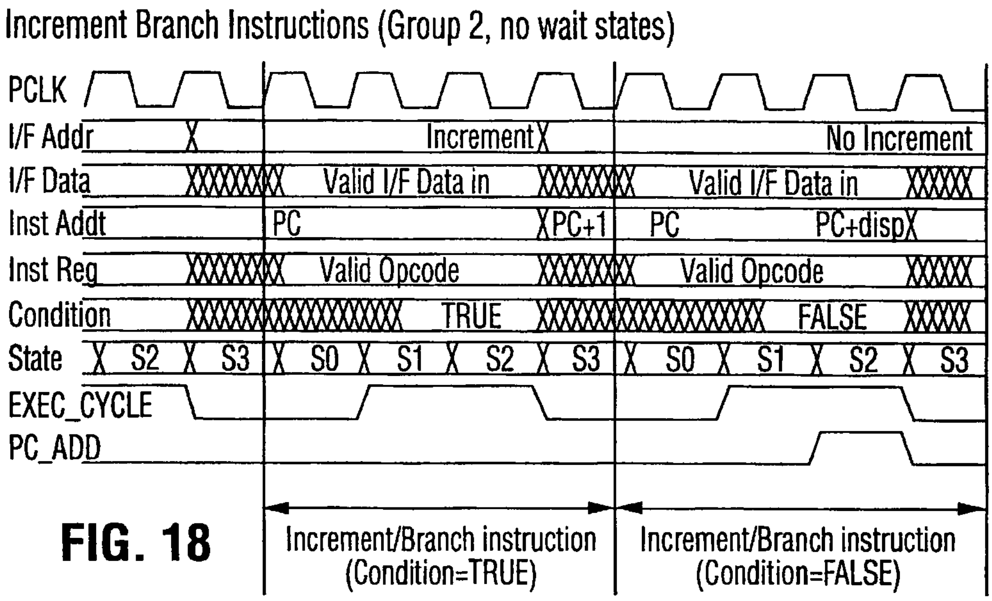

Increment Branch Instructions (Group 2, no wait states)
PCLK
I/F Addr
I/F Data
Inst Addt
Inst Reg
Condition
State
EXEC_CYCLE
PC_ADD
Increment
No Increment
Valid I/F Data in
Valid I/F Data in
PC
PC+1
PC
PC+disp
Valid Opcode
Valid Opcode
TRUE
FALSE
S2
S3
S0
S1
S2
S3
S0
S1
S2
S3
Increment/Branch instruction (Condition=TRUE)
Increment/Branch instruction (Condition=FALSE)

FIG. 18

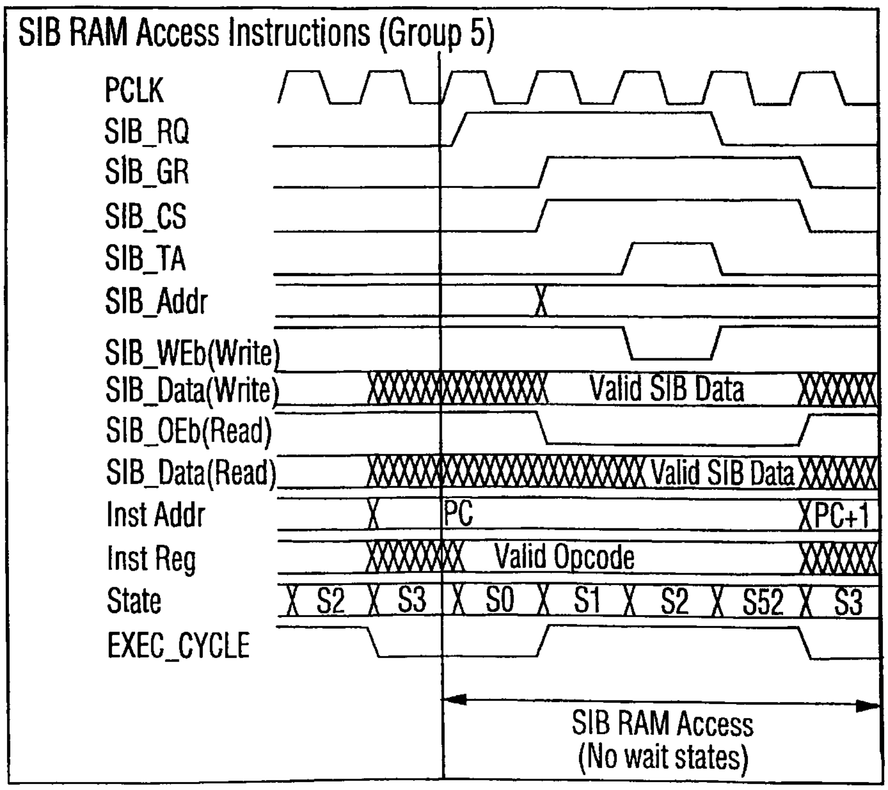

SIB RAM Access Instructions (Group 5)
PCLK
SIB_RQ
SIB_GR
SIB_CS
SIB_TA
SIB_Addr
SIB_WEb(Write)
SIB_Data(Write)
SIB_OEb(Read)
SIB_Data(Read)
Inst Addr
Inst Reg
State
EXEC_CYCLE
Valid SIB Data
Valid SIB Data
PC
PC+1
Valid Opcode
S2
S3
S0
S1
S2
S52
S3
SIB RAM Access (No wait states)

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 information 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 μsecs. 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.

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 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 ATM 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.

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

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 ihstruction		
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:		

-continued

Condition		
010 - DALE Lookup Result		
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):		
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)
$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):		
X(11:8)	f(15:0)	
0	0000000000000001	
1	0000000000000010	
2	0000000000000100	
3	0000000000001000	
4	0000000000010000	
5	0000000000100000	
6	0000000001000000	
7	0000000010000000	
8	0000000100000000	
9	0000001000000000	
10	0000010000000000	
11	0000100000000000	
12	0001000000000000	
13	0010000000000000	
14	0100000000000000	
15	1000000000000000	

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)

0ee 01	MOVEF #Value,Extra Move Immediate Value to FIFO with Extra bits
0ee 00	MOVEF Y,Extra Move Y Register to FIFO with Extra bits
1ee 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 Y -> X
100 111 001 00	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

-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 aaa OR bb -> dst
dst 010 aaa bb	ANDN aaa,bb,dst aaa OR bb -> dst
dst 011 aaa bb	ORN aaa,bb,dst aaa OR 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:                                  ; check if SNAP header
        ESCNE. w    #$AAAA,Label2        ; compare to SNAP value
        ESCNE.w     #$0003,OtherLabel
        . . .
Label2:
```

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

-continued

Cond/LSel	
110 100	BXEQ #Value,Label Branch if X is equal to value
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	Label Increment Byte and Branch Always
011 10 ddd $000 d IBRA.w	Label 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 Label	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 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 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 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
                     CheckEnetType
     ESCNE.w         $AAAA,            ;check DSAP/SSAP
                     UnknownType
     ESCNE.w         $0300,            ;check CTL field
                     SNAPUnknown-
                     Type
     ESCNE.w         $0000,
                     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_
                     withOpts
     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
     SWITCH          ;check IP Class
     BRA.u           BdgSNAPIP-        ;0xxx = Class A address
                     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-        ;10xx = Class B address
                     ClassB
     BRA.u           BdgSNAPIP-
                     ClassB
     BRA.u           BdgSNAPIP-
                     ClassB
     BRA.u           BdgSNAPIP-
                     ClassB
```

-continued

```
    BRA.u        BdgSNAPIP-      ;110x = Class C address
                 ClassC
    BRA.u        BdgSNAPIP-
                 ClassC
    BRA.u        BdgSNAPIP-      ;1110 = Class D address
                 ClassD
    BRA.u        BdgSNAPIP-      ;1111 = Class E address (future)
                 ClassE
    BRA.u        BdgSNAPIP-      ;0xxx = Class A Address
                 ClassA
BdgSNAPIPClassA:
    OR           X,$FF00,X       ;check if broadcast
    BXNE         $FFFF,
                 BdgSNAPIP_
                 NLSARealign
    MOVE         (IP)+,X         ;check lower address word
    BXEQ         $FFFF,          ;all ones host address
                 BdgSNAPIP_
                 NLSAInvalid
    BRA.u        BdgSNAPIP_      ;broadcast SA is not allowed
                 NLSAValid
BdgSNAPIP_
NLSARealign:
    SKIP.w
    BRA.u        BdgSNAPIP_
                 NLSAValid
BdgSNAPIPClassB:
    MOVE         (IP)+,X         ;check lower address word
    BXNE         $FFFF,
                 BdgSNAPIP_
                 NLSAValid
    BRA.u        BdgSNAPIP_
                 NLSAValid
BdgSNAPIPClassC:
    MOVE         (IP)+,X         ;check lower address byte
    OR           X,$FF00,X       ;check if broadcast
    BXEQ         $FFFF,
                 BdgSNAPIP_
                 NLSAInvalid
    BRA.u        BdgSNAPIP_
                 NLSAValid
BdgSNAPIPClassD:
    SKIP.w
    BRA.u        BdgSNAPIP_
                 NLSAValid
BdgSNAPIP_
NLSAInvalid
    SWAIT        ;clean up after
                 SALE and
                 DALE
    DWAIT
    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_SRC_
                 INVALID
    STOP
BdgSNAPIP_
NLSAValid:
    SKIP.w       ;skip NLDA
    SKIP.w
    BCSERR BDG_
SNAPIP_CSError
    RSEL         LP1             ;Store source SIB pointer in LP1
    SWAIT
    SMOVE        Y               ;Y contains SALE result
    MOVE         Y,LP,LP2        ;LF1 points to Source Addr SIB
                                 ;Store dest SIB pointer in LP2
    BYNZ         BDG_SrcHit
BDG_SrcMiss:     ;*** Source
                 Cache Miss ***
    OR           XP,CMD_         ;Load command Word
                 FWDCP |
                 CMD_
                 UNICAST,Y
                                 ;Default MAC Ethernet Type
                                 ;Detault Low priority
    MOVEF        Y, FIRST        ;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
                CheckSrcPort
BDG_DestMiss:   ;*** 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
                CheckDestArea
BDG_SrcMove:    ;*** 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_DestAreaInvalid: ;*** 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           ;Done!!!
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           ;Done!!!
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           ;Done !!!
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 μs (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 date 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.

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.

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:

(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 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.

* * * * *

Our Docket/Ref. No.: APPT-0__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:

OIPE JC33 APR 11 2001 PATENT & TRADEMARK OFFICE

RECEIVED APR 12 2001 TC 2600 MAILROOM

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

_X_ 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.

_X_ Return postcard.

_X_ 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: Apr. 9, 2001

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

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

#5

4-22-02

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

OIPE JC190 APR 12 2002 PATENT & TRADEMARK OFFICE

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:

X 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)

X 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.

X (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: [signature]
Dov Rosenfeld, Reg. No. 38,687

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

	ATTY. DOCKET NO.	SERIAL NO.
INFORMATION DISCLOSURE STATEMENT	APPT-001-4	09/608266
	APPLICANT: Sarkissian et al.	
(Use several sheets if necessary)	FILING DATE: 6/30/2000	GROUP: 2667

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
AV	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	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES	NO
	AO							

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

	AP	

EXAMINER	DATE CONSIDERED
[signature]	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.

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

Patent 2731 2664

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al.	Group Art Unit: 2731
Serial No.: 09/608266	Examiner:
Filed: June 30, 2000	OIPE APR 12 2002 PATENT & TRADEMARK OFFICE
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	RECEIVED APR 17 2002 Technology Center 2600

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

X 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.

X Return postcard.

X 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.	Group Art Unit: 2731
Serial No.: 09/608266	Examiner:
Filed: June 30, 2000	
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	

OIPE JC180 APR 1 2 2002 PATENT & TRADEMARK OFFICE

RECEIVED
APR 1 7 2002
Technology Center 2600

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

_X_ 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.

_X_ Return postcard.

_X_ 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

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER FOR PATENTS
P.O 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	6

DATE MAILED: 09/10/2003

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

PTO-90C (Rev. 07-01)

Office Action Summary	Application No.	Applicant(s)
	09/608,266	SARKISSIAN ET AL.
	Examiner	**Art Unit**
	Alan Nguyen	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 negatived 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.

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

[56] **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.

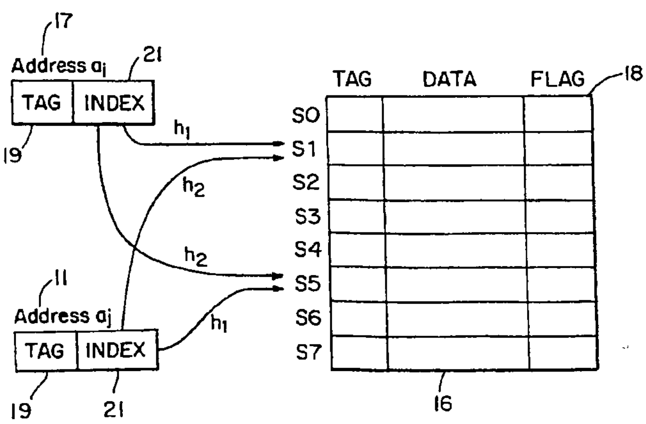
"A High Performance Memory Management Scheme"; Thakkar, Shreekant S. and Knowles, Alan E.; Computer; May 1986; IEEE Computer Society; pp. 8–20.

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



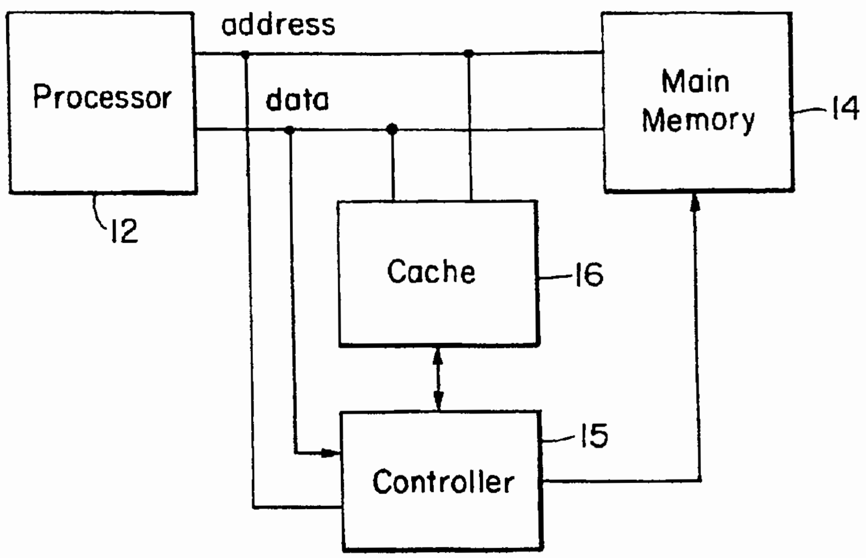

Processor
address
data
Main Memory
14
12
Cache
16
Controller
15

Fig. 1

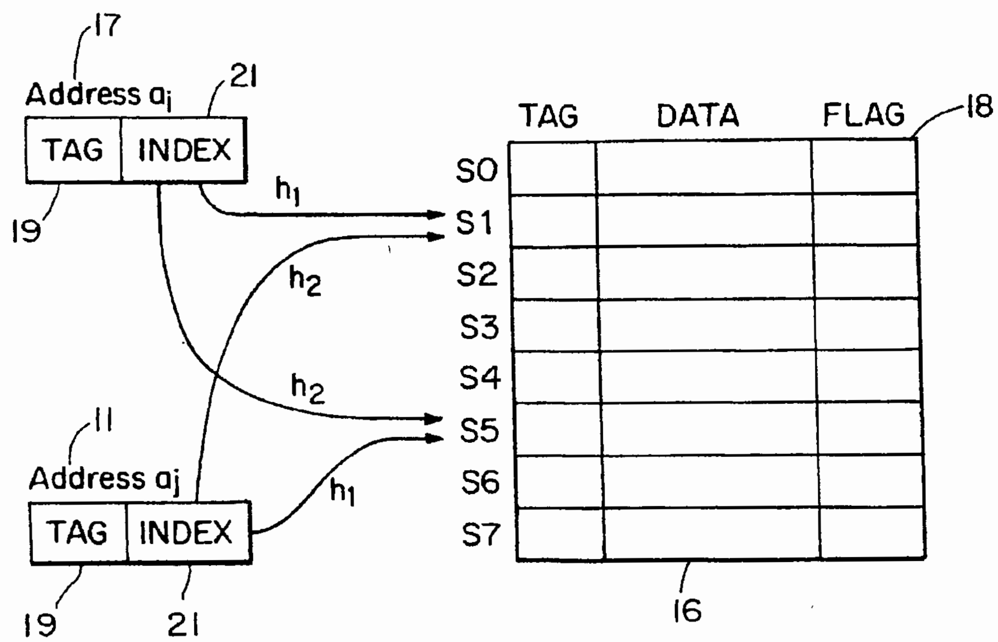

17
Address a_i
21
TAG
INDEX
19
h_1
h_2
h_2
11
Address a_j
TAG
INDEX
h_1
19
21
TAG
DATA
FLAG
18
S0
S1
S2
S3
S4
S5
S6
S7
16

Fig. 2A

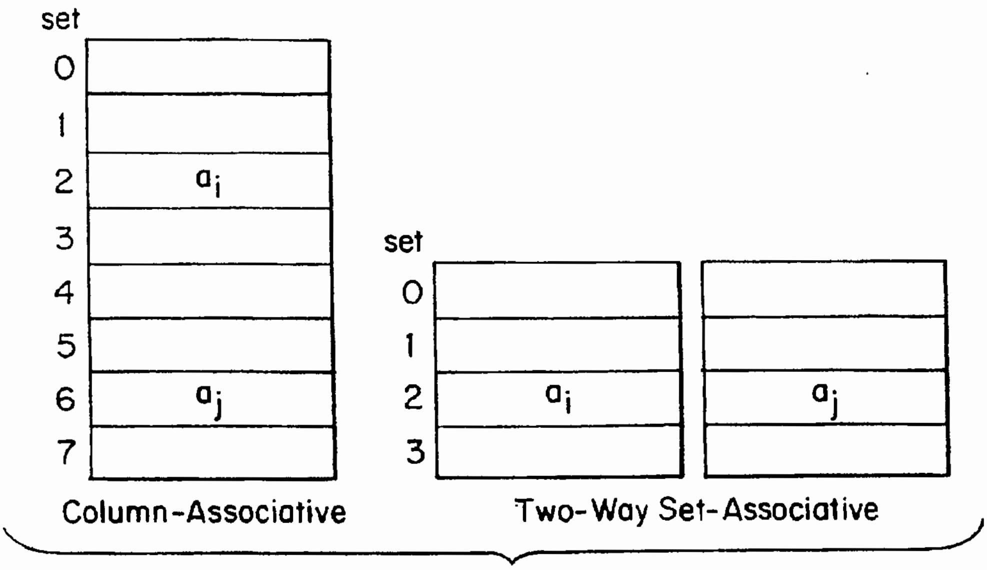

set
0
1
2
3
4
5
6
7
a_i
a_j
set
0
1
2
3
a_i
a_j
Column-Associative
Two-Way Set-Associative

Fig. 2B

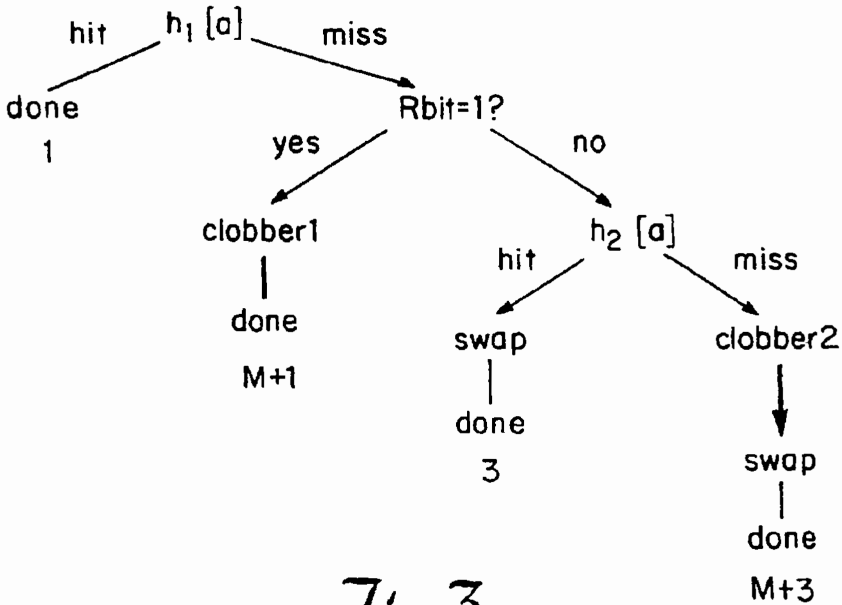

h1 [a]
hit
miss
done
1
Rbit=1?
yes
no
clobber1
done
M+1
h2 [a]
hit
miss
swap
done
3
clobber2
swap
done
M+3

Fig. 3

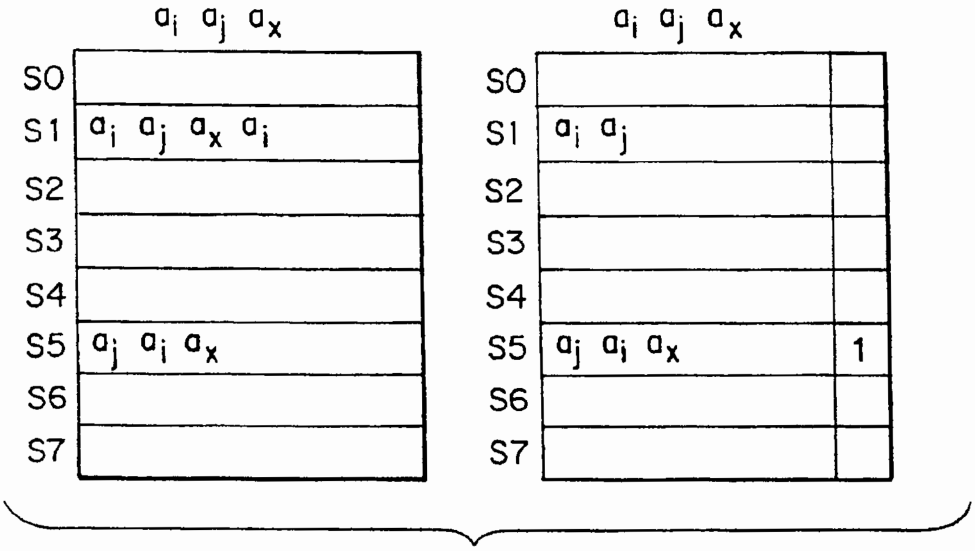

a_i a_j a_x
S0
S1
S2
S3
S4
S5
S6
S7
a_i a_j a_x a_i
a_j a_i a_x
a_i a_j a_x
a_i a_j
a_j a_i a_x
1

Fig. 4

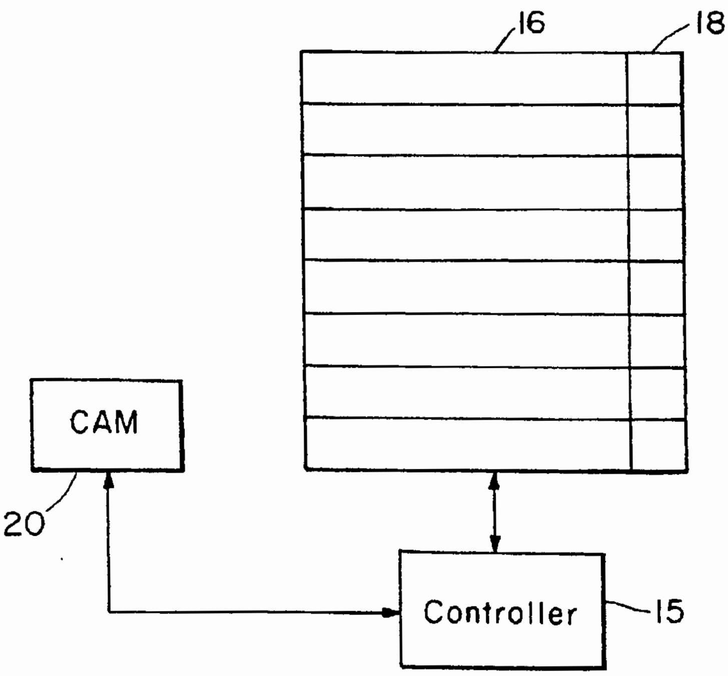

16
18
CAM
20
Controller
15

Fig. 5

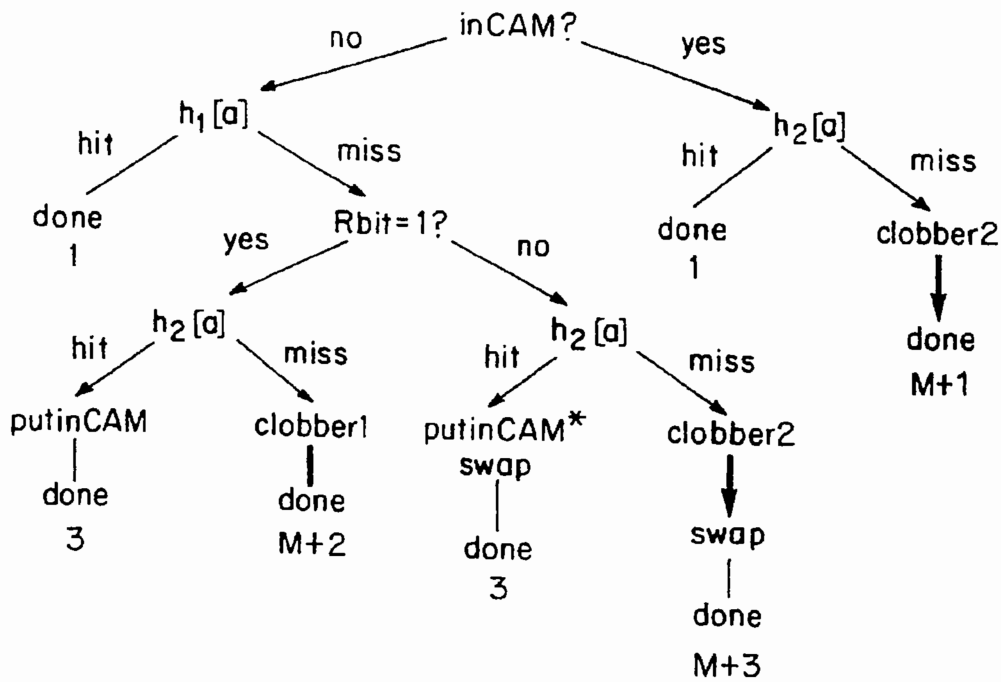

inCAM?
no
yes
h_1[a]
hit
miss
done 1
Rbit=1?
yes
no
h_2[a]
hit
miss
putinCAM
done 3
clobber1
done M+2
h_2[a]
hit
miss
putinCAM*
swap
done 3
clobber2
swap
done M+3
h_2[a]
hit
miss
done 1
clobber2
done M+1

Fig. 6

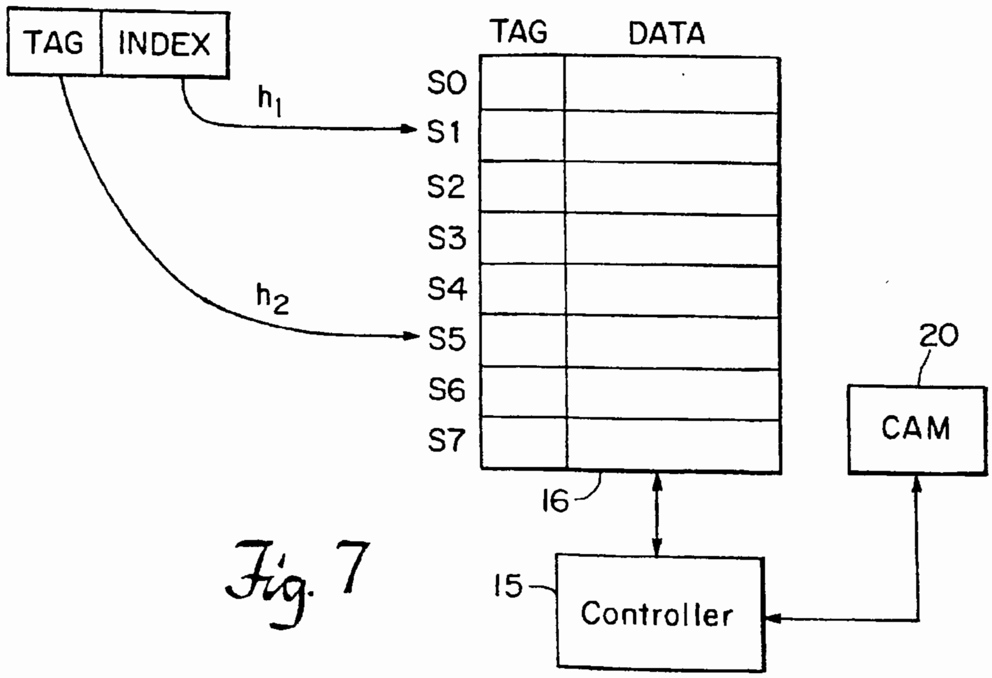

TAG
INDEX
h_1
h_2
TAG
DATA
S0
S1
S2
S3
S4
S5
S6
S7
16
20
CAM
15
Controller

Fig. 7

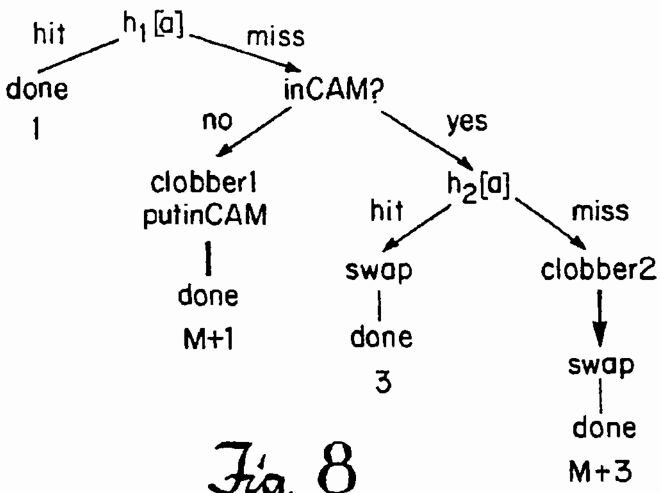

h1 [a]
hit
miss
done
1
inCAM?
no
yes
clobber1
putinCAM
done
M+1
h2[a]
hit
miss
swap
done
3
clobber2
swap
done
M+3

Fig. 8

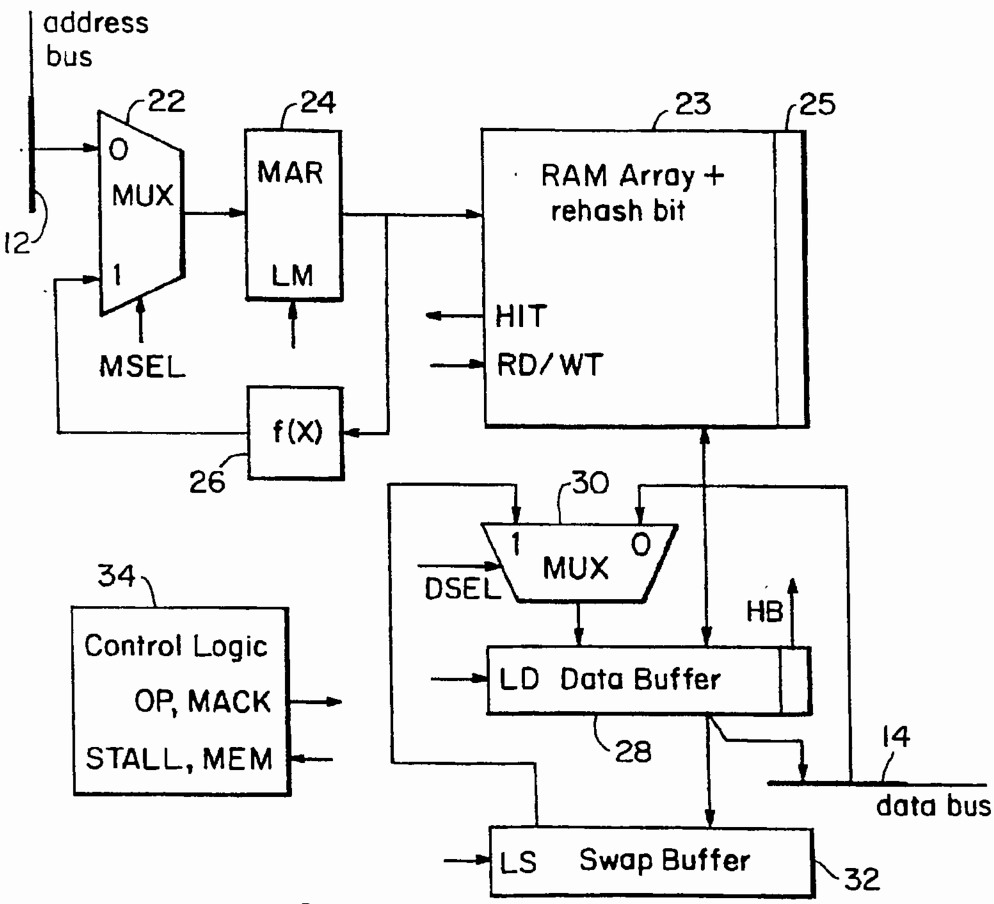

address
bus
12
22
0
MUX
1
MSEL
24
MAR
LM
23
25
RAM Array +
rehash bit
HIT
RD/WT
f(X)
26
30
1
MUX
0
DSEL
HB
34
Control Logic
OP, MACK
STALL, MEM
LD Data Buffer
28
14
data bus
LS Swap Buffer
32

Fig. 9

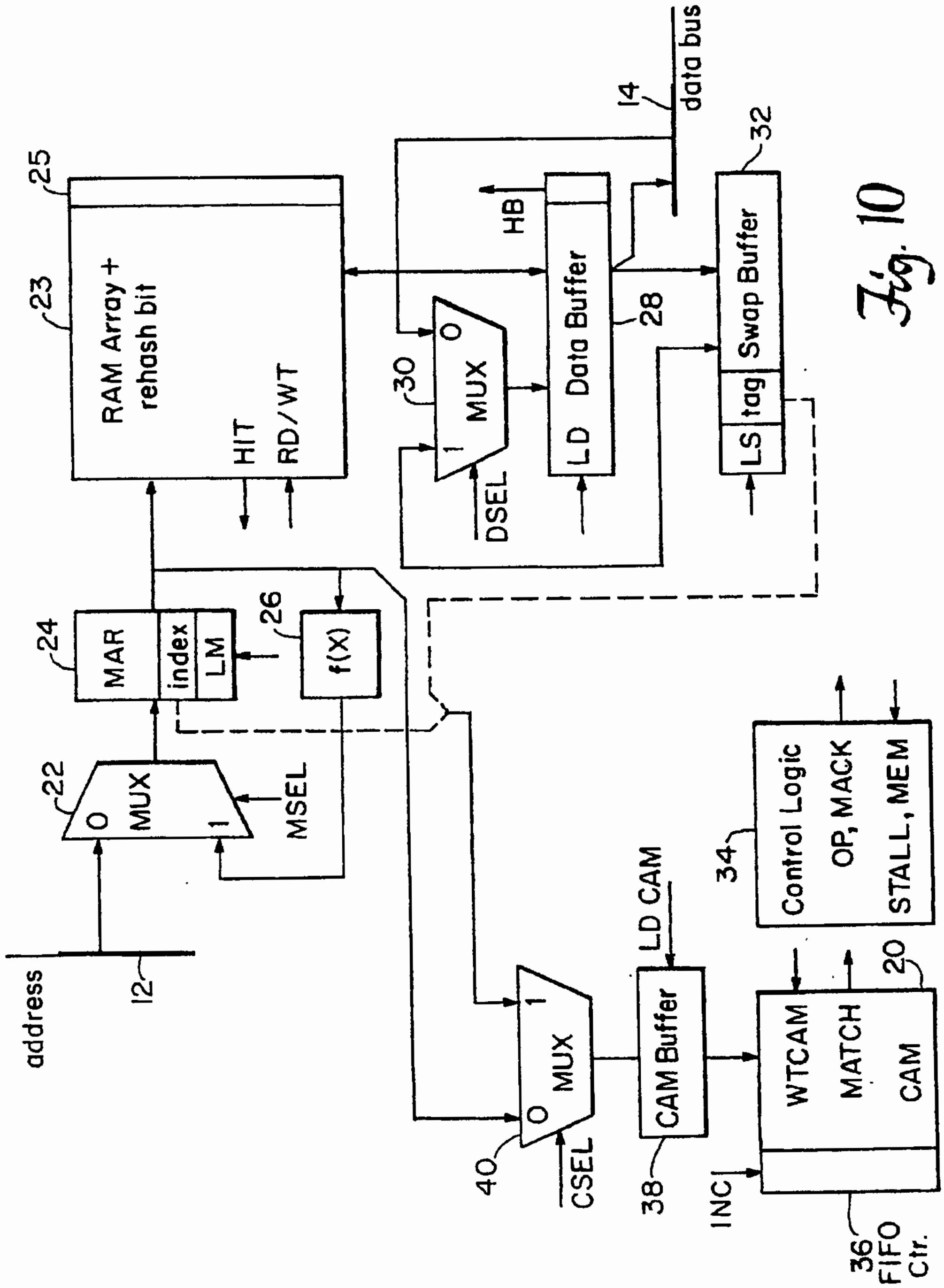
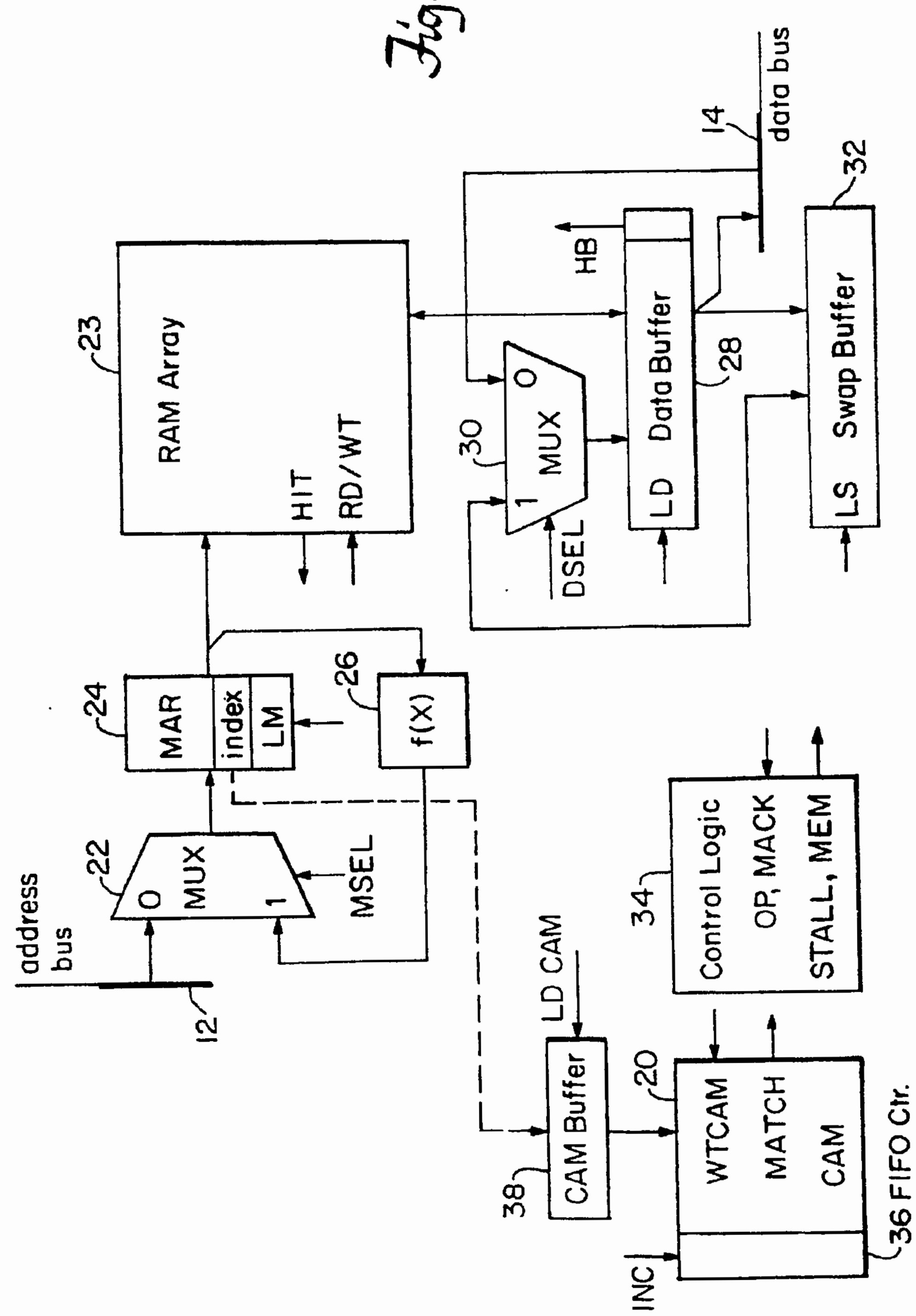

data bus
14
25
23
RAM Array + rehash bit
HIT
RD/WT
HB
LD
Data Buffer
28
32
Swap Buffer
LS
tag
30
MUX
0
1
DSEL
24
MAR
index
LM
26
f(X)
22
MUX
MSEL
address
12
34
Control Logic
OP, MACK
STALL, MEM
LD CAM
CAM Buffer
38
MUX
40
CSEL
WTCAM
MATCH
CAM
20
INC
36
FIFO
Ctr.

Fig. 10

Fig. 11



data bus
14
32
HB
RAM Array
23
Data Buffer
28
Swap Buffer
MUX
30
0
1
HIT
RD/WT
LD
LS
DSEL
24
MAR
index
LM
26
f(X)
22
MUX
0
1
MSEL
address bus
12
Control Logic
OP, MACK
STALL, MEM
34
LD CAM
CAM Buffer
38
20
WTCAM
MATCH
CAM
36 FIFO Ctr.
INC

CACHE MEMORY SYSTEM AND METHOD WITH MULTIPLE HASHING FUNCTIONS AND HASH CONTROL STORAGE

This application is a continuation of No. 07/926,613 filed Aug. 7, 1992, now abandoned.

BACKGROUND OF THE INVENTION

This invention relates generally to the field of high 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 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 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 for nearly all modern computer systems is to service all memory references from the cache and to minimize references 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 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., 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 implement 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, 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 drawbacks. 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 contains 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 locations allowed, one in each set. Thus, a block of data arriving from the main memory can go into a particular block 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 multiplexing 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 designing cache memory systems that can incorporate the advantages of both the direct-mapped cache and the d-way associative 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 $a_i, a_j, a_i, a_j, \ldots$, the second reference a_j misses and forces the data i indexed by a_i out of the set. The data i that is forced out is placed in the victim cache. Thus, the third reference address, a_i, 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 improvement 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_i **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 a_j a_x . . . 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 ai_j are swapped so that i is in S1 and x is in S5. This pattern continues as long as a_j 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 a_i . . . , 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 . . . . 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 $h_i[a_i]$. 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 location.

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 be 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 additions, 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 f1[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	f1[a]
	!HIT,HB	STALL,MSEL,LM,RD,LS	fc[a]
f1[a]	HIT	MSEL,LM,WT,CSEL, LDCAM,WTCAM	f2[a]
	!HIT	MEM	WAIT1
f2[a]		DSEL,LD,INC	f3[a]
f3[a]		MSEL,LM,WT,LDCAM	IDLE
fc[a]	HIT	LDCAM,WTCAM	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 f1[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 f1[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]
	!HIT!MATCH	MSEL,STALL,WTCAM	XWAIT
f1[a]	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 $h_1[a]$ 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, h_1 and h_2, 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 dependent 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 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.

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

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;

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-indentified 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

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

United States Patent [19]

Chang

[11] **Patent Number:** **4,458,310**

[45] **Date of Patent:** **Jul. 3, 1984**

[54] **CACHE MEMORY USING A LOWEST PRIORITY REPLACEMENT CIRCUIT**

[75] Inventor: **Shih-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

[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

[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.

17 Claims, 5 Drawing Figures

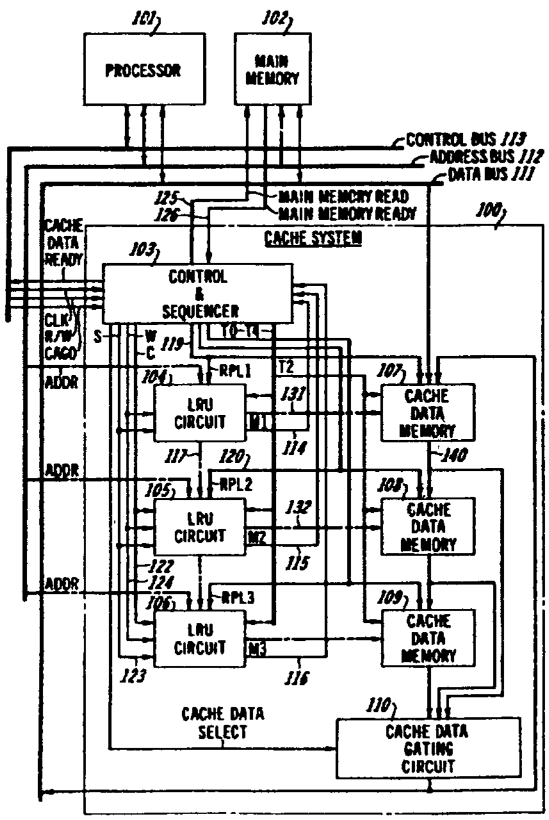
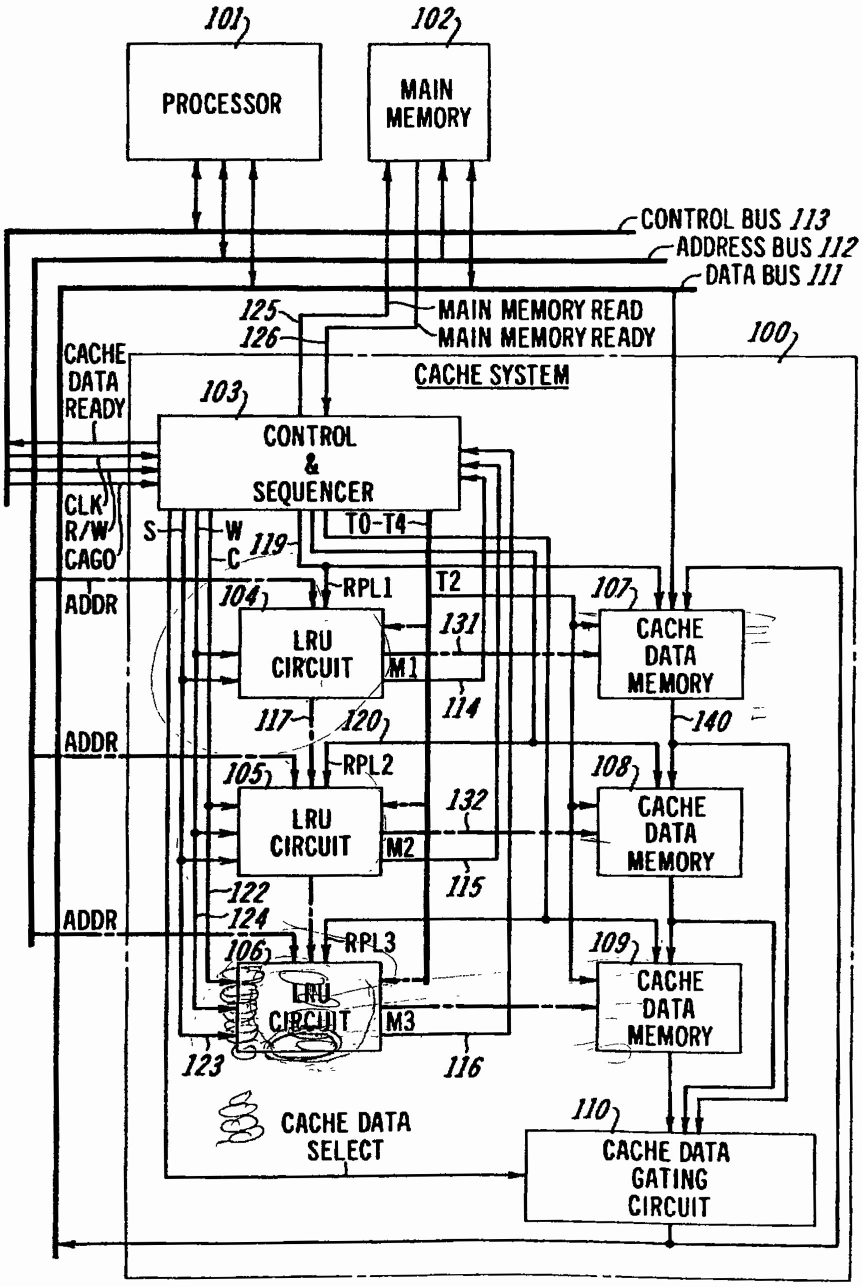


FIG. 1


101
102
PROCESSOR
MAIN MEMORY
CONTROL BUS 113
ADDRESS BUS 112
DATA BUS 111
125
126
MAIN MEMORY READ
MAIN MEMORY READY
100
CACHE DATA READY
CACHE SYSTEM
103
CONTROL & SEQUENCER
CLK
R/W
CAGO
S
W
C
119
T0-T4
ADDR
104
RPL1
T2
107
LRU CIRCUIT
M1
131
114
CACHE DATA MEMORY
140
117
120
ADDR
105
RPL2
108
LRU CIRCUIT
M2
132
115
CACHE DATA MEMORY
122
124
ADDR
106
RPL3
109
LRU CIRCUIT
M3
116
CACHE DATA MEMORY
123
CACHE DATA SELECT
110
CACHE DATA GATING CIRCUIT

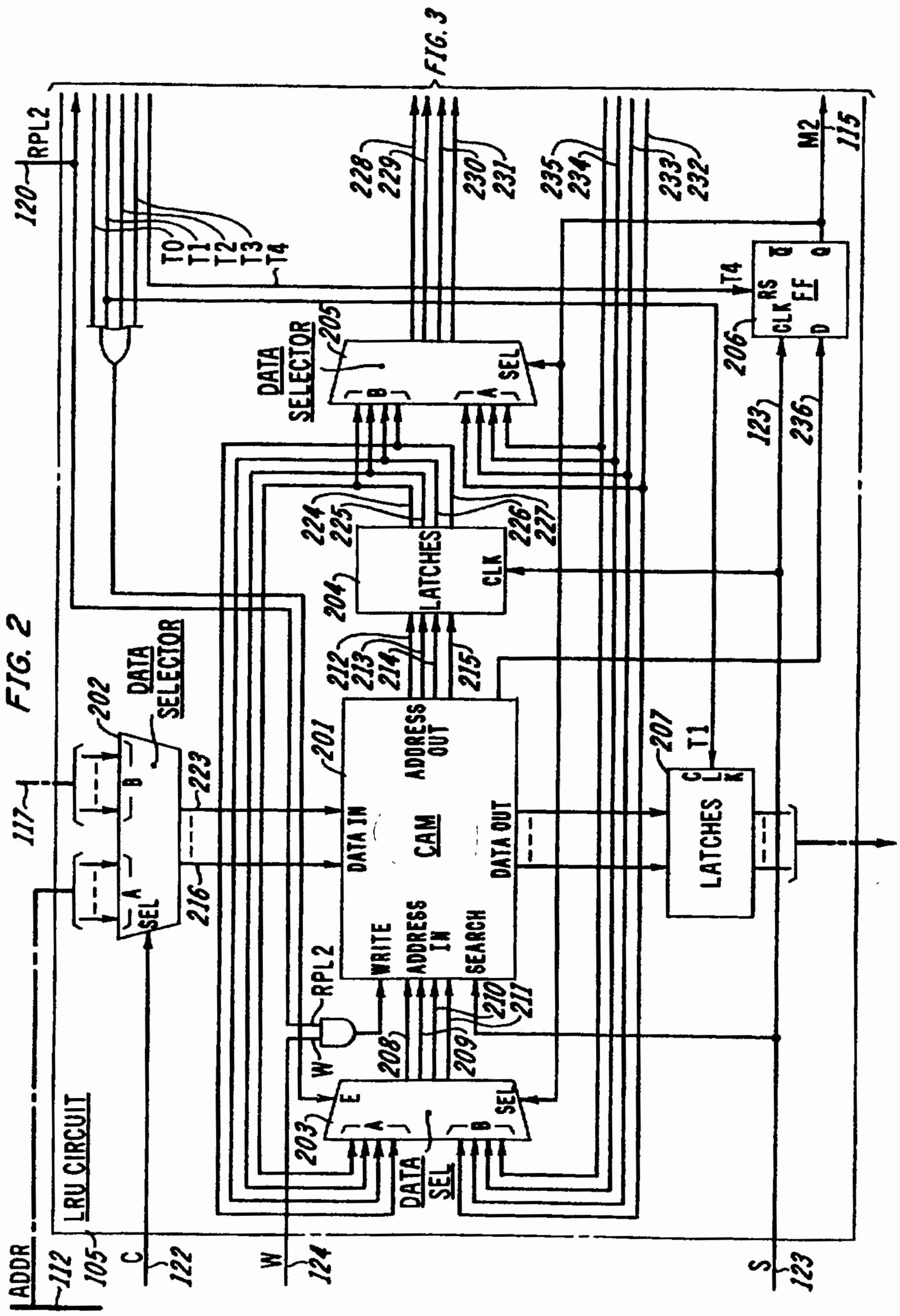


FIG. 3
RPL2
120
T0
T1
T2
T3
T4
228
229
230
231
235
234
233
232
M2
115
DATA SELECTOR
205
SEL
FF
RS
CLK
D
Q
206
123
236
LATCHES
CLK
204
224
225
226
227
212
213
214
215
DATA SELECTOR
202
117
223
216
SEL
CAM
DATA IN
ADDRESS OUT
DATA OUT
WRITE
ADDRESS IN
SEARCH
201
207
T1
LATCHES
RPL2
W
208
209
210
211
203
DATA SEL
LRU CIRCUIT
ADDR
112
105
C
122
W
124
S
123

FIG. 2

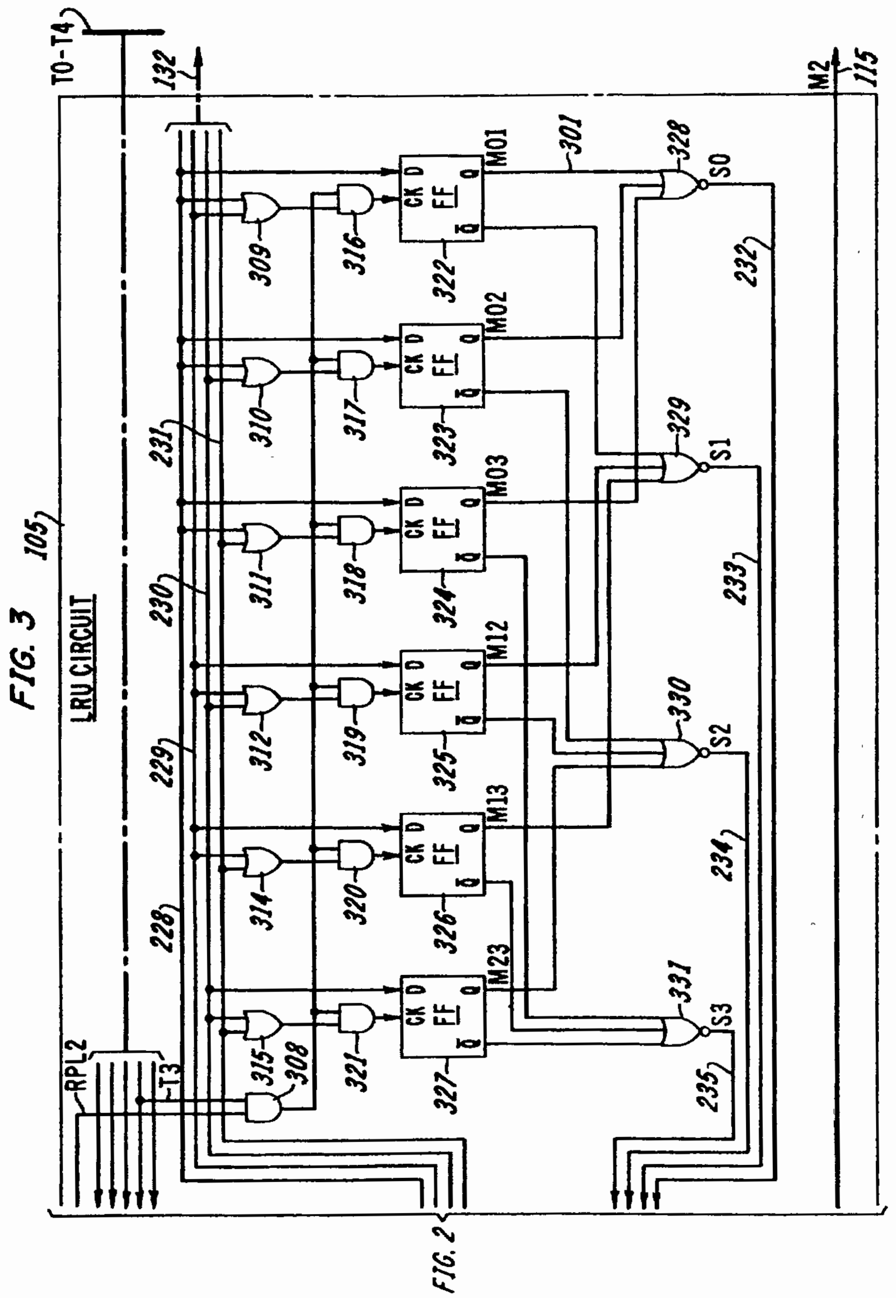


FIG. 3
105
LRU CIRCUIT
TO-T4
132
M2
115
RPL2
T3
M01
M02
M03
M12
M13
M23
S0
S1
S2
S3
FIG. 2

FIG. 4

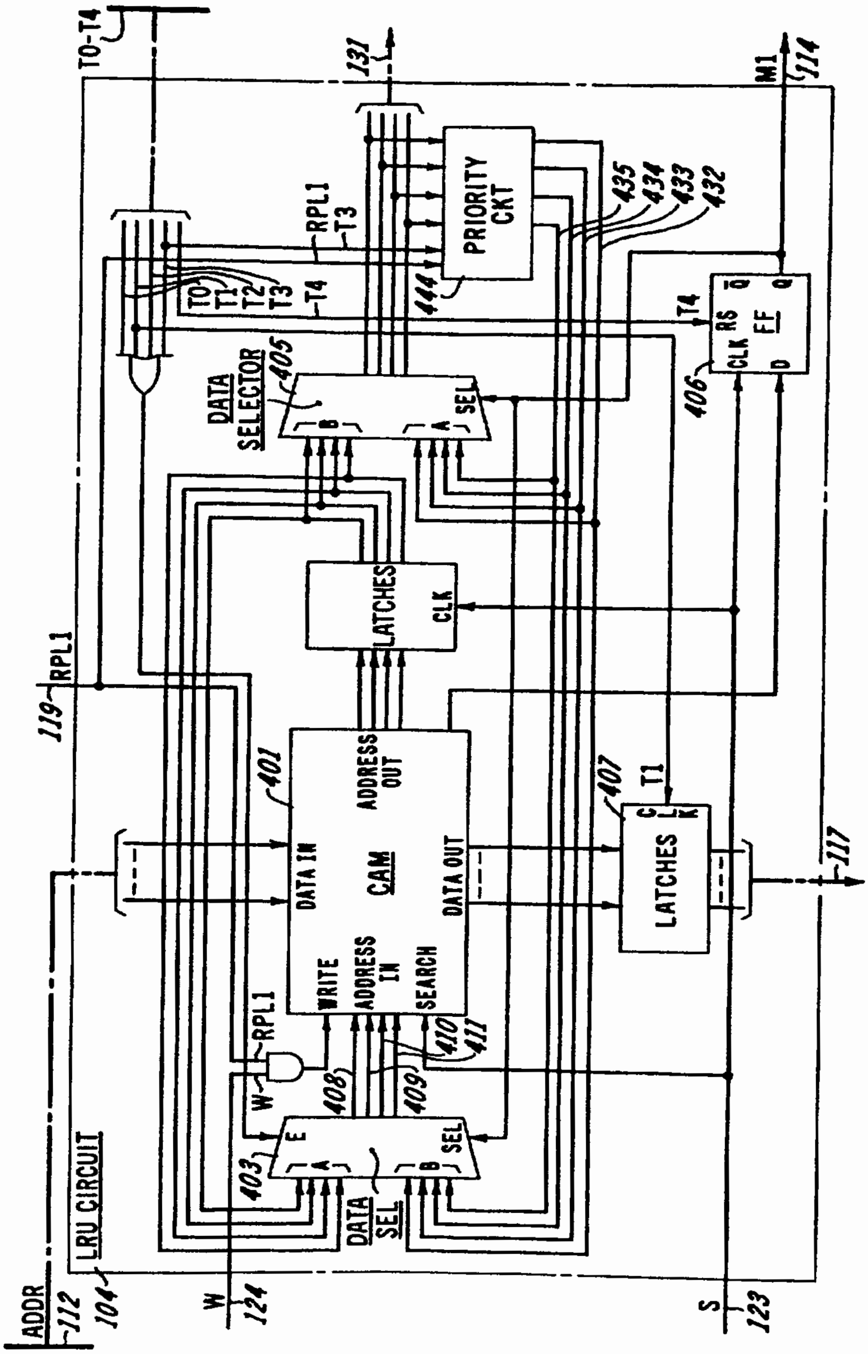

TO-T4
131
M1
114
PRIORITY CKT
435
434
433
432
RPL1
T3
T0
T1
T2
T3
T4
444
DATA SELECTOR
405
B
A
SEL
RS
FF
CLK
D
Q
406
LATCHES
CLK
119
RPL1
401
CAM
DATA IN
ADDRESS OUT
DATA OUT
407
T1
LATCHES
117
WRITE
ADDRESS IN
SEARCH
RPL1
410
411
W
408
409
E
A
B
SEL
403
DATA SEL
LRU CIRCUIT
ADDR
112
104
W
124
S
123

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 circit 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.

Cl. 4, 5, 6, 7

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 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 coductor 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 = $\overline{M01} \cdot \overline{M02} \cdot \overline{M03}$
S1 = $M01 \cdot M12 \cdot \overline{M13}$
S2 = $M02 \cdot M12 \cdot M23$
S3 = $M03 \cdot M13 \cdot M23$

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 308 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 108 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:

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 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.

* * * * *

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.

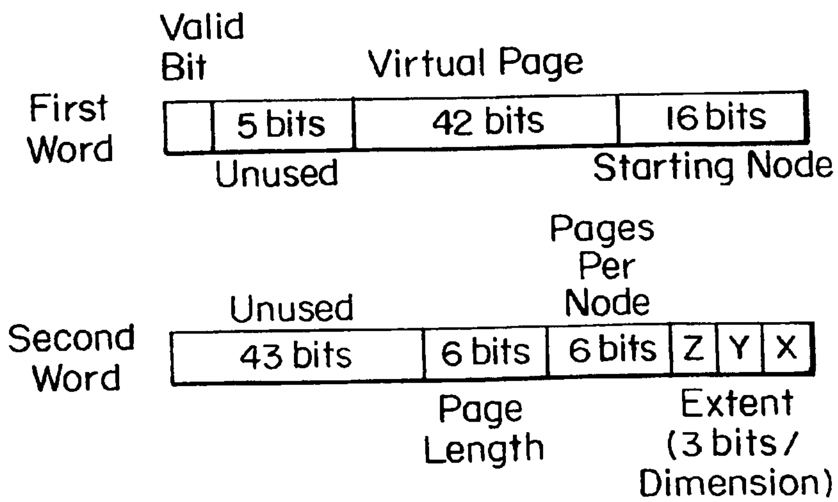
Dally, William J, et al., "M-Machine Architecture v1.0 MIT Concurrent VLSI Architecture Memo 58," Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Aug. 24, 1994, pp. 1–50.

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



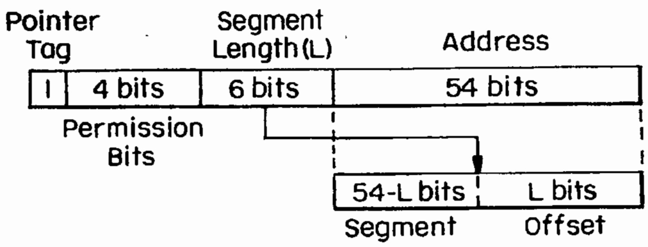

Pointer Tag
Segment Length (L)
Address
1
4 bits
6 bits
54 bits
Permission Bits
54-L bits
L bits
Segment
Offset

FIG. 1A

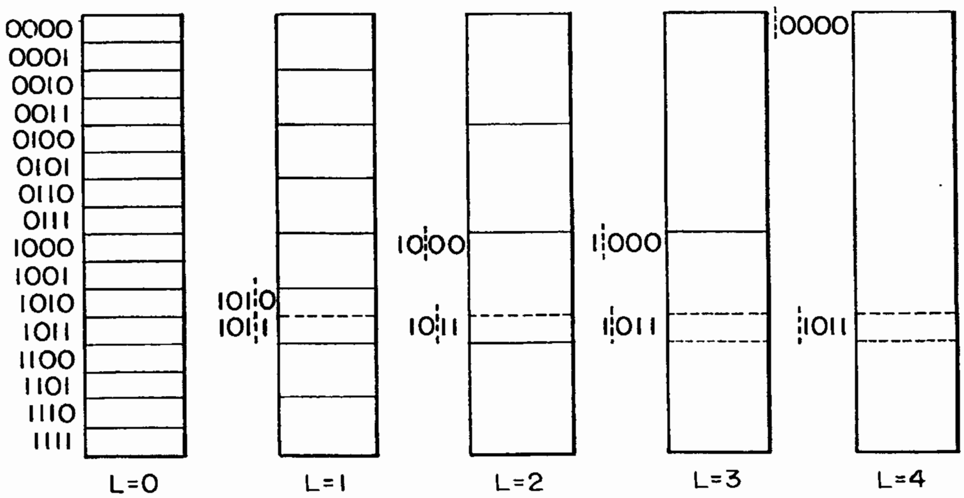
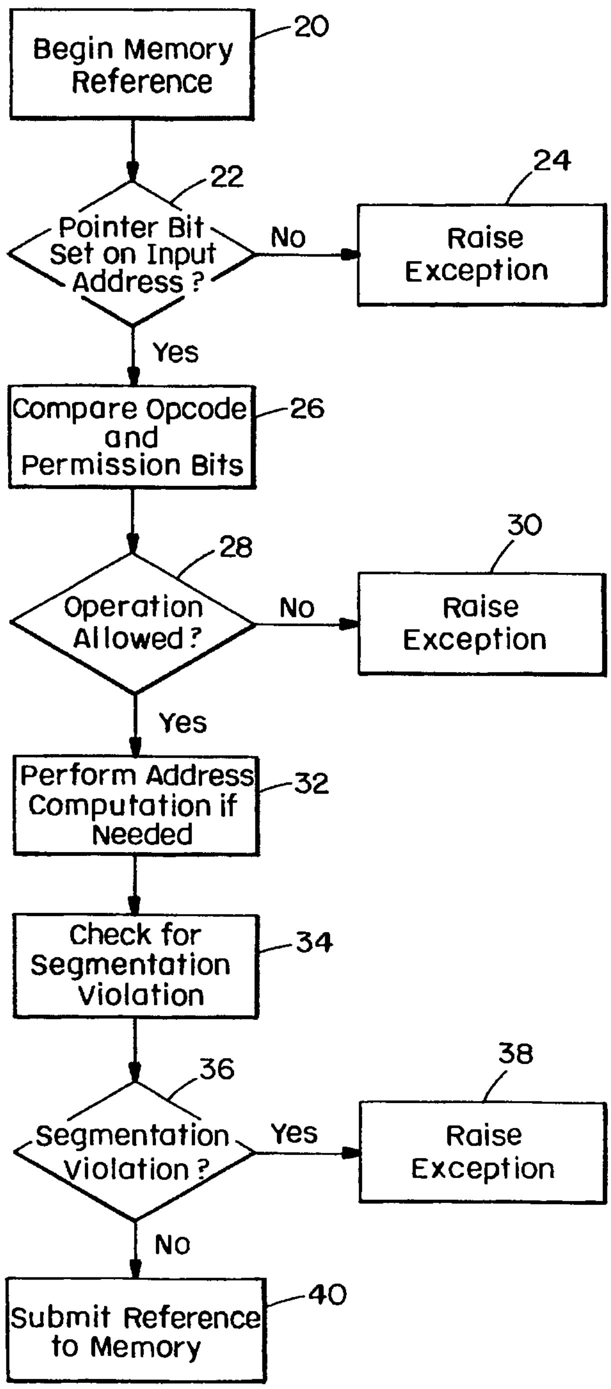

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
101:0
101:1
10:00
10:11
1:000
1:011
:0000
:1011
L=0
L=1
L=2
L=3
L=4

FIG. 1B



Begin Memory Reference
20
Pointer Bit Set on Input Address ?
22
No
Raise Exception
24
Yes
Compare Opcode and Permission Bits
26
Operation Allowed ?
28
No
Raise Exception
30
Yes
Perform Address Computation if Needed
32
Check for Segmentation Violation
34
Segmentation Violation ?
36
Yes
Raise Exception
38
No
Submit Reference to Memory
40

FIG. 2A

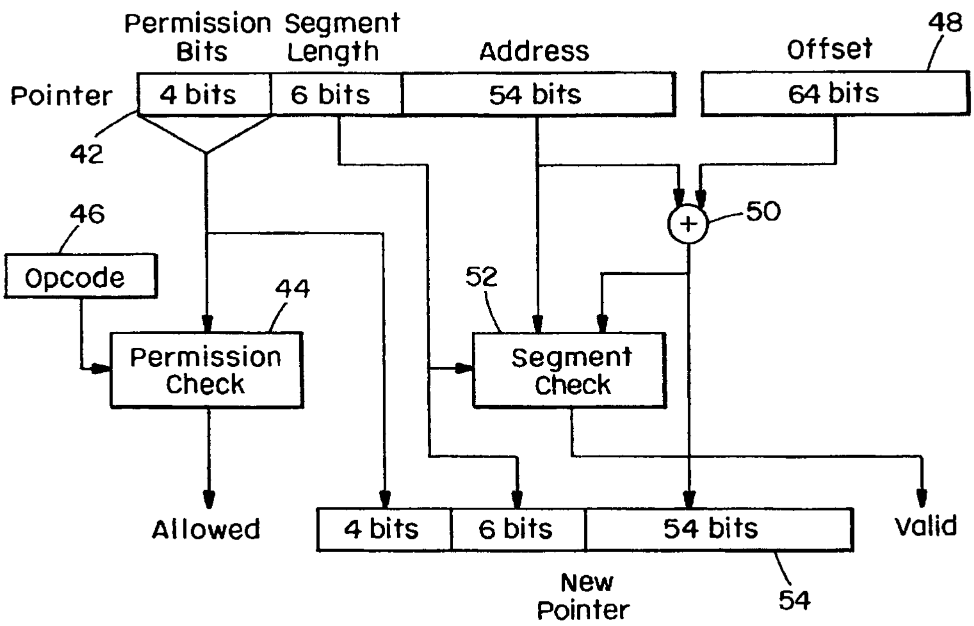
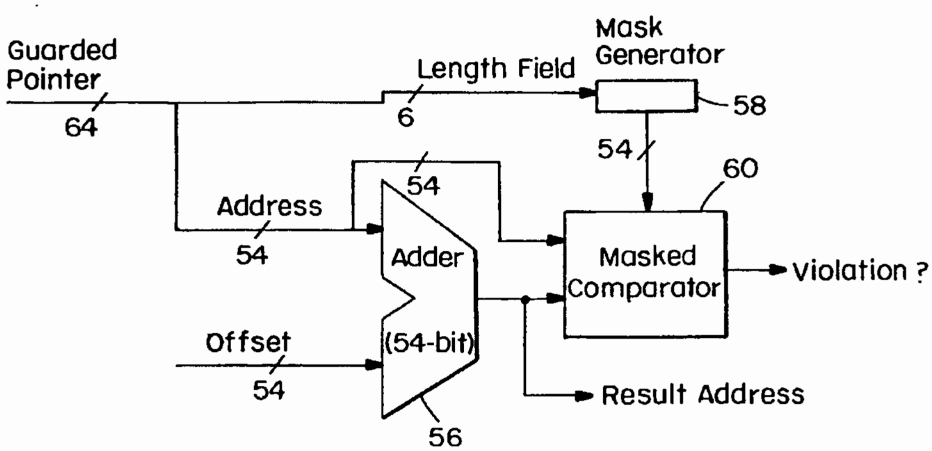
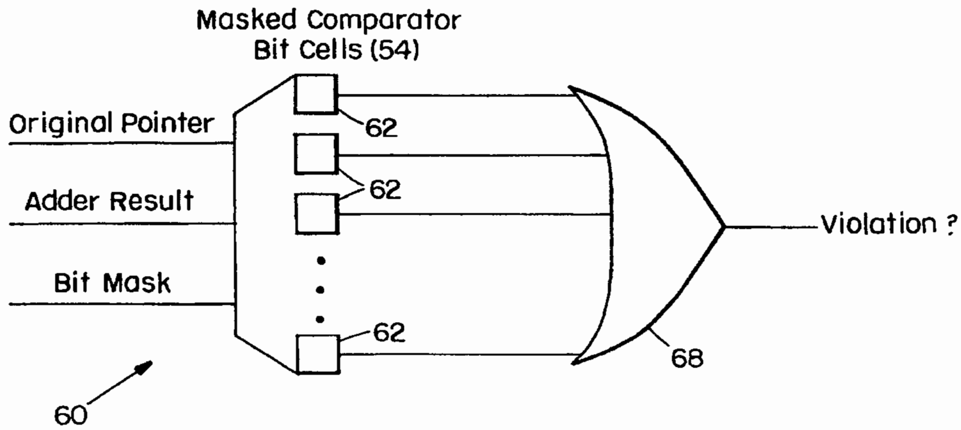

Permission Bits
Segment Length
Address
Offset
48
Pointer
4 bits
6 bits
54 bits
64 bits
42
46
50
Opcode
44
52
Permission Check
Segment Check
Allowed
4 bits
6 bits
54 bits
Valid
New Pointer
54

FIG. 2B



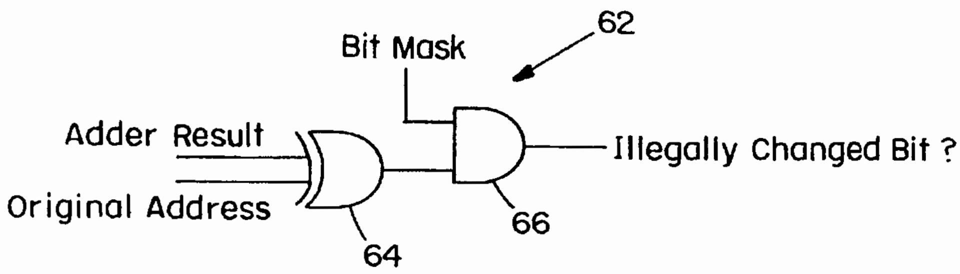
Guarded Pointer
64
Length Field
6
Mask Generator
58
54
60
Address
54
54
Adder
(54-bit)
Offset
54
56
Masked Comparator
Violation ?
Result Address

FIG. 3



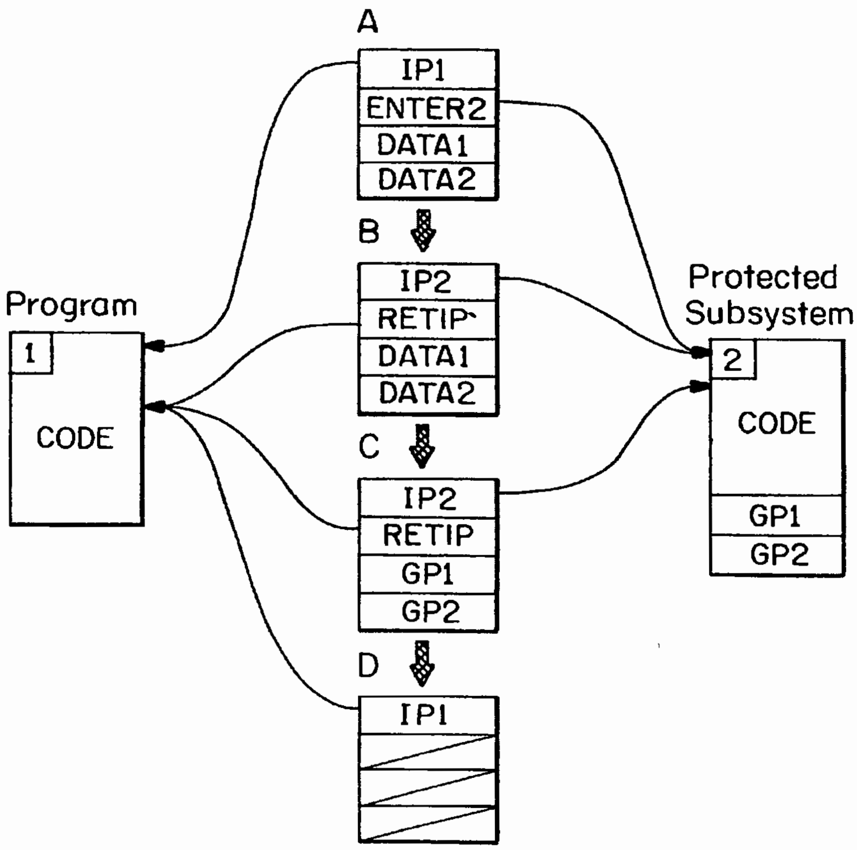
Masked Comparator Bit Cells (54)
Original Pointer
Adder Result
Bit Mask
62
62
62
68
Violation ?
60

FIG. 4



62
Bit Mask
Adder Result
Original Address
Illegally Changed Bit ?
66
64

FIG. 5



A
IP1
ENTER2
DATA1
DATA2
B
IP2
RETIP'
DATA1
DATA2
C
IP2
RETIP
GP1
GP2
D
IP1
Program
1
CODE
Protected
Subsystem
2
CODE
GP1
GP2

FIG. 6

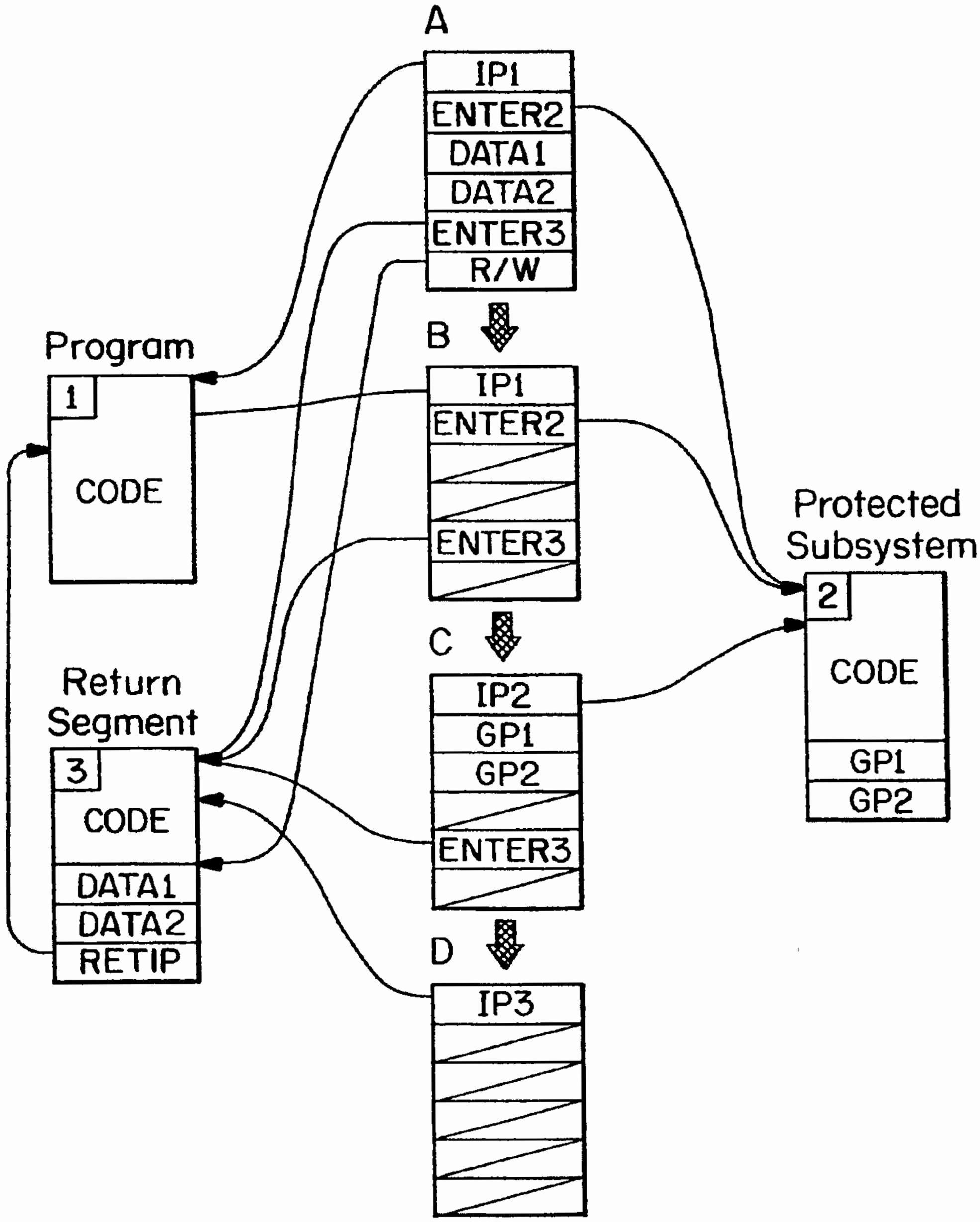

A
IP1
ENTER2
DATA1
DATA2
ENTER3
R/W
B
IP1
ENTER2
ENTER3
C
IP2
GP1
GP2
ENTER3
D
IP3
Program
1
CODE
Return Segment
3
CODE
DATA1
DATA2
RETIP
Protected Subsystem
2
CODE
GP1
GP2

FIG. 7

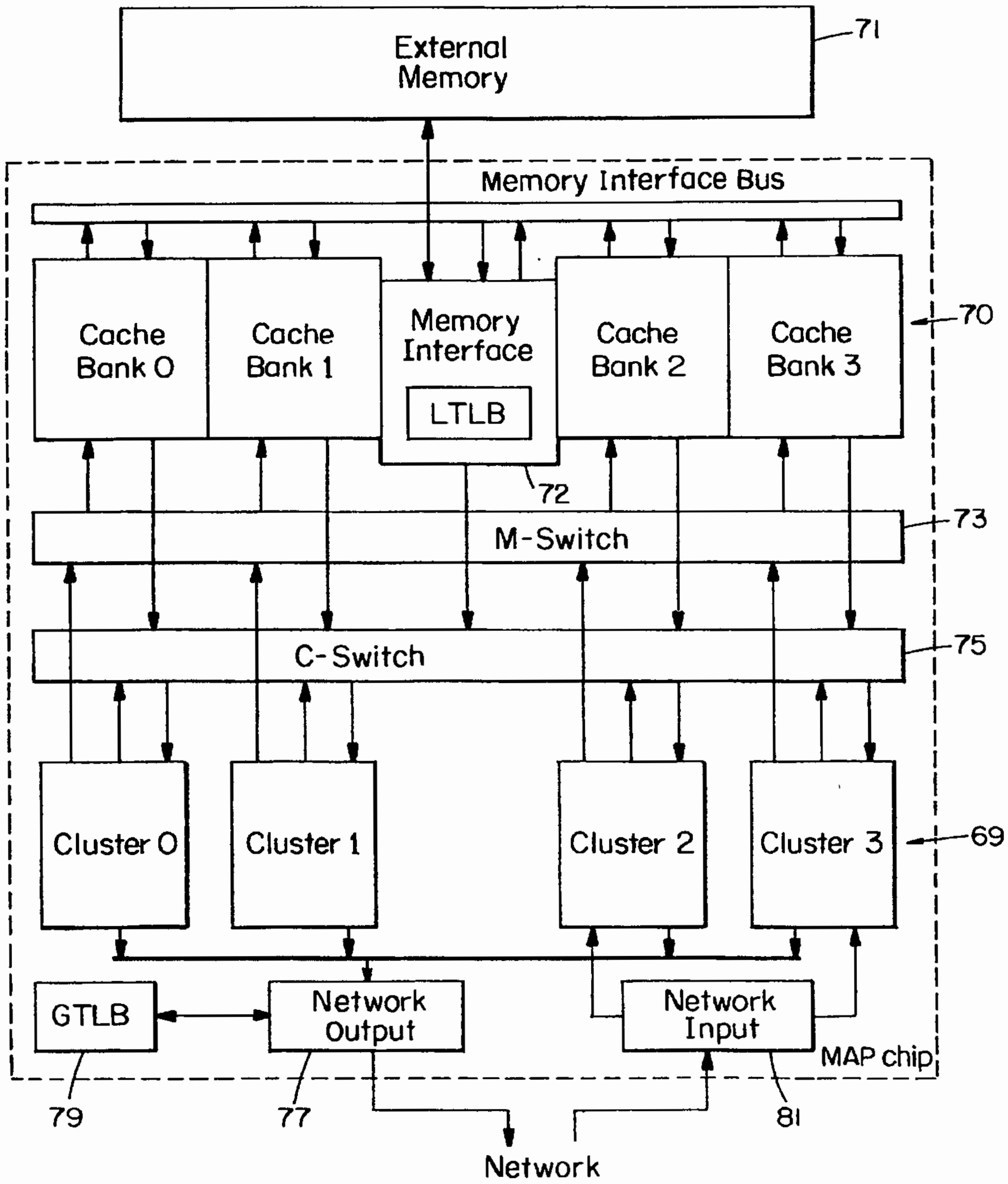

External Memory
71
Memory Interface Bus
Cache Bank 0
Cache Bank 1
Memory Interface
LTLB
Cache Bank 2
Cache Bank 3
70
72
M-Switch
73
C-Switch
75
Cluster 0
Cluster 1
Cluster 2
Cluster 3
69
GTLB
Network Output
Network Input
MAP chip
79
77
81
Network

FIG. 8

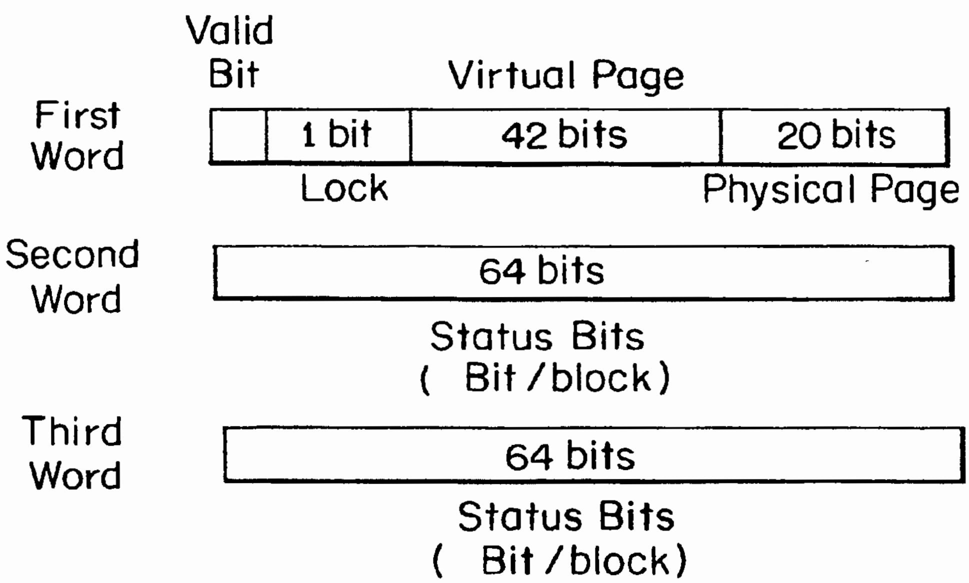

Valid Bit
Virtual Page
First Word
1 bit
42 bits
20 bits
Lock
Physical Page
Second Word
64 bits
Status Bits
(Bit /block)
Third Word
64 bits
Status Bits
(Bit /block)

FIG. 9

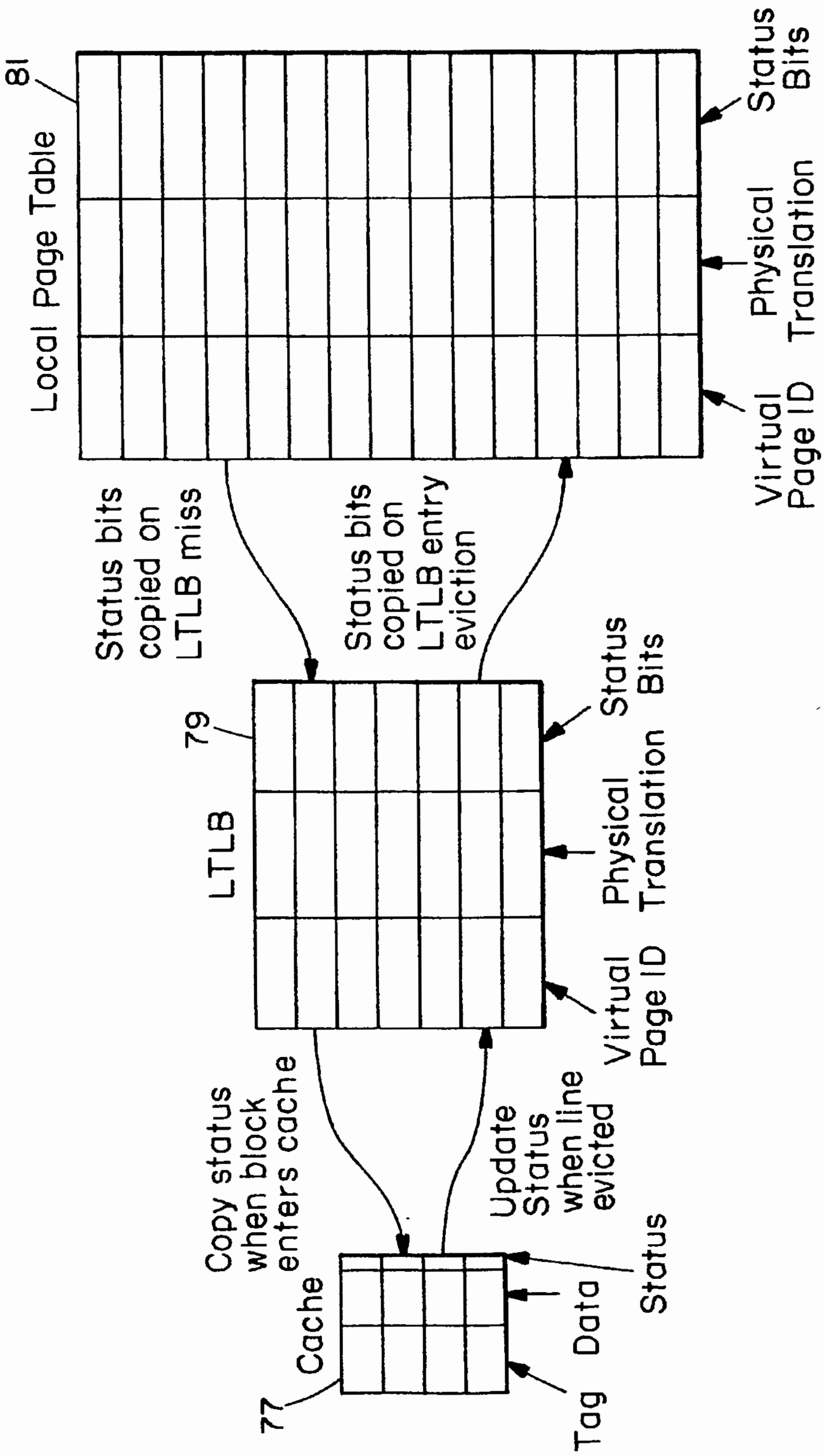

Local Page Table
81
Status Bits
Physical Translation
Virtual Page ID
Status bits copied on LTLB miss
Status bits copied on LTLB entry eviction
LTLB
79
Status Bits
Physical Translation
Virtual Page ID
Copy status when block enters cache
Update Status when line evicted
Cache
77
Status
Data
Tag

FIG. 10

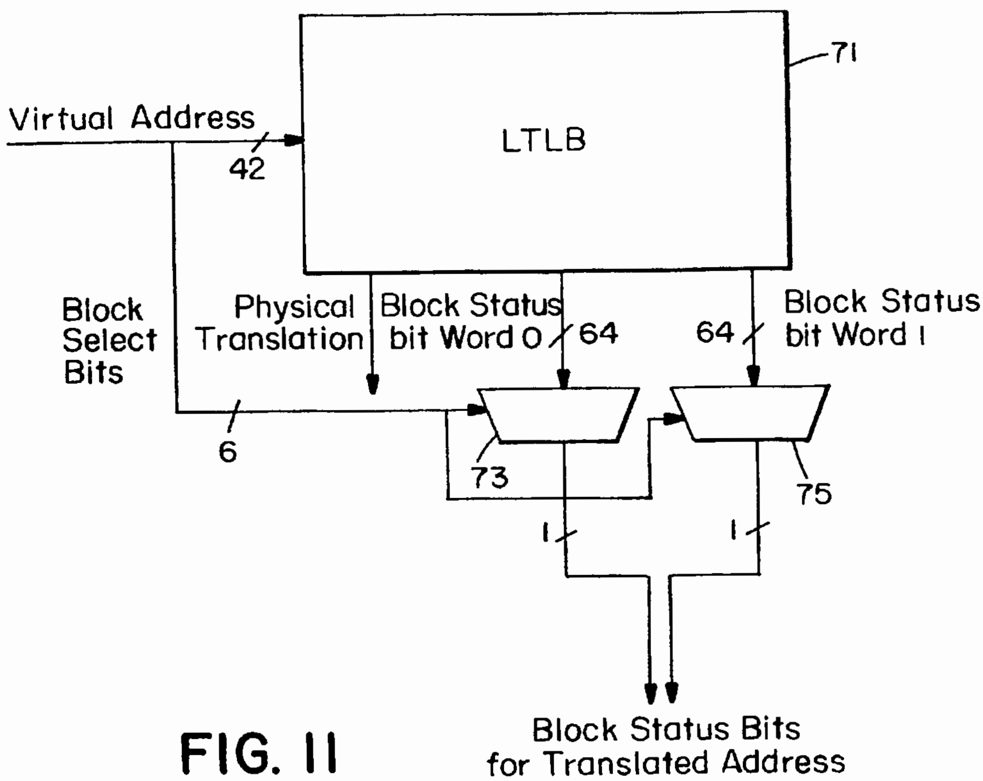
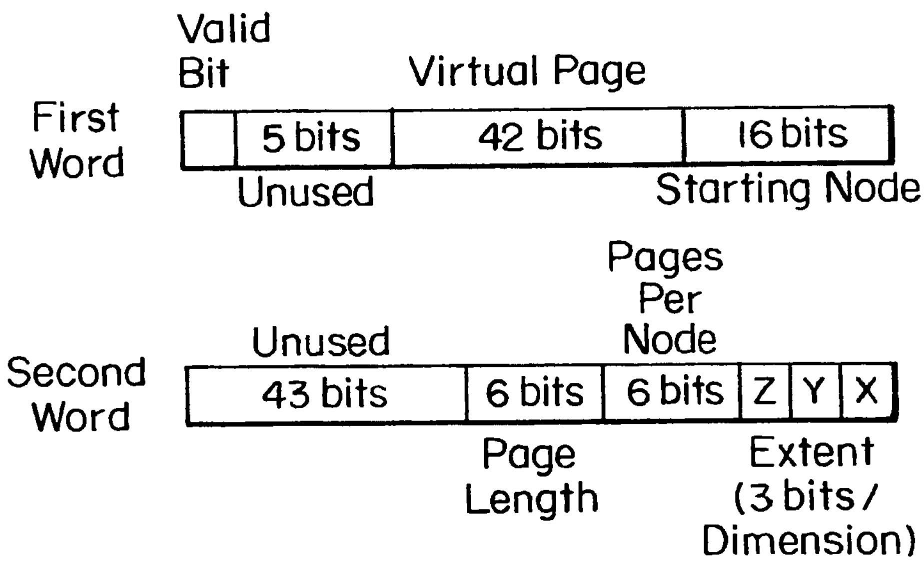

71
LTLB
Virtual Address
42
Block
Select
Bits
Physical
Translation
Block Status
bit Word 0
64
64
Block Status
bit Word 1
6
73
75
1
1
Block Status Bits
for Translated Address

FIG. 11



Valid Bit
Virtual Page
First Word
5 bits
42 bits
16 bits
Unused
Starting Node
Pages Per Node
Unused
Second Word
43 bits
6 bits
6 bits
Z
Y
X
Page Length
Extent (3 bits / Dimension)

FIG. 13

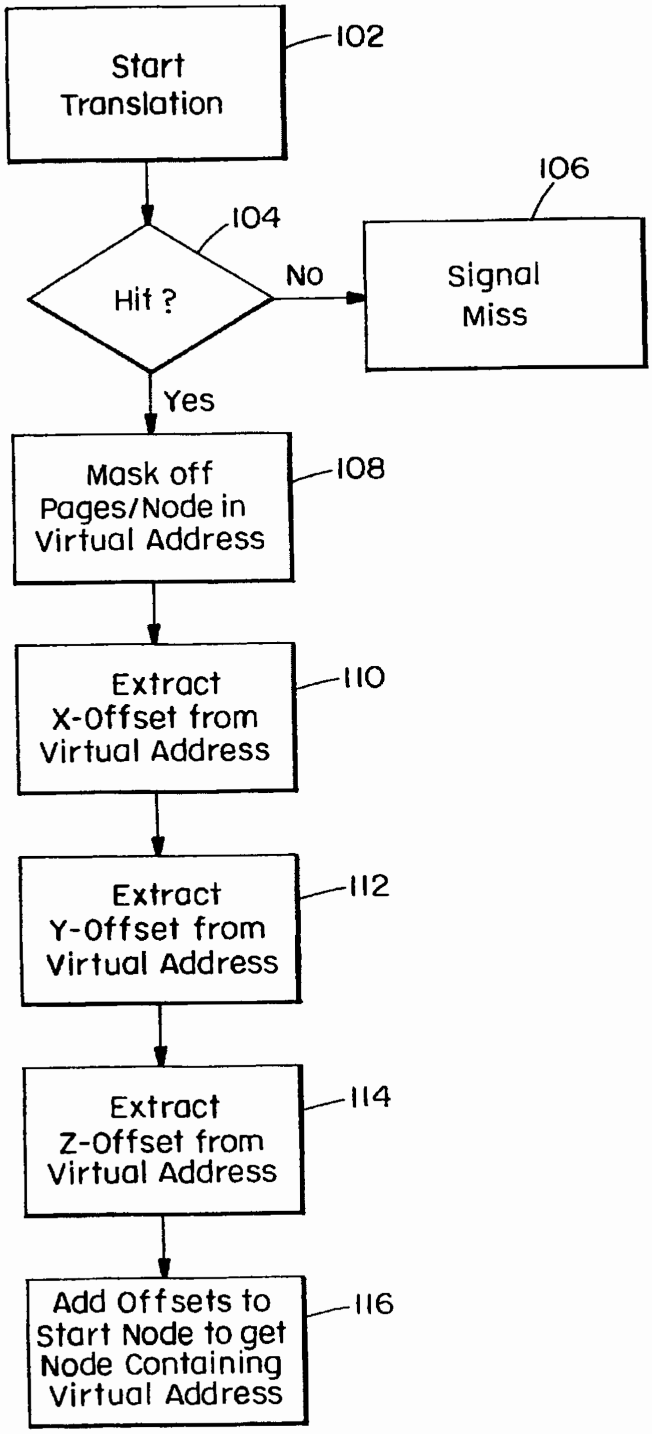

Start Translation
102
106
104
Hit ?
No
Signal Miss
Yes
Mask off Pages/Node in Virtual Address
108
Extract X-Offset from Virtual Address
110
Extract Y-Offset from Virtual Address
112
Extract Z-Offset from Virtual Address
114
Add Offsets to Start Node to get Node Containing Virtual Address
116

FIG. 14A

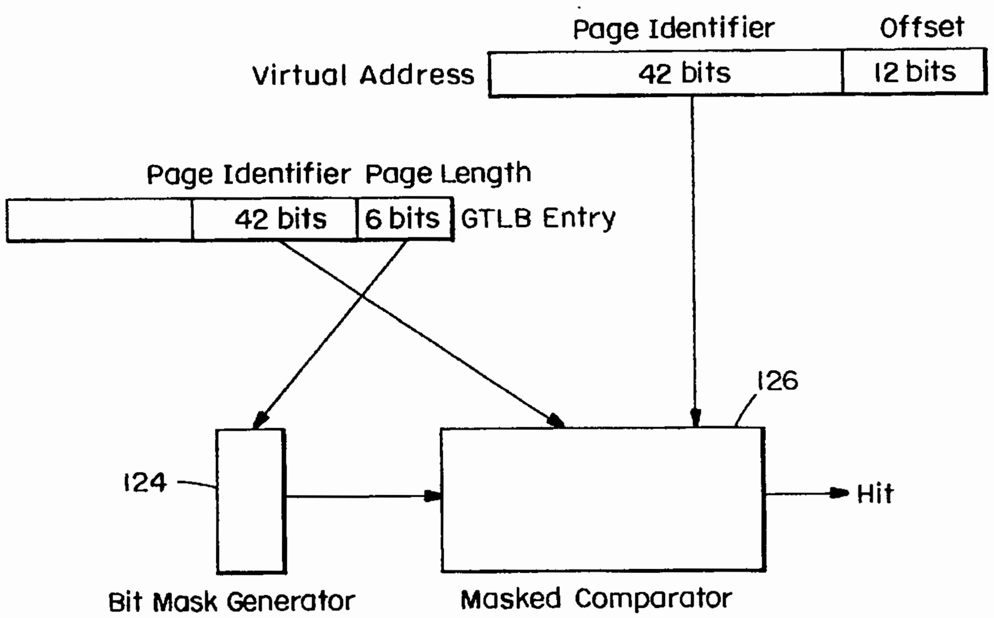

Page Identifier
Offset
Virtual Address
42 bits
12 bits
Page Identifier
Page Length
42 bits
6 bits
GTLB Entry
126
124
Hit
Bit Mask Generator
Masked Comparator

FIG. 14B

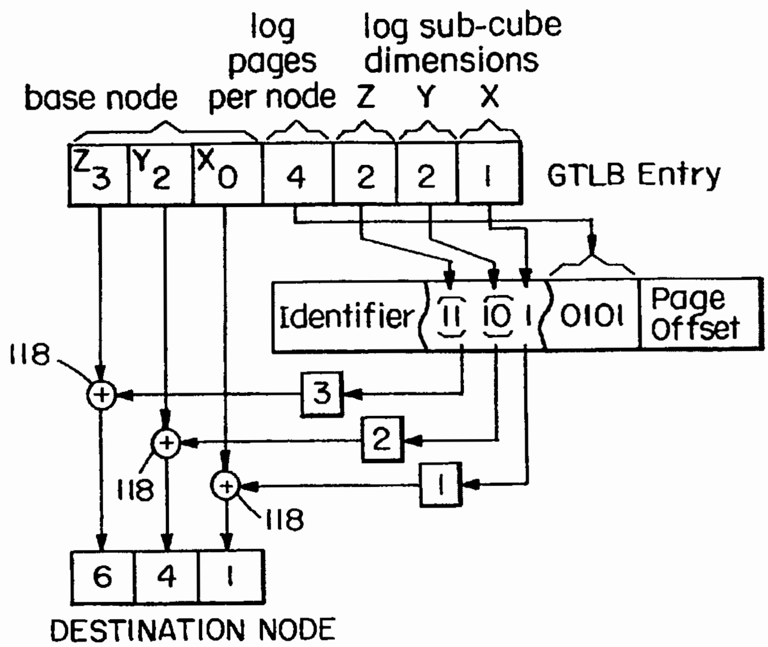


FIG. 15A

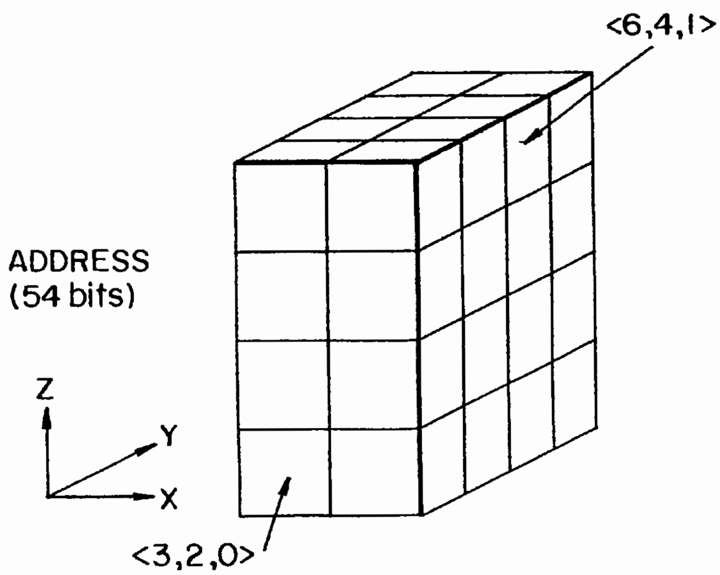
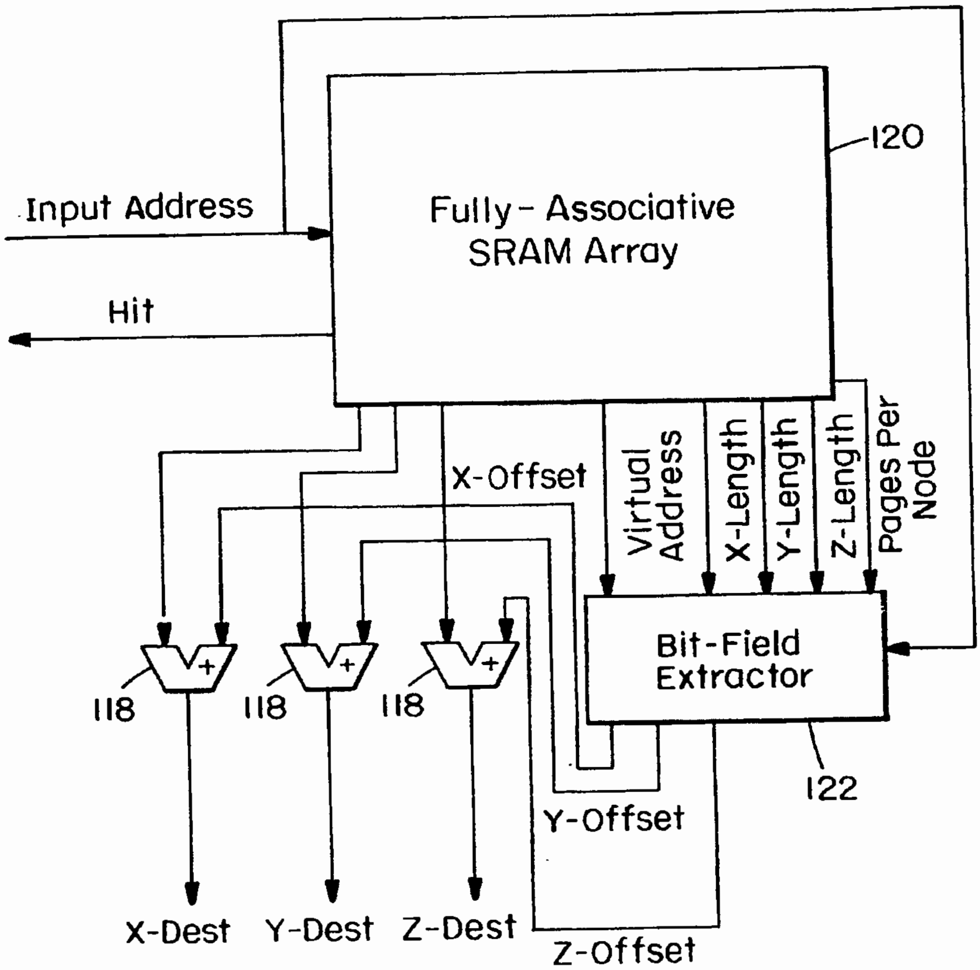


FIG. 15B



120
Input Address
Fully-Associative
SRAM Array
Hit
X-Offset
Virtual
Address
X-Length
Y-Length
Z-Length
Pages Per
Node
Bit-Field
Extractor
118
118
118
122
Y-Offset
X-Dest
Y-Dest
Z-Dest
Z-Offset

FIG. 16

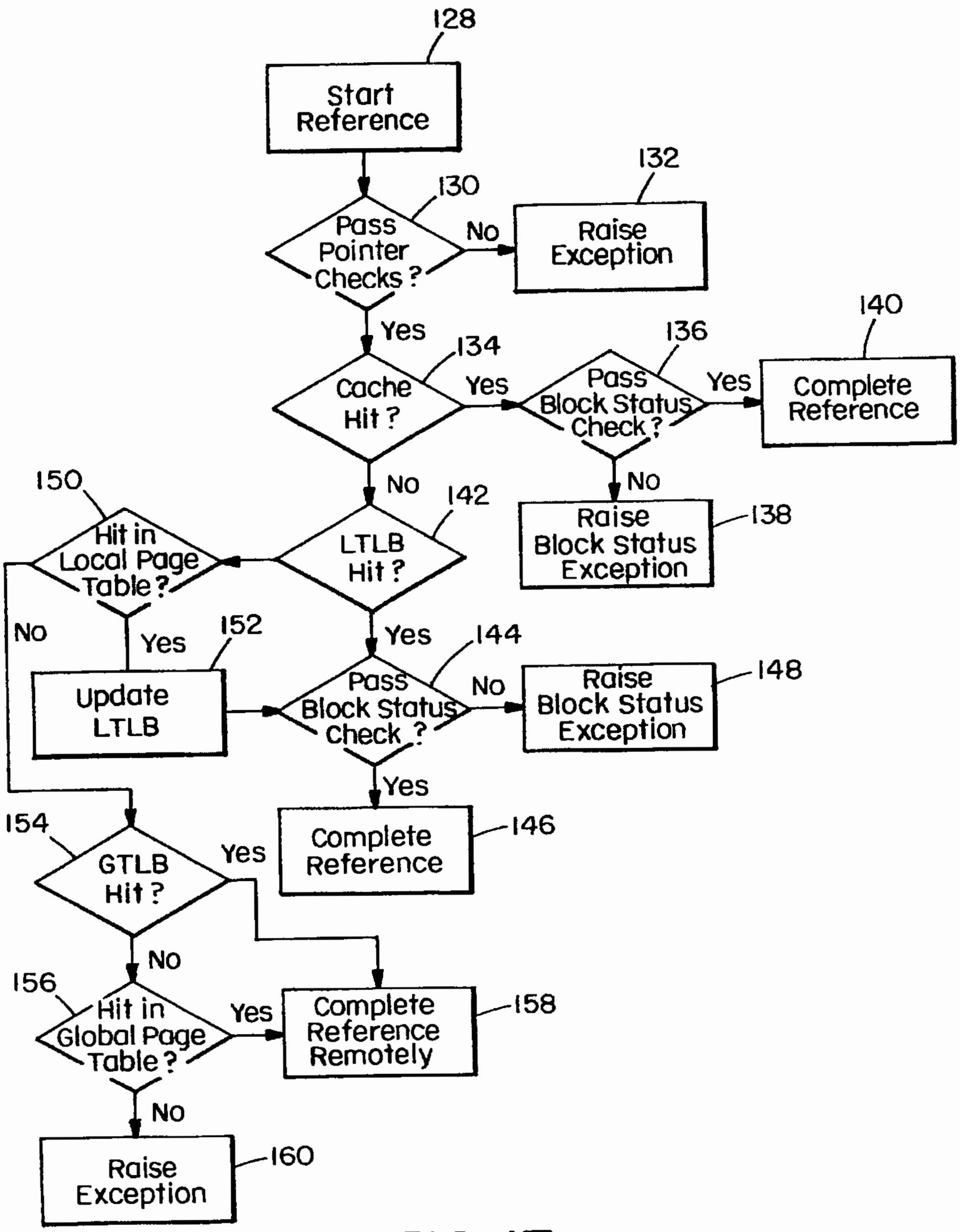

128
Start Reference
130
Pass Pointer Checks ?
No
132
Raise Exception
Yes
134
Cache Hit ?
Yes
136
Pass Block Status Check ?
Yes
140
Complete Reference
No
138
Raise Block Status Exception
No
142
LTLB Hit ?
150
Hit in Local Page Table ?
No
Yes
152
Update LTLB
Yes
144
Pass Block Status Check ?
No
148
Raise Block Status Exception
Yes
146
Complete Reference
154
GTLB Hit ?
Yes
No
156
Hit in Global Page Table ?
Yes
158
Complete Reference Remotely
No
160
Raise Exception

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 mutiprocessor," *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 over 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 place 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^{54} 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^{54} 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+a[i];
```

In a conventional system, eacb 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 Rossum, 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 of two words in length and aligned. Internal fragmentation may result when the space needed by an object must be rounded up to the next power of two words. However, this does not result in much wasted physical memory, since physical space is allocate on a page-by-page basis, independent of segmentation. External fragmentation of the virtual address space may occur when recycled segments cannot be coalesced into contiguous sections of usable sizes. A buddy memory allocation scheme, which combines adjacent free segments into larger segments, can be used to reduce this fragmentation problem.

Software Implementations

While guarded pointers enable efficient implementation of many desirable operating system features, some shortcomings inherent in single-address-space and capability-based architectures can be addressed by the software system designer using guarded pointers.

The efficiency of guarded pointers is largely due to eliminating indirection through protected segment tables. With guarded pointers there is no need to store these tables or to access them on each memory reference. Without protected indirection, modifying a capability requires scanning the entire virtual address space to update all copies of the capability. This is needed, for example, when relocating a segment or revoking access rights to a segment. In some cases this expensive operation can be avoided by exploiting the paging translation, user-level indirection or protected subsystems.

All guarded pointers to a segment can be simultaneously invalidated by unmapping the segment's address space in the page table. All subsequent accesses using pointers to this segment will raise exceptions. This directly revokes all capabilities to a segment. Segments can be relocated by updating the pointer causing the exception on each reference to the relocated segment. One limitation of this approach is that it operates on a page granularity while segments may be any size, down to a single byte in length. Thus relocating or revoking access to a segment may affect the performance of references to several unrelated bystander segments.

Indirection can be performed explicitly in software where it is required. If a segment's location is unknown or is expected to move frequently, a program can make all segment references to offsets from a single segment base pointer. Only this single pointer needs to be updated when the segment is moved. With explicit indirection, overhead is incurred only when indirection is needed, and then it is exposed to the compiler for optimization. Since no hardware prevents user code from copying the segment base pointer, relocation or revocation through explicit indirection requires adherence to software conventions.

It is impossible in any capability-based system to directly revoke a single process' rights to access a segment without potentially affecting other processes. Since possession of a capability confers access rights, the only way to remove access rights from a single process is to remove all capabilities containing those access rights from the memory addressable by the process. This can be accomplished by sweeping the memory that the process can address, and overwriting the correct capabilities, so long as none of the memory containing those capabilities is shared. If the pointers that need to be overwritten are contained within a shared segment, all processes which rely on the pointer will lose access privileges. This is due to the lack of a protected table that stores permission information on a per-process basis.

Protected indirection can be implemented by requiring that all accesses to an object be made through a protected subsystem. In addition to restricting the access methods for the object, the subsystem can relocate the object at will and can implement arbitrary protection mechanisms. For example, the subsystem could implement a per-process access control list. Revoking a single process' access rights can be performed by updating the access control list. Accessing an object through a protected subsystem is advisable if the object must be relocated or have its access rights changed frequently and if the object is referenced infrequently or only via the subsystem access methods.

Without indirection, address space is allocated "for all time," requiring the system software to periodically garbage collect the virtual address space, so that addresses no longer in service can be reused. This is simplified with guarded pointers, as pointers are self identifying via the tag bit. Thus, the live segments can be found by recursively scanning the reachable segments from all live processes and persistent objects.

The M-Machine

The M-Machine memory system provides an example of how guarded pointers may be used. The M-Machine is a multicomputer with a 3-dimensional mesh interconnect and multithreaded processing nodes (Dally, W. J., Keckler, S. W., Carter, N., Chang, A., Fillo, M., and Lee, W. S. "M-Machine architecture v1.0," Concurrent VLSI Architecture Memo 58, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, January 1994 and Keckler, S. W., and Dally, W. J., "Processor coupling: Integrating compile time and runtime scheduling for parallelism", *Proceedings of the 19th International Symposium on Computer Architecture* (Queensland, Australia, May 1992), ACM, pp 202–213, and U.S. application Ser. No. 08/062,388). One of the major research goals of the M-Machine is to explore the best use of the increasing number of transistors that can be placed on a single chip.

The processing nodes of the M-Machine (known as multi-alu processors, or MAPs) operate on 64-bit integer and floating-point data types and use 64-bit guarded pointers (plus a tag bit) to access a 54-bit, byte-addressable, global address space, which is shared by all processes and nodes of the machine. FIG. 8 shows a block diagram of a MAP chip. Each MAP chip contains twelve execution units: four integer units, four floating-point, and four memory units. These execution units are grouped into four clusters 69, each containing one execution unit of each type.

To increase the utilization of these hardware resources when executing programs that have insufficient instruction-level parallelism, the M-Machine implements multithreading. Four user threads share the processing resources of each cluster, for a total of sixteen user threads in execution at any time. Each cycle, the hardware on each cluster examines the threads in execution on it and selects one thread to execute on the hardware resources. The three execution units in a cluster are allocated and statically scheduled as a long instruction word processor.

Each M-Machine node contains an on-chip 4-bank cache 70 and 1MWord (8MBytes) of off-chip memory 71. The cache is virtually addressed, and addresses are interleaved across the banks. This allows the memory system to accept up to four memory requests during each cycle, matching the peak rate at which the processor clusters can generate requests. Requests that miss in the cache arbitrate for the external memory interface 72, which can only handle one request at a time. The interface 72 also holds the LTLB. Request to memory are made by cluster 69 through an M-switch 73, and responses are passed back through a C-switch 75. Transfers between clusters are also made through the C-switch.

Messages are routed through the network by an output interface 77 using the GTLB 79. Incoming messages are queued in an input interface 81.

The M-Machine presents two challenges to a protection system. The first is cycle-by-cycle interleaving of instructions and memory references from different protection domains, while still allowing efficient sharing among them. Because guarded pointers provide memory protection without requiring each thread to have its own virtual to physical translations, memory references from different threads may be in flight simultaneously without comprising security. This enables zero cost context switching as no work must be performed to switch between protection domains.

The other challenge for both the protection and translation systems is the interleaved cache of the M-Machine, which may service up to four references simultaneously. The single address space implemented with guarded pointers allows the cache to be virtually addressed and tagged so that translations need only to be performed on cache misses. In addition, encoding all protection information in a guarded pointer eliminates any need for table lookup prior or during cache access. These two features eliminate the need to replicate or quad-port the TLB or other protection tables.

Guarded Pointer Conclusions

We have introduced guarded pointers as a hardware mechanism to implement capability-based protection and allow fast multithreading among threads from different protection domains, including concurrent execution of user programs and the operating system. We have described the M-Machine as an example of an architecture which implements guarded pointers.

A guarded pointer is an unforgeable handle to a segment of memory. Each pointer is comprised of segment permission, length, base, and offset fields. The advent of 64-bit machines allows this information to be encoded directly in a single word, without unduly limiting the memory address space. An additional tag bit is provided to prevent a user from illicitly creating a guarded pointer. Guarded pointers are an efficient implementation of capabilities without capability tables and mandatory indirection on memory access.

Guarded pointers can be used to implement a variety of software systems. Threads in different protection domains can share data merely by owning copies of a pointer into that segment. A thread can grant another thread access to private data by passing a guarded pointer to it. Protected entry points and cross-domain calls can be efficiently implemented using an entry type guarded pointer.

The costs of implementing guarded pointers are minimal. An additional tag bit is required to identify pointers, and the virtual address space is reduced by the number of bits required to encode segment permissions and lengths. In a 64 bit machine, 54 virtual address bits are left, which is ample space for the foreseeable future. A small amount of hardware is also required to perform permission checking on memory operations.

Like all single global virtual address space systems, guarded pointers permit processes from different protection domains to share the cache and paging systems without comprising security. Also like these systems, guarded pointers eliminate multiple translations and permit processes to access an interleaved virtual cache without requiring multiple TLBs. Guarded pointers do share some of the deficiencies of single address space memory systems (garbage collecting virtual address space), and capability systems (relocating and revoking access to segments).

By encoding a segment descriptor in the pointer itself and checking access permissions in the execution unit, guarded pointers obviate the need to check protection data in the cache bank. This permits in-cache sharing, which is not possible with methods that append the PID to the cache tag, without the expense of providing protection tables in hardware.

Consequently, guarded pointers concentrate process state in general purpose registers instead of auxiliary or special memory. Threads become more agile as less processor resident state is needed. This will enable better resource utilization in parallel systems as threads may begin execution, migrate and communicate with other threads with lower latency.

Block Status Bits

The addition of block status bits to a memory system allows relocation of data objects that are smaller than individual pages, without requiring a lookup table entry for each object. Each page of memory (4 KB) is divided into 64-byte (8 word) blocks. Two block status bits are assigned to each of the 64 blocks in a page. The status bits are used to encode the following states:

INVALID: Any attempt to reference the block raises an exception.

READ ONLY: The block may be read, but an exception occurs if a write is attempted.

READ/WRITE: Reads and writes to the block are permitted.

DIRTY: Reads and writes to the block are permitted. The line has been written at least once since the page table entry was created.

One method in which block status bits may be used to control the relocation of data is to assign each block in the memory a home node, which is responsible for managing the relocation of the blocks assigned to it. A mechanism such as the GTLB may be used to provide fast location of the home node of a block, but this is not necessary.

The home node maintains a software record of which other nodes have copies of a block, and the status of those copies. Only one node is allowed to have a copy of a block that is in the read-write state, but many nodes may have read-only copies of a block if no node has a read-write copy. This prevents different nodes from having different versions of the data in a block.

When a node requests a read-only copy of a block, the home node examines its records of which nodes have copies of the block. If no node has a read-write copy of the block, the home node issues a read-only copy of the block to the requesting node, and adds the requesting node to the list of nodes that have a copy of the block. If another node has a read-write copy of the block, the home node sends an invalidate message to the node, telling it to give up its copy of the block, and to inform the home node of the new contents of the block if the block has changed. When the home node receives notification that the read-write copy of the block has been invalidated, it issues the read-only copy of the block to the requesting node and records that the requesting node has a copy of the block.

Requests for read-write copies of a block are handled in the same manner, except that any node that has a copy of the block must invalidate its copy before the read-write copy can be given out, to prevent data inconsistency problems.

When a node receives a message telling it to invalidate its copy of a block, it examines the block status bits of that block. If the block is in a read-only or read-write state, then the node has not changed the contents of the block, and the block can be discarded and the home node informed that this has been done. If the block is in the dirty state, then its contents have been changed since the node received its copy of the block, and the node must send the changed copy of the block back to the home node before its discards the block.

When a data word is accessed in the memory, the block status bits corresponding to that word are retrieved as well as the word being accessed. The block status bits are compared to the operation being attempted; and an exception is raised if any operation is attempted on a word whose block status bits are in the invalid state, or if an operation that modifies memory is attempted on a word whose block status bits are in the read-only state. If an operation is not allowed, the operation is cancelled before it modifies the state of the memory. If the operation modifies the location being referenced, the block status bits corresponding to that location are set to "dirty" if the operation is allowed. This allows the hardware to quickly determine if a block has been modified, as any modifications to a block will cause its status bits to enter the dirty state.

The block status bits for each mapped page on a node are contained in the local page table of that node. When the translation for a page is brought into the local translation lookaside buffer (LTLB), the status bits for the blocks contained in that page are copied into the LTLB as well. When a block of data is brought into the cache from the main memory, the block status bits for that block are examined in the LTLB. The cache status of the block is set to read-only if the block status in the LTLB entry is read-only. If the LTLB block status is read/write or dirty, then the cache status is set to read/write. Attempts to bring a block in the invalid state into the cache causes an exception. The dirty bit of a block's 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. This does not change the status of the block in the LTLB. When a block is evicted from the cache, its dirty bit is examined, and the status of the block in the LTLB changed to dirty if the cache dirty bit is set to one. When an LTLB entry is evicted, its block status bits are simply copied out to the local page table, as the LTLB entry contains the most recent copy of the status bits.

FIG. 9 shows the format of an LTLB entry, while FIG. 10 shows the transfers of status bits between storage locations, FIG. 11 shows the hardware that extracts the status bits for a block from the LTLB, and FIG. 12 is a flow chart of a memory request using the block status bits.

As shown in FIG. 9, an entry for each virtual page in the local page table and local table lookup buffer comprises three words. The first word includes the translation from virtual page to physical page. The virtual page is identified by the first 42 bits of the 54-bit virtual address. Since the translation to physical address is only for the physical space on a particular node, 20 bits are sufficient to identify the physical page location. The second and third words each include a single bit for each of 64 blocks of the virtual page.

As shown in FIG. 11, the first 42 bits of the virtual address are used to locate the page table entry n the LTLB 71 and three words for that entry are output as shown. To select the appropriate block status bits, the next 6 bits of the virtual address, which are the first 6 bits of the page offset, are applied to the select inputs of multiplexers 73 and 75, each selecting one of the two block space bits for that virtual address.

Caching the block status bits in the LTLB and in the cache allows the memory system to examine a word's block status bits when that word is referenced without requiring a page table access on each memory reference. FIG. 12 shows the sequence of events involved in performing a memory access in a system that implements block status bits. First (not shown on the flow chart) any permission checks that are necessary to determine whether or not the user is allowed to access the address in question are performed. This includes all of the procedures of FIG. 2A if the system incorporates Guarded Pointers.

Once that has been done, the request is submitted at 74 to the cache memory 77 (FIG. 10). If the address is found in the cache at 76, the block status bits corresponding to the address are examined and compared to the operation being performed at 78 and 80. If the operation is allowed, the cache memory completes the operation at 82 and is ready for the next request. If the operation is not allowed, an exception is raised at 84.

If the address is not in the cache 76, the local translation lookaside buffer (LTLB) 79 is probed at 86 to determine if it contains a translation for the address. If the LTLB does not contain a translation, an exception occurs at 88 to check the local page table 81, and software is invoked at 90 to load a translation into the LTLB from the local page table. As shown in FIG. 10, the LTLB entry which is evicted carries with it status bits for updating those bits in the local page table. Similarly, the new entry carries the status bits from the local page table. When the data is read into the cache memory 77, the status bits for the cache line are copied from the associated entry of the LTLB, with the exception that a dirty entry is entered in the cache as a read/write. The dirty designation is retained in the LTLB for purposes of providing the dirty flag to a home node when requested. However, the operating program which loads from the cache need only determine whether it is authorized to read or write. Within the cache, the status bit will be converted to dirty with a write to cache in order to facilitate updating the status bits in the LTLB and the data in memory with later eviction of the cache line.

Once a translation has been found, either in the page table or the LTLB, the block status bits corresponding to the address are compared at 92 and 94 to the operation being performed. If the block status bits allow the operation being attempted, the operation is completed from the main memory at 96. If the block status bits do not allow the operation, an exception is raised at 98.

If no translation for the address can be found in either the LTLB or the local page table, the software attempts at 100 to locate the data on another node, possibly using a GTLB as described below.

The operating system must have the ability to change the status bits of a memory block. This can be provided either through privileged operations that probe the cache to change the status bits in the cache as well as in the LTLB entry, or by requiring the system to remove the appropriate block from the cache before altering its status bits, and to ensure that the block is not returned to the cache before the status bits have been updated.

These states allow a variety of relocation and replication (cache coherence) schemes to be implemented efficiently, by handling the common case (the user attempting an access which is allowed) in hardware while giving the software the ability to determine how illegal accesses are handled. For example, block status bits allow the efficient implementation of a system in which small data objects are relocated from node to node. When a data object is brought onto a node, a page table entry is created for the page containing that object if one does not already exist. The status bits for the memory blocks containing the object being relocated are set to one of the three valid states, while the status bits for each memory block that does not contain valid data on the local node are set to INVALID. Users can then access the object in any way that is consistent with the status bits associated with it. If a user attempts to reference a block that has not been brought on to the local node, its status bits will be in the INVALID state, and any attempt to reference it will cause an exception, invoking an exception handler to resolve the situation. Moving an object off of a node can be accomplished by copying it to another node, and changing the status bits associated with it to INVALID, prohibiting access to the object. This allows small data objects to be relocated throughout a multicomputer efficiently without requiring overly large tables to contain information about which objects are located on a given node. The system will have to maintain a table in software which contains information on where each object is in the system, but the space constraints on software tables are not nearly as great as on hardware tables.

Block status bits can also be used to implement cache coherence schemes. Many cache-coherence schemes assign states to data which are very similar to the block status states: INVALID, READ-ONLY, READ-WRITE, and DIRTY. The differences between these schemes lie in their handling of cases where data is referenced in a manner which is inconsistent with its state. Block status bits allow the hardware to handle the (common) case where data is accessed in an allowed manner, with software being invoked to handle the uncommon case where an illegal access is attempted. Since system software can manipulate the status bits of a block, operations such as system-wide invalidation of a block so that one node can gain an exclusive copy of the block, can be efficiently implemented.

Global Translation Lookaside Buffer

A Global Translation Lookaside Buffer (GTLB) is used to cache translations between virtual addresses and the nodes containing those addresses. Translation of virtual addresses to physical addresses is handled by a Local Translation Lookaside Buffer (LTLB) which may essentially be the same as a conventional translation lookaside buffer. The intended use of the GTLB is to allow hardware and software to quickly determine which node of a multicomputer contains a given datum. A message can then be sent to that node to access the datum. On the node that contains the datum, the LTLB can be used to translate the virtual address into a physical address in order to reference the datum.

In order to allow large blocks of virtual address space to be mapped by a small number of GTLB entries without increasing the size of the smallest block of data that can be mapped, each GTLB entry maps a variable-sized page-group of virtual address space across a number of nodes. In order to simplify the interaction between the local and global translation mechanisms, and to reduce the number of bits required to encode the length of a page-group, each page group must be a power of two local pages in length.

The address space contained in a page-group may be mapped across a 3-D sub-cube of nodes, with the following restrictions: each side of the sub-cube must be a power of two nodes long, and the amount of address space allocated to each node must be a power of two local pages. While these restrictions constrain the mapping of address space to nodes somewhat, they greatly reduce the size of the GTLB entry and the complexity of the hardware needed to perform the translation.

FIG. 13 shows the format of a GTLB entry. 42 bits encode the virtual page identifier, which is obtained by truncating the low 12 bits off a 54-bit virtual address, since these bits represent the offset within a local page. Sixteen bits encode the start node of the sub-cube of nodes that the page-group maps across. This node ID is divided into a six-bit Z-Coordinate, and 5-bit X- and Y-coordinates to give the position of the start node within the machine. Six bits encode the base-2 logarithm of the length of the page-group in local pages. Six bits encode the base-2 logarithm of the number of local pages of address space to be placed on each node. Three bits encode the base-2 logarithm of the length of the prism of nodes that the page-group maps across in each of the X-, Y-, and Z-dimensions.

FIGS. **14A**, **14B**, **15A** and **15B** show the manner in which the GTLB translates a virtual address. The virtual address is submitted to the GTLB at **102**. If a hit is not located at **104**, a miss is signalled at **106** to call an exception which reads the global page table. FIG. **15A** illustrates an example GTLB entry located with a hit.

Since the GTLB is fully associative, the page identifier portion of each virtual address, that is, the first 42 bits of each virtual address, must be compared to the virtual page identifier of each entry in the GTLB. Further, since the grouping of pages allows for a single GTLB entry for each page group, the least significant bits of the page identifier corresponding to the number of pages in the group need not be considered in the comparison. Thus, as illustrated in FIG. **14B**, the six bits of each GTLB entry which indicate the number of pages per group can be decoded to create a mask in bit mask generator **124**. Using the bit mask generator **124**, only the more significant bits of the page identifiers required to identify a group are compared in the mask comparator **126**. On the other hand, the full 42 bits of both the virtual address and the GTLB entry are applied to the comparator since groups can be of different lengths and thus require masking of different sets of bits. Applying the full 42 bit identifiers to the comparator allows for a group of only one page.

From the entry illustrated in FIG. **15A**, it is determined that the start node of the sub-cube is node **[3,2,0]** and that 2^4 pages of address space are mapped to each node within the sub-cube. The page-group is mapped across a sub-cuhe of nodes that extends 22 nodes in the Z-direction, 22 nodes in the Y-direction, and 2 nodes in the X-direction. The start node **[3,2,0]** and the full cubic group of nodes is illustrated in FIG. **15B**.

To determine the node containing the address being translated, the GTLB masks off at **108** the page offset bits of the address which contain the offset from the start of the local page to the address being translated. The next four bits of address **0101** are discarded, as they all map to the same node, as shown by the value 4 in the "log pages per node" filed. The next bit of the address contains the X-offset from the start node to the node containing the address, as shown by the value of 1 in the X subfield of the "log sub-cube dimensions" field, and that bit is extracted at **110**. Similarly, two bits contain the Y-offset and two bits contain the Z-offset from the start node to the node containing the address being translated, and those are extracted at **112** and **114**. Examining the selected bit fields reveals that the node containing the address lies at offset X=1, Y=2, Z=3 from the start node. Adding these values to the coordinates of the start node at **116** in the address **118** gives the coordinates of the node containing the address X=1, Y=4, Z=6, shown in FIG. **15B**.

FIG. **16** shows a block diagram of the GTLB hardware. The GTLB comprises a content addressable memory CAM **120** which contains the GTLB entries, a bit-field extractor **122** to extract the X-, Y-, and Z-Offset fields from the source address, and three adders **118** to add the offsets to the appropriate portions of the start node. The SRAM array must be fully-associative, as the variable size of page-groups makes it impossible to use a fixed number of bits to select a set within the array to be searched. When an address is submitted to the GTLB for translation, it is passed to the CAM array. If the address is found in the array, the Hit output is asserted, and the start node, the page-group length, the pages-per-node information, and the X-, Y-, and Z-lengths of the sub-cube of nodes containing the address being translated are outputted. The bit-field extractor takes the dimension of the prism, and the page-length and pages-per-node information, and extracts from the virtual addresses the bit fields containing the X-, Y-, and Z-offsets from the start node of the page-group to the node containing the address being translated. The offsets are then added to the appropriate field within the address of the start node to get the address of the node containing the address.

Integration of all Three Systems

FIG. 17 shows a flow chart of the execution of a memory reference from 128 in a system that combines Guarded Pointers, Block Status Bits, and a Global Translation Lookaside Buffer. The first step in performing a memory operation is to perform at 130 the pointer permission checks described in the section on Guarded Pointers. If those checks pass, the memory request is sent to the memory system. Otherwise, an exception is raised at 132.

If the data is located in the cache at 134, its block status bits are examined at 136, and an exception is raised at 138 if they do not allow the operation being attempted. Otherwise, the operation is completed in the cache at 140. If the data is not located in the cache, the LTLB is probed at 142 for a translation for the address. If a translation is found, the block status bit of the address are examined at 144, and the operation completed from the main memory at 146 if the status bits allow it, or an exception raised at 148 if they do not.

If a translation for the address is not found in the LTLB at 142, software searches the local page for a translation at 150. If a translation is found, the LTLB is updated at 152 to contain the translation, and the reference proceeds at 144 as if an LTLB hit occurred.

If no translation is found in the local page table at 150, the software probes the GTLB at 154 to see if the node containing the address can be determined. If a GTLB miss occurs, the global page table is searched at 156 for an entry corresponding to the address. If the node containing the address can be located either through the GTLB or the global page table, the software can send a message to that node to complete the request at 158. Otherwise, an error is signalled at 160, as the reference can not be completed.

While each of these mechanisms is useful separately, they complement each other to form the basis for the memory system of a multicomputer. Guarded Pointers provide a protection mechanism that allows a number of independent processes to share the resources of the multicomputer without compromising the security of those processes. The Global Translation Lookaside Buffer provides an effective mechanism for distributing data objects across the multicomputer by mapping virtual addresses to nodes within the multicomputer. The block Status Bits make the process of moving or copying data from node to node more efficient by reducing the size of the smallest datum that can be relocated, without increasing the number of translation table entries required if no remote data is accessed.

A related paper has been submitted for presentation at the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI), Oct. 5–6, 1994.

EQUIVALENTS

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. Those skilled in the art will recognize or be able to ascertain using no more than routine experimentation, many equivalents to the specific embodiments of the invention described specifically herein. Such equivalents are intended to be encompassed in the scope of the claims.

What is claimed is:

1. In a parallel processing system, a method of addressing data across plural processor nodes comprising:

applying a virtual address to a global translation buffer to identify a mapping of a page group of plural pages across a set of plural but less than all processor nodes in the system, the page group containing the physical page to which the virtual address corresponds; and

from the virtual address and mapping, determining a destination node as a node within the set of processor nodes which contains the physical page to which the virtual address corresponds.

2. A method as claimed in claim 1 further comprising forwarding a message to the destination node.

3. A method as claimed in claim 2 further comprising, at the destination node, translating the virtual address to a physical address.

4. A method as claimed in claim 1 wherein each page group is specified by a group size.

5. A method as claimed in claim 4 wherein the group size is logarithmically encoded.

6. A method as claimed in claim 1 wherein the translation buffer specifies a start node and the range of the set of nodes.

7. A method as claimed in claim 6 wherein the range is specified in plural dimensions.

8. A method as claimed in claim 7 wherein the range is logarithmically encoded in each of the plural dimensions.

9. A method as claimed in claim 8 wherein the translation buffer specifies the number of pages of the page group per node of the set of nodes.

10. A method as claimed in claim 6 wherein the translation buffer specifies the number of pages of the page group per node of the set of nodes.

11. A method as claimed in claim 1 wherein the translation buffer specifies the number of pages of the page group per node of the set of nodes.

12. A data processing system comprising a plurality of processor nodes, each processor node comprising:

a global translation buffer for identifying relative to a virtual address a mapping of a page group of plural pages to a set of plural processor node s in the system, the page group containing the physical page to which the virtual address corresponds;

electronics which determines, from the virtual address and the identified mapping, a destination node as a node within the set of processor nodes having the physical address corresponding to the virtual address.

* * * * *

US005530834A

United States Patent [19]

Colloff et al.

[11] **Patent Number: 5,530,834**

[45] **Date of Patent: Jun. 25, 1996**

[54] **SET-ASSOCIATIVE CACHE MEMORY HAVING AN ENHANCED LRU REPLACEMENT STRATEGY**

[75] Inventors: **Ian G. Colloff**, Ascot; **Albert S. Hilditch**, Wokingham, both of England

[73] Assignee: **International Computers Limited**, Putney, United Kingdom

[21] Appl. No.: **206,001**

[22] Filed: **Mar. 3, 1994**

[30] **Foreign Application Priority Data**

Mar. 30, 1993 [GB] United Kingdom 9306647

[51] **Int. Cl.**[6] **G06F 12/12**

[52] **U.S. Cl.** **395/463**; 395/421.06; 395/487

[58] **Field of Search** 395/200, 400, 395/425, 421.06, 463, 445, 486, 487

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,949,369 4/1976 Churchill, Jr. 340/172.5

4,511,994 9/1982 Webb et al. 395/487

FOREIGN PATENT DOCUMENTS

1087189 10/1967 United Kingdom .

1475785 6/1977 United Kingdom .

Primary Examiner—Jack A. Lane

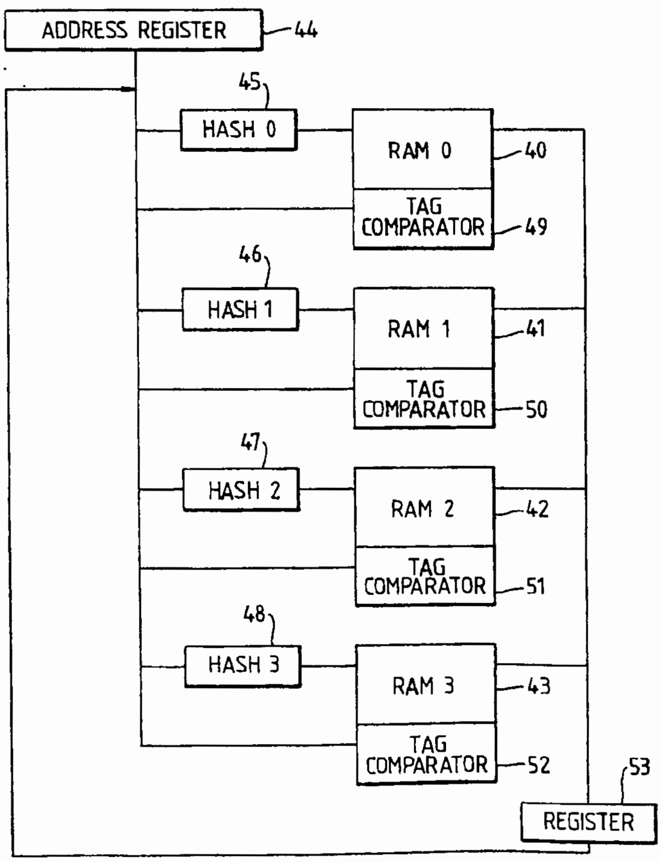
Assistant Examiner—Fadi A. Stephan

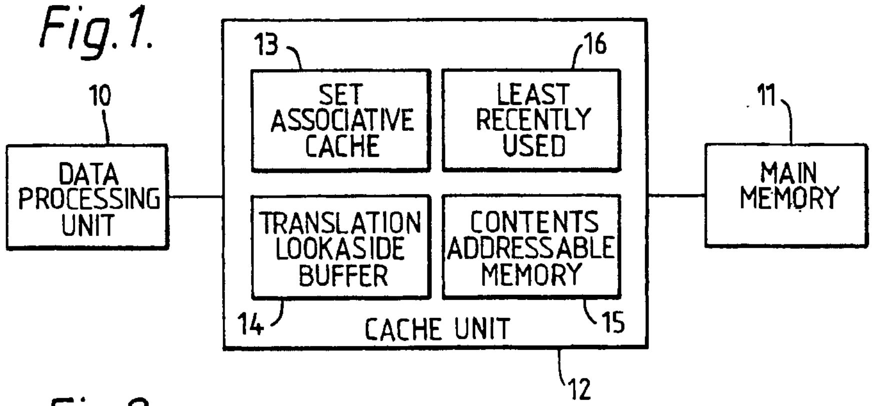
Attorney, Agent, or Firm—Lee, Mann, Smith, McWilliams, Sweeney & Ohlson

[57] **ABSTRACT**

A cache memory contains a number of RAMs. The RAMs are addressed by independent hashing functions, so as to access a set of locations, one in each RAM. If the required data item is resident in the addressed set, it is accessed. Otherwise, the least-recently used location in the set is selected for overwriting with data from main memory. The contents of the RAM location that is about to be overwritten are saved, and then used to access the memory again in order to address a further set of locations. If any of this further set of locations is less recently used than the saved contents, the saved contents are loaded back into that location.

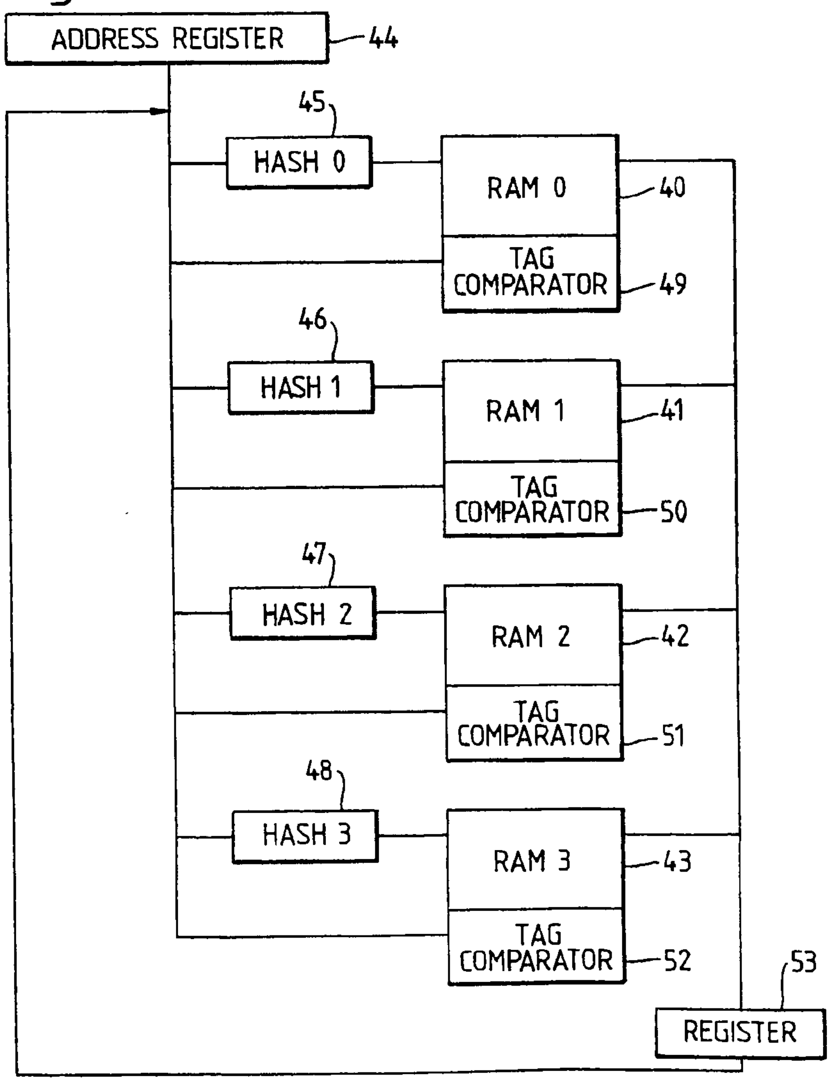
3 Claims, 3 Drawing Sheets





10
DATA PROCESSING UNIT
13
SET ASSOCIATIVE CACHE
16
LEAST RECENTLY USED
TRANSLATION LOOKASIDE BUFFER
CONTENTS ADDRESSABLE MEMORY
14
CACHE UNIT
15
12
11
MAIN MEMORY

Fig.1.



ADDRESS REGISTER
44
45
HASH 0
RAM 0
40
TAG COMPARATOR
49
46
HASH 1
RAM 1
41
TAG COMPARATOR
50
47
HASH 2
RAM 2
42
TAG COMPARATOR
51
48
HASH 3
RAM 3
43
TAG COMPARATOR
52
53
REGISTER

Fig.2.

Fig. 3.

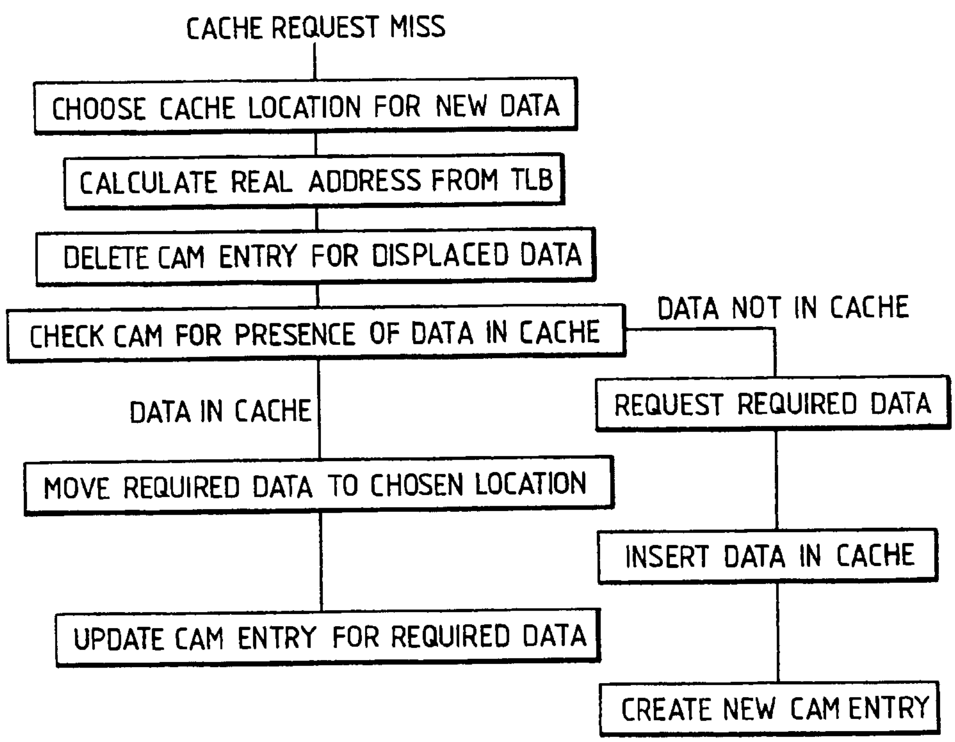

CACHE REQUEST MISS
CHOOSE CACHE LOCATION FOR NEW DATA
CALCULATE REAL ADDRESS FROM TLB
DELETE CAM ENTRY FOR DISPLACED DATA
CHECK CAM FOR PRESENCE OF DATA IN CACHE
DATA NOT IN CACHE
DATA IN CACHE
REQUEST REQUIRED DATA
MOVE REQUIRED DATA TO CHOSEN LOCATION
INSERT DATA IN CACHE
UPDATE CAM ENTRY FOR REQUIRED DATA
CREATE NEW CAM ENTRY

Fig. 4.

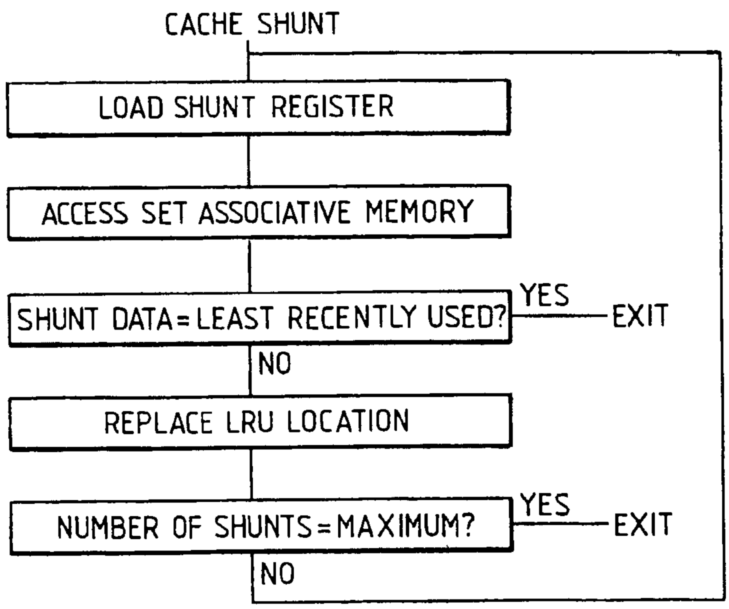
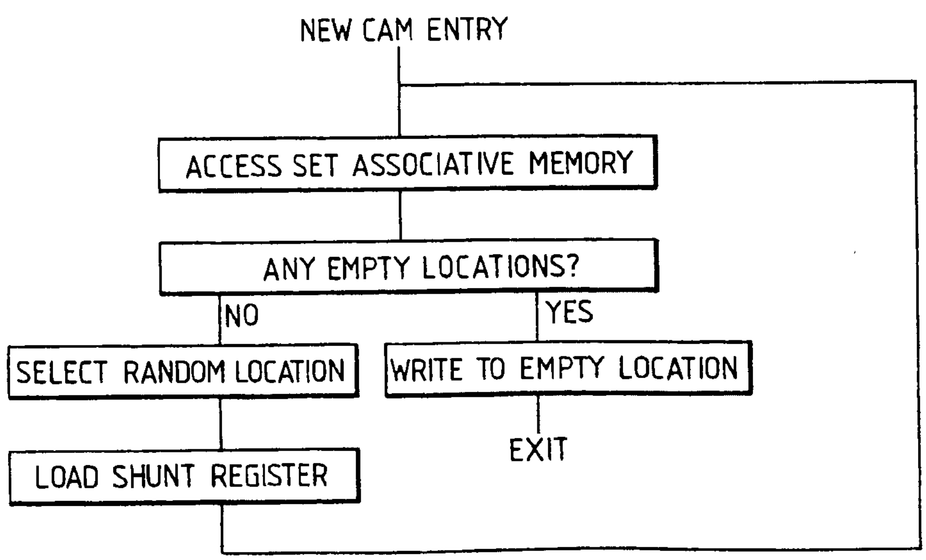

CACHE SHUNT
LOAD SHUNT REGISTER
ACCESS SET ASSOCIATIVE MEMORY
SHUNT DATA=LEAST RECENTLY USED?
YES
EXIT
NO
REPLACE LRU LOCATION
NUMBER OF SHUNTS=MAXIMUM?
YES
EXIT
NO

Fig. 5.


NEW CAM ENTRY
ACCESS SET ASSOCIATIVE MEMORY
ANY EMPTY LOCATIONS?
NO
YES
SELECT RANDOM LOCATION
WRITE TO EMPTY LOCATION
EXIT
LOAD SHUNT REGISTER

SET-ASSOCIATIVE CACHE MEMORY HAVING AN ENHANCED LRU REPLACEMENT STRATEGY

BACKGROUND OF THE INVENTION

This invention relates to set-associative memories.

One conventional form of set-associative memory comprises a plurality of random access memories (RAMs), each RAM location holding a data item and a tag value identifying the data. An input address is hashed (i.e. transformed by a many-to-one mapping function) to produce a hash address, which is applied in parallel to all the RAMs, so as to select one location in each RAM. The tag values in the addressed locations are then examined to see if the desired data is resident in one of them and, if so, the data is accessed.

If there are n RAMs, so that n locations at a time are examined, the memory is referred to as an n-way set-associative memory and is said to have an associativity of n. The usual choice for the value of n is 2 or 4.

Such a set-associative memory may be used, for example, as a cache memory for a computer system. The aim of a cache is to keep the most useful data of a large amount of data in a small, fast memory in order to avoid having to retrieve the data from the larger, slower main memory. If the required data is in the cache, it is said that a "hit" has occurred; otherwise a "miss" has occurred. The percentage of misses is called the "miss rate". A common engineering problem in designing a cache is to minimize the miss rate while keeping the cache size, the access speed, the power consumption and the amount of implementation logic fixed.

In general, the miss rate of such a cache decreases as its set associativity increases. On the other hand, the cost of implementation increases as set associativity increases. Thus, in general, known caches that deliver minimum miss rates demand large amounts of logic and space to implement, while known caches that are the cheapest to implement deliver higher miss rates.

Another use of set-associative memory is to form a content addressable memory (CAM). The aim of a CAM is to store and reference data according to its contents. For instance, performing a join of two relations within a relational database query can be implemented by first storing the contents of one relation in the CAM, indexed by the join attribute, and then secondly by comparing the rows of the second relation to the CAM using the join attribute again. Content addressable memories can be implemented by fully associative memories but their size is limited by the space demanded by fully associative logic.

One object of the present invention is to provide an improved set-associative memory which is capable of performing as well as conventional set-associative memories of higher set associativity, or better than conventional set-associative memories of the same set associativity. For example, when the set-associative memory is used as a cache, this means that it is able to deliver the same miss rate as conventional caches of larger size and cost, or lower miss rates than conventional caches of the same size and cost.

A second object of the present invention is to provide a CAM using a modified set-associative memory. This allows both much larger CAMs to be constructed and an improved read performance over present CAMs.

SUMMARY OF THE INVENTION

According to one aspect of the invention, there is provided an n-way set-associative memory (where n is an integer greater than 1), comprising a plurality of n RAMs, each RAM location holding a data item and a tag value identifying the data, addressing means for addressing the RAMs to access a set of locations, one in each RAM, and means for examining said set of locations to detect whether a required data item is resident in any of those locations, wherein the addressing means comprises means for performing n independent hashing functions to hash an input memory address into n separate addresses for respectively addressing said RAMs, characterised by means for saving the contents of a RAM location that is about to be overwritten, means for using the saved contents to access the memory again to address a further set of locations, and a means for loading the saved contents into one of said further set of locations.

As will be shown, this "shunting" operation can improve the performance of the set-associative memory, by effectively increasing its set associativity.

According to a second aspect of the invention there is provided a contents addressable memory comprising a plurality of n RAMs (where n is an integer greater than 1), each RAM location holding a data item and a tag value identifying the data, means for performing n independent hashing functions to hash an input memory address into n separate addresses, means for addressing the RAMs with said n separate addresses to access a set of locations, one in each RAM, a means for examining said set of locations to detect whether any of said addressed set of locations is empty and, if so, loading an input data item into that location and a means operative if none of said addressed set of locations is empty, for selecting one of said addressed set of locations for replacement, saving the tag value and data item of the selected location, loading the input data item into the selected location, using the saved contents to access the memory again to address a further set of locations and, if any of the addressed set of locations is empty, loading the saved data item into that location.

As will be shown, a set-associative memory with repeated shunting can deliver a content addressable memory without the need for full associativity thus reducing the logic needed and greatly increasing the size of CAM possible. Further, the read performance of such a "repeated shunting CAM" will be faster than an equivalent fully-associative CAM.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a data processing system including a cache comprising a set-associative memory in accordance with the invention.

FIG. 2 shows a set-associative memory with the enhancement of "shunting".

FIG. 3 is a flow chart showing the operation of the cache.

FIG. 4 is a flow chart showing the way that shunting is used in operation of the cache.

FIG. 5 is a flow chart showing the operation of a contents addressable memory using the set-associative memory of FIG. 2.

DESCRIPTION OF EMBODIMENTS OF THE INVENTION

A data processing system embodying the invention will now be described by way of example with reference to the accompanying drawings.

Referring to FIG. 1, the data processing system comprises a data processing unit 10, a main memory 11, and a virtually addressed cache controller 12 connected between the processing unit and main memory. The cache within the cache controller is smaller and faster than the main memory, and holds copies of the most recently used data items, allowing these items to be accessed by the processing unit without having to retrieve them from the slower main memory.

clm 4

The cache controller 12 comprises a 4-way set-associative cache 13, a translation look-aside buffer (TLB) 14, a contents addressable memory (CAM) 15, and a least-recently-used replacement mechanism (LRU) 16. The set-associative cache holds the cache data, indexed by the virtual address of the data. The TLB contains a virtual address to real address mapping, indexed by the virtual address, for allowing virtual addresses to be translated into real addresses. The CAM contains a real address to cache location number mapping, indexed by the real address, the purpose of which will be described later. The LRU contains recency-of-usage information for the data items held in the set-associative memory.

SET-ASSOCIATIVE MEMORY WITH SHUNTING

FIG. 2 shows the 4-way set-associative memory in more detail. The memory comprises four RAMs 40–43, each of which contains a plurality of addressable locations. Each RAM location holds a data item and a virtual address tag, identifying the data item.

An input virtual memory address is received in an address register 44. This input address is hashed in four different ways by four different efficient hashing functions 45–48 to produce four separate hash addresses. The hashing is done concurrently. A good implementation of the hashing functions can be achieved by using the random matrix hashing algorithm as described in British patent specification GB 2240413. This algorithm generates an arbitrary number of independent hashing functions which can be implemented easily and which allow hashing to be completed within a few simple gate delays.

The four hash addresses are used to address the four RAMs, so as to address four locations, one in each RAM. Because the hashing functions are independent, these four hash addresses will, in general, be different. The virtual address tags in the four addressed locations are compared with the input virtual address by means of comparators 49–52, to see whether any of these locations contains the desired data.

The set-associative memory also includes a register 53, referred to herein as the shunt register, the purpose of which will be described.

OPERATION

The operation of the cache is as follows. When the data processor requires to access a data item, it sends a request to the cache, specifying the virtual address of the required data. The virtual address is loaded into the address register 44, so as to address four locations in the RAMs. If any of the addressed locations contains the required data, a hit has occurred, and the required data can be accessed immediately from that location. The LRU is updated to reflect the usage of this data.

If on the other hand none of the addressed locations contains the required data, a miss has occurred. The operation of the cache in the event of a miss is shown in FIG. 3.

The LRU is accessed to decide which of the four addressed RAM locations is least recently used, and this location is selected for replacement with the desired data. The TLB is then consulted to calculate the real address of the required data. The entry in the CAM for the data to be replaced is deleted.

The CAM is then consulted, using the real address, to determine whether the required data is already resident in the virtual cache, in another cache location under a different virtual address. If the data is present in a different cache location, under a different virtual alias, it is moved to the required cache location, and the entry for that data in the CAM is updated to the new cache location number. If on the other hand the data is not present in the virtual cache under a different virtual alias, it is requested from the main memory using the real address obtained from the TLB.

When the required data has been fetched from the main memory it is stored in the replacement location of the set-associative memory, and a new entry is added to the CAM for the new data.

In the case of a cache miss, after the required data has been requested from the main memory, a shunting procedure is performed, as will be described with reference to FIG. 4. This shunting is performed while the required data is being retrieved from main memory.

Referring to FIG. 4, the first step of the shunting procedure is to load the existing contents of the least-recently used one of the four addressed locations (i.e. the location that will be overwritten by the requested data) into the shunt register 53.

The virtual address tag in the shunt register is then used to address the set-associative memory, in place of the input virtual memory address. Four RAM locations will therefore be accessed, one in each of the four RAMs. One of these locations is where the data was shunted from. However, in general, the other three locations will be different from those accessed by the input virtual memory address, because of the different hashing functions used to access the four RAMs.

The recency of usage of the data in these other three addressed RAM locations is then compared with that of the data in the shunt register. If the data in the shunt register is more recently used than any of those three RAM locations, the RAM location with the oldest access time is replaced with the contents of the shunt register. The existing contents of the RAM location are loaded into the shunt register.

The shunting procedure is repeated, using the new contents of the shunt register, up to a fixed number of times or until it is found that the shunted data is less recently used than the data in any of the addressed RAM locations.

It can be seen that, after shunting is completed, the cache location lost is the least recently used cache location of all those examined. This implies that with a 4-way set-associative cache, shunting once on each miss provides the equivalent of a 7-way set-associative cache. Repeating the shunt each time adds 3 to the effective set associativity.

CONTENTS ADDRESSABLE MEMORY

The set-associative memory shown in FIG. 2 may also be used as a contents-addressable memory (CAM) such as, for example, the CAM 15 of FIG. 1.

Since a CAM is only used to store a finite amount of data, we assume that the number of locations in the RAMs is enough to hold all needed data. This means that we never discard any data in the CAM. However, for the set-associative memory to be used efficiently as a CAM between 20 and 30% of the total locations in the CAM should be surplus to requirement. This means that the expected number of shunts is not greater than 1 and optimum efficiency is ensured.

Referring to FIG. 5, this shows the operation of the CAM when it is required to load a new data entry into the CAM.

The address of the data is hashed by the four hashing functions to access four RAM locations, one in each RAM. The four addressed locations are then examined to see if any of them is empty. If so, the new data entry is loaded into that location, and the process is complete.

If, on the other hand, none of the four addressed RAM locations is empty, one of these four locations is selected at random, and its contents are loaded into the shunt register 53. The address tag in the shunt register is then used to address the set-associative memory, in place of the original input address. A further three RAM locations will therefore be accessed together with the location from which the data was shunted. This shunting process is repeated until, eventually, an empty RAM location is found, and the new data entry is loaded into that location.

When the CAM is searched for data, the data will always be found in one of the four cache locations initially searched. When adding data to the CAM it may take one or more shunts in order to find an empty cache location, but an empty location will always be found eventually. Deletion of data can be achieved without the need of shunting. A special command is provided for clearing the CAM for reuse.

The CAM described above has a number of advantages over CAMs implemented using a fully associative memory: less logic, less power consumption and faster access times. This will allow much larger CAMs to be constructed than normally possible. The CAM described above has two advantages over CAMs implemented using standard hashing techniques that must resort to inefficient means for resolving hashing collisions: better space utilisation and faster access times.

We claim:

1. A memory system including a main memory and a faster, smaller cache memory, wherein said cache memory comprises:

a) a plurality of n RAMs (where n is an integer greater than 1), each RAM comprising a plurality of addressable locations;

b) hashing means for performing n independent hashing functions, to hash an input address into n separate addresses for addressing said RAMs;

c) LRU means for storing recency-of-use information for each location in said RAMs;

d) means for applying a memory address as an input to said hashing means, to access a first set of locations in said RAMs, one location in each RAM;

e) means for using said LRU means to select a least recently used one of said first set of locations;

f) means for applying data from said least recently used one of said first set of locations as a further input to said hashing means, to access a further set of locations in said RAMs, one location in each RAM; and

g) means for using said LRU means to select one of said further set of locations that is less recently used than said least recently used one of said first set of locations and for loading said data from said least recently used one of said first set of locations into said one of said further set of locations.

2. A data processing system including a data processing unit, a main memory, and a faster, smaller cache memory, wherein said cache memory comprises:

a) a plurality of n RAMs (where n is an integer greater than 1), each RAM comprising a plurality of addressable locations;

b) hashing means for performing n independent hashing functions, to hash an input address into n separate addresses for addressing said RAMs;

c) LRU means for storing recency-of-use information for each location in said RAMs;

d) means for applying a memory address as an input to said hashing means, to access a first set of locations in said RAMs, one location in each RAM;

e) means for using said LRU means to select a least recently used one of said first set of locations;

f) means for applying data from said least recently used one of said first set of locations as a further input to said hashing means, to access a further set of locations in said RAMs, one location in each RAM; and

g) means for using said LRU means to select one of said further set of locations that is less recently used than said least recently used one of said first set of locations and for loading said data from said least recently used one of said first set of locations into said one of said further set of locations.

3. A method of operating a memory system including a main memory and a faster, smaller cache memory, the cache memory comprising a plurality of n RAMs (where n is an integer greater than 1), and hashing means for performing n independent hashing functions to hash an input memory address into n separate addresses for addressing said RAMs, said method comprising the steps:

a) applying a memory address as an input to said hashing means, to access a first set of locations in said RAMs, one location in each RAM;

b) selecting a least recently used one of said first set of locations;

c) applying data from said least recently used one of said first set of locations as a further input to said hashing means, to access a further set of locations in said RAMs, one location in each RAM; and

d) selecting one of said further set of locations that is less recently used than said least recently used one of said first set of locations and loading said data from said least recently used one of said first set of locations into said one of said further set of locations.

* * * * *

US005749087A

United States Patent [19]

Hoover et al.

[11] **Patent Number: 5,749,087**

[45] **Date of Patent: May 5, 1998**

[54] **METHOD AND APPARATUS FOR MAINTAINING N-WAY ASSOCIATIVE DIRECTORIES UTILIZING A CONTENT ADDRESSABLE MEMORY**

[75] Inventors: **Russell D. Hoover**, Rochester; **George W. Nation**, Eyota; **Kenneth M. Valk**, Rochester, all of Minn.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **688,313**

[22] Filed: **Jul. 30, 1996**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **711/108**; 711/128; 711/133; 711/141; 711/146; 364/DIG. 1

[58] **Field of Search** 395/435, 449, 395/455, 457, 460, 468, 473

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,587,610	5/1986	Rodman	395/417
4,914,577	4/1990	Stewart et al.	395/417
4,972,338	11/1990	Crawford et al.	395/416
5,249,282	9/1993	Segers	395/440
5,261,106	11/1993	Lentz et al.	395/726
5,329,405	7/1994	Hou et al.	395/800
5,383,146	1/1995	Threewitt	365/49
5,404,482	4/1995	Stamm et al.	395/435
5,404,483	4/1995	Stamm et al.	395/435
5,414,704	5/1995	Spinney	370/60
5,457,788	10/1995	Machida	395/435
5,504,874	4/1996	Galles et al.	395/435
5,530,958	6/1996	Agarwal et al.	395/435
5,537,623	7/1996	Chamberlain et al.	395/435

Primary Examiner—Frank J. Asta
Attorney, Agent, or Firm—Joan Pennington

[57] **ABSTRACT**

A method and apparatus are provided for maintaining a N-way associative directory utilizing a content addressable memory (CAM). A congruence class from the N-way associative directory including a directory entry identified for a data operation is read into the CAM for the data operation. The directory entry for the data operation in the CAM is locked while the data operation is pending. Other entries in the congruence class are available. When the data operation is completed, checking for a state change is performed. Responsive to an identified state change, the directory entry for the data operation in the CAM is updated or marked as changed. The congruence class including the updated directory entry is marked as dirty. In accordance with features of the invention, the changed congruence class directory entries in the CAM are accumulated and scheduled to be written back to the N-way associative directory. The congruence classes including the changed directory entries in the CAM are written back to the N-way associative directory when the N-way associative directory is idle. After the congruence classes including the changed directory entries in the CAM are written back to the N-way associative directory, these CAM entries are marked as not busy and not dirty and can be reused.

16 Claims, 5 Drawing Sheets

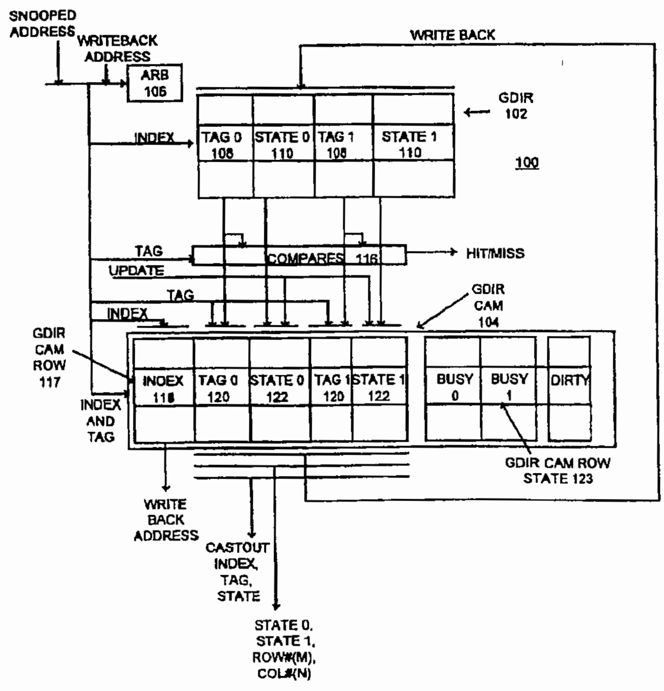
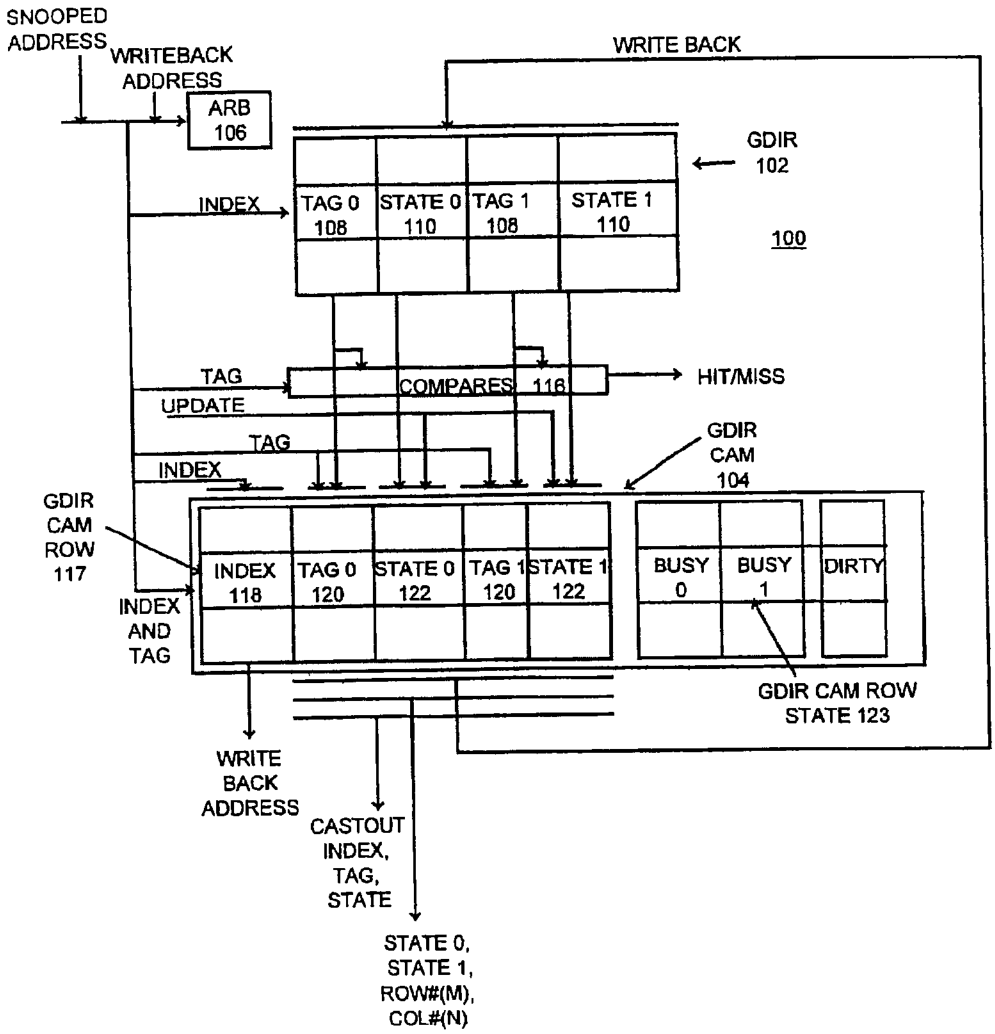


FIGURE 1A


SNOOPED ADDRESS
WRITEBACK ADDRESS
ARB 106
WRITE BACK
GDIR 102
INDEX
TAG 0 108
STATE 0 110
TAG 1 108
STATE 1 110
100
TAG
COMPARES 116
HIT/MISS
UPDATE
TAG
INDEX
GDIR CAM 104
GDIR CAM ROW 117
INDEX 118
TAG 0 120
STATE 0 122
TAG 1 120
STATE 1 122
BUSY 0
BUSY 1
DIRTY
INDEX AND TAG
GDIR CAM ROW STATE 123
WRITE BACK ADDRESS
CASTOUT INDEX, TAG, STATE
STATE 0, STATE 1, ROW#(M), COL#(N)

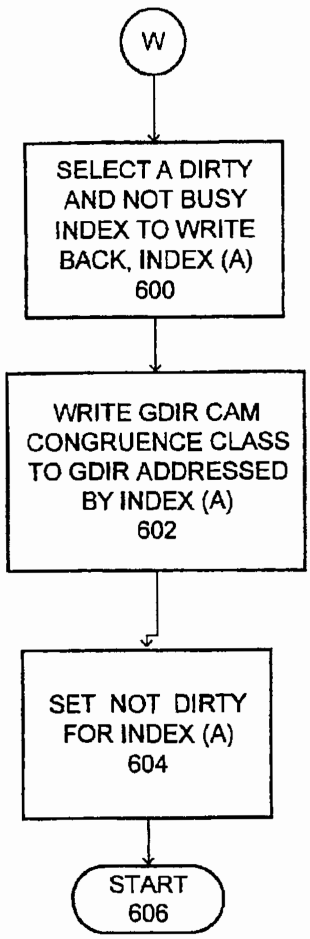


FIGURE 6

PRIOR ART

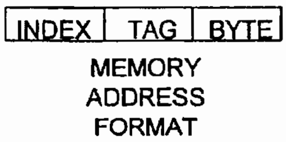


FIGURE 1B

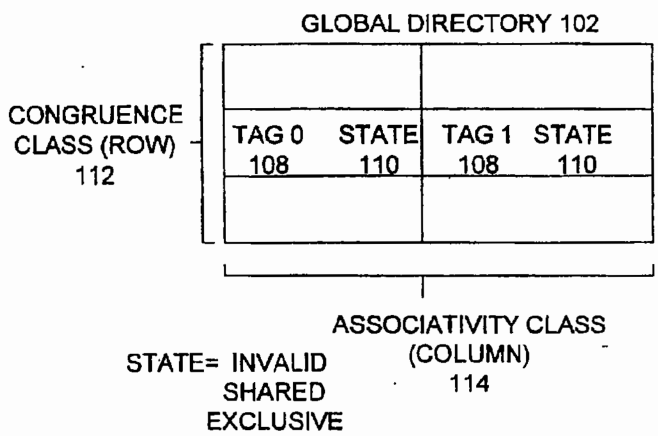


FIGURE 1C

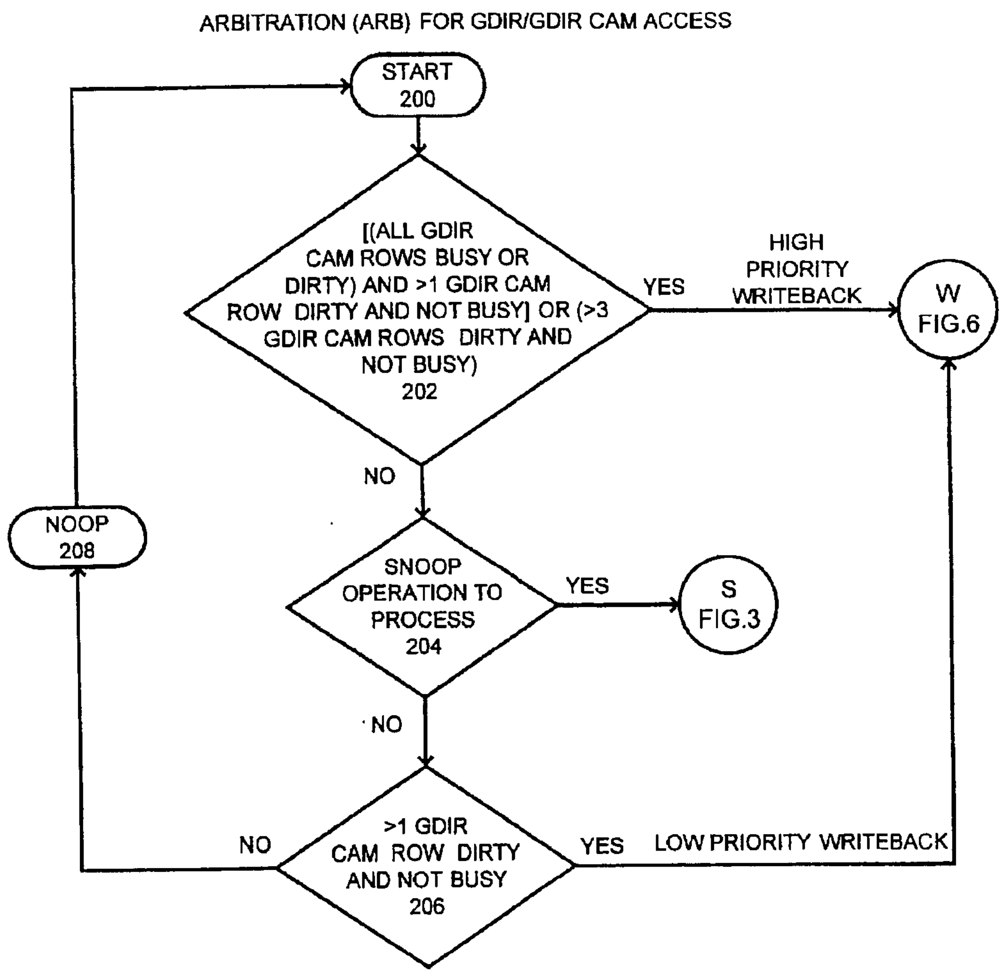

ARBITRATION (ARB) FOR GDIR/GDIR CAM ACCESS
START 200
[(ALL GDIR CAM ROWS BUSY OR DIRTY) AND >1 GDIR CAM ROW DIRTY AND NOT BUSY] OR (>3 GDIR CAM ROWS DIRTY AND NOT BUSY) 202
YES
HIGH PRIORITY WRITEBACK
W FIG.6
NO
NOOP 208
SNOOP OPERATION TO PROCESS 204
YES
S FIG.3
NO
>1 GDIR CAM ROW DIRTY AND NOT BUSY 206
NO
YES
LOW PRIORITY WRITEBACK

FIGURE 2

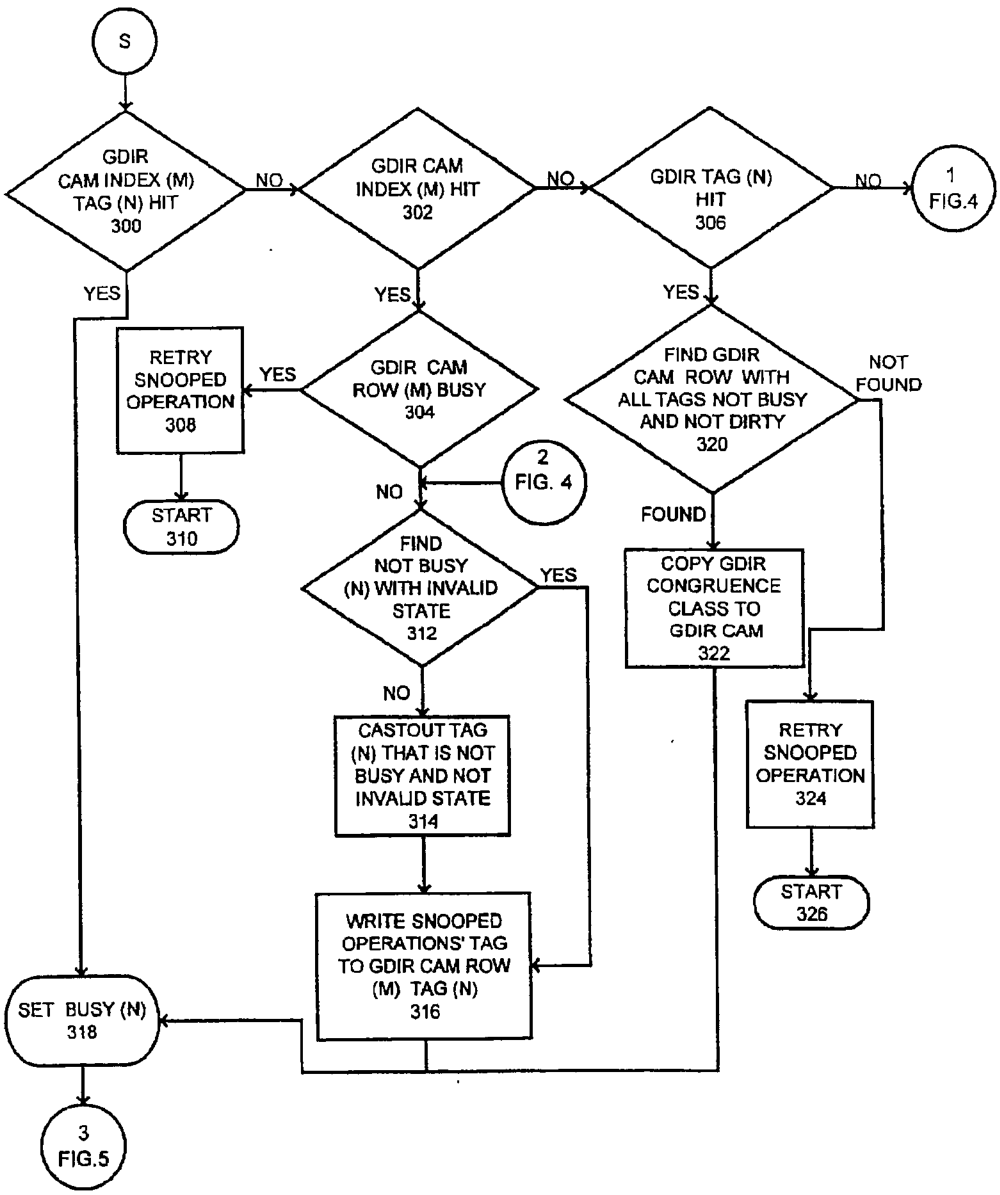

S
GDIR CAM INDEX (M) TAG (N) HIT 300
NO
GDIR CAM INDEX (M) HIT 302
NO
GDIR TAG (N) HIT 306
NO
1 FIG.4
YES
YES
YES
RETRY SNOOPED OPERATION 308
YES
GDIR CAM ROW (M) BUSY 304
FIND GDIR CAM ROW WITH ALL TAGS NOT BUSY AND NOT DIRTY 320
NOT FOUND
START 310
NO
2 FIG. 4
FOUND
FIND NOT BUSY (N) WITH INVALID STATE 312
YES
COPY GDIR CONGRUENCE CLASS TO GDIR CAM 322
NO
CASTOUT TAG (N) THAT IS NOT BUSY AND NOT INVALID STATE 314
RETRY SNOOPED OPERATION 324
START 326
WRITE SNOOPED OPERATIONS' TAG TO GDIR CAM ROW (M) TAG (N) 316
SET BUSY (N) 318
3 FIG.5

FIGURE 3

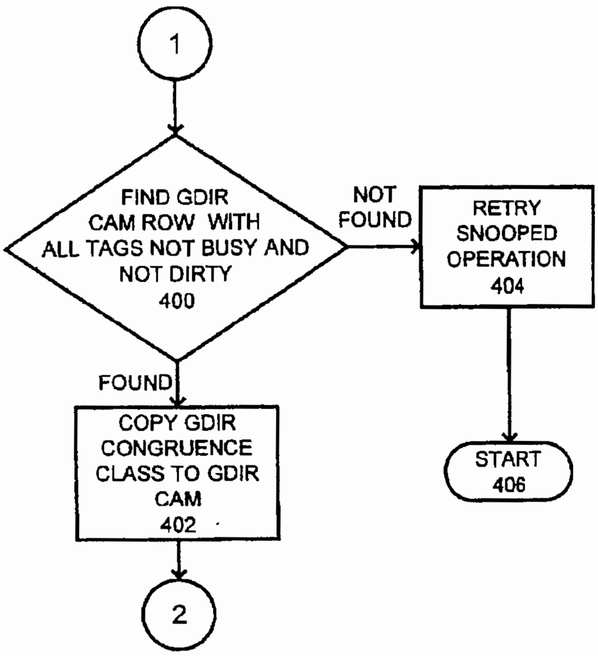


FIGURE 4

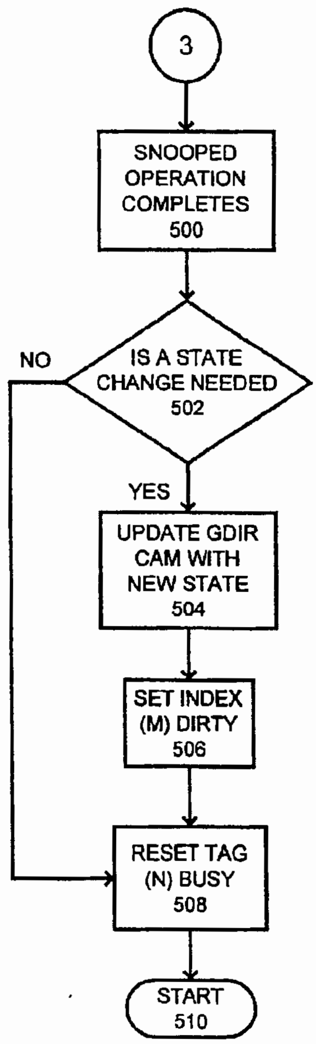


FIGURE 5

METHOD AND APPARATUS FOR MAINTAINING N-WAY ASSOCIATIVE DIRECTORIES UTILIZING A CONTENT ADDRESSABLE MEMORY

FIELD OF THE INVENTION

The present invention relates to a N-way associative directory, and more particularly to an improved method and apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM).

DESCRIPTION OF THE PRIOR ART

A content addressable memory (CAM) is known for many diverse uses. For example, known system have used a content addressable memory (CAM) for address translation, for example, as described in U.S. Pat. Nos. 4,972,282 and 5,457,788.

U.S. Pat. No. 5,249,282 discloses a cache memory for interfacing between a central processing unit and a main system memory. The cache memory includes a primary cache comprised of SRAMS and a secondary cache comprised of DRAM. A respective tag directory is associated with each of a plurality of secondary data cache memories. A respective content addressable memory (CAM) is associated with each of a plurality of primary data cache memories. Each of the CAMs stores data consisting of a tag and a value.

In cases where an N-way associative directory is used and operations on multiple lines (including when those lines belong to the same set) need to be performed in parallel, then when updating the directory a read modify write must be performed. For synchronous SRAMs, the performance degradation for changing from a write to a read, or from a read to a write can be significant. A need exists for a directory arrangement that provides improved efficient performance.

SUMMARY OF THE INVENTION

Important objects of the present invention are to provide an improved method and apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM), to provide such apparatus and method substantially without negative effects and that overcome many disadvantages of prior art arrangements.

In brief, a method and apparatus are provided for maintaining a N-way associative directory utilizing a content addressable memory (CAM). A congruence class from the N-way associative directory including a directory entry identified for a data operation is read into the CAM for the data operation. The directory entry for the data operation in the CAM is locked while the data operation is pending. Other entries in the congruence class are available. When the data operation is completed, checking for a state change is performed. Responsive to an identified state change, the directory entry for the data operation in the CAM is updated or marked as changed or dirty.

In accordance with features of the invention, the changed directory entries in the CAM are accumulated and scheduled to be written back to the N-way associative directory. The changed directory entries in the CAM can be used again before being written back to the N-way associative directory. A congruence class including the changed directory entry in the CAM is written back to the N-way associative directory when the N-way associative directory is idle. After the directory entries in the CAM are written back to the N-way associative directory, these CAM entries are marked not busy and not dirty and can be reused.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention together with the above and other objects and advantages may best be understood from the following detailed description of the preferred embodiments of the invention illustrated in the drawings, wherein:

FIG. 1A is a functional data flow block diagram of a directory system including a global or N-way associative directory with a content addressable memory (CAM) in accordance with the present invention;

FIG. 1B is a block diagram illustrating a conventional memory address format;

FIG. 1C is a block diagram illustrating a global directory of the present invention; and

FIGS. 2–6 are flow charts illustrating directory maintenance methods in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides an improved directory arrangement and method for maintaining a global or N-way associative directory utilizing a content addressable memory (CAM) that can be used in supporting many processor caches, each with many outstanding operations; large numbers of line fill buffers in a processor (not shown); and in caches with many outstanding transactions, such as, shared caches and lock-up free caches.

Having reference now to FIGS. 1A and 1C, in FIG. 1A there is shown a directory arrangement in accordance with the invention generally designated **100** including an N-way associative or global, coherence directory generally designated GDIR **102** with a content addressable memory (CAM) generally designated GDIR CAM **104**. GDIR CAM **104** is used in accordance with the invention to improve the performance of the N-way associative directory GDIR **102**. In accordance with features of the invention, a full congruence class or row **112**, the entry from each associativity class or column **114**, as illustrated in FIG. 1C including the entries TAG **0** **108**, STATE **0** **110**, TAG 1, STATE 1 **110**, is the unit of data moved between the coherence directory GDIR **102** and the GDIR CAM **104**. In FIGS. 1A and 1C, a two-way associtive directory GDIR **102** and GDIR CAM **104** are shown; however, it should be understood that the present invention can be used generally with an N-way associative directory. In FIG. 1B, a prior art memory address format including an index, tag, and byte is shown. In the preferred embodiment, the lower order address bits or byte of the prior art memory address format is not used.

In the GDIR CAM **104**, each GDIR CAM row **117** includes a single index **118**, multiple keys or tags **120** and associated states **122** together with BDIR CAM row state information **123** including respective BUSY **0**, BUSY **1**, and DIRTY bits. Each key **120** and associated state **122**, such as TAG **0**, STATE **0**, and TAG 1, STATE 1, corresponds to a respective associtiviy class **114**, CLASS **0**, CLASS 1 of the N-way associtive directory GDIR **102**. Moving the full congruence class **112** avoids having to do read modify write when data is moved between GDIR CAM **104** and coherence directory GDIR **102**. The GDIR CAM **104** contains GDIR entries that are in transition from one state to another state. The associated state **110**, **122** with a respective directory tag **108**, **120** include exclusive, shared, and invalid. An exclusive state indicates that one and only one cache in the system of the GDIR **102** has this block of data, where a shared state indicates that the block of data is shared. An invalid state indicates that the block of data is not cached.

GDIR CAM 104 serves as a CAM for directory entries. When an entry in the GDIR CAM 104 is updated and the operation using that entry is completed, that GDIR CAM row 117 is marked as dirty. Dirty GDIR CAM 104 entries are accumulated and scheduled for writing back to the global coherence directory GDIR 102. The accumulation of write-backs is more efficient because there is a number of cycles penalty for switching from read to write and vise-versa. The scheduling of these accumulated writebacks are more efficient because the writes are done when the global coherence directory GDIR 102 is idle. After the write-backs to the global coherence directory GDIR 102 are completed the entries of the GDIR CAM 104 are marked as not dirty and can be reused.

GDIR CAM 104 is a small CAM that duplicates some number of the directory rows 112 of GDIR 102. Global coherence directory GDIR 102 can be implemented with external SRAM off-chip because a large on-chip array may not be feasible to implement the total size needed for the global coherence directory GDIR 102. An arbitration (ARB) functional block 106 arbitrates access to GDIR 102 and GDIR CAM 104. ARB functional block 106 is implemented with logic arranged for directory access control of the invention as illustrated and described with respect to FIGS. 2–6. When an address is presented to the GDIR CAM 104, the address associated with the tag that matches this address is accessed. A Hit/Miss indication is provided by compares 116 and possibly, the location within the GDIR CAM 104 that address matched.

When a data line is accessed, the directory set or congruence class 112 of GDIR 102 that contains the line is read into the GDIR CAM 104. While an operation is pending the GDIR CAM row 117 including the particular congruence class entry 120, 122, TAG 0, STATE 0, or TAG 1, STATE 1 that contains the line is locked in place and released when the operation is finished. For an N-way associative directory GDIR 102, each of the N entries in a directory row may be locked by a different operation. When an operation modifies an entry in a GDIR CAM row 117 held in the GDIR CAM 104, that GDIR CAM row 117 is marked dirty to be written back to the directory when all entries are non-busy. The number of GDIR CAM rows 117 that the GDIR CAM 104 can hold advantageously can be provided to be greater than a maximum number of outstanding possible operations. The writing back dirty GDIR CAM rows 117 in the GDIR CAM 104 can be delayed until a number of GDIR CAM rows 117 are ready to be written back. Thus providing improved performance, for example, in synchronous SRAMs, grouping writes into adjacent cycles reduces the bandwidth taken up by writes to the SRAM. Also, a dirty GDIR CAM row 117 can be used by another data operation before being written back to the global coherence directory GDIR 102.

FIGS. 2–6 are flow charts illustrating directory maintenance methods in accordance with the present invention. Referring now to FIG. 2, arbitration (ARB) for access to GDIR 102 and GDIR CAM 104 start at a block 200. Checking whether all GDIR CAM rows 117 or all indexes in the GDIR CAM 104 are busy or dirty and more than one GDIR CAM row 117 is dirty and not busy; or more than a selected number of, for example, three GDIR CAM rows 117 in the GDIR CAM 104 are dirty and not busy is performed as indicated at a decision block 202. When determined at decision block 202 that all GDIR CAM rows 117 or all indexes in the GDIR CAM 104 are busy or dirty and more than one GDIR CAM row 117 or index is dirty and not busy; or more than the selected number of GDIR CAM rows 117 or indexes are dirty and not busy, then a high priority writeback is performed with the sequential operations continuing following entry point W in FIG. 6.

Otherwise when determined that it is not true at decision block 202 that all indexes in the GDIR CAM 104 are busy or dirty and more than one index is dirty and not busy; or more than the selected number of indexes are dirty and not busy, then checking for a snoop data operation to process is performed as indicated at a decision block 204. When a snoop data operation to process is identified at decision block 204, then the sequential operations continue following entry point S in FIG. 3. Otherwise when a snoop data operation to process is not identified at decision block 204 so that the global coherence directory GDIR 102 is idle, then checking whether the GDIR CAM 104 has more than one GDIR CAM row or index that are dirty and not busy is performed as indicated at a decision block 206. When determined at block 206 that the GDIR CAM 104 has more than one GDIR CAM row or index dirty and not busy, then a low priority writeback is performed with the sequential operations continuing following entry point W in FIG. 6. When determined at block 206 that the GDIR CAM 104 does not have more than one GDIR CAM row or index dirty and not busy, then the sequential steps return to start block 200 with no operation as indicated at a block 208.

Referring to FIG. 3, when a snoop data operation to process is identified at decision block 204, then the sequential operations continue following entry point S. Checking for a GDIR CAM row or index (M) and tag (N) hit is provided as indicated at a decision block 300. When a GDIR CAM row (M) and tag (N) hit is not identified at block 300, then checking for a GDIR CAM row or index (M) hit is performed as indicated at a decision block 302. When a GDIR CAM row or index (M) hit is identified at block 302, then checking whether all tags are busy at GDIR CAM row (M) in the GDIR CAM is performed as indicated at a decision block 304. When a GDIR CAM row (M) hit is not identified at block 302, then checking for a global directory tag (N) hit is provided as indicated at a decision block 306. When a global directory tag (N) hit is not identified at decision block 306, then the sequential steps continue following entry point 1 in FIG. 4.

Referring to FIG. 4, following entry point 1 checking for a GDIR CAM row with all tags not busy and not dirty is provided as indicated at a decision block 400. When a GDIR CAM row with all tags not busy and not dirty is found at decision block 400, then the congruence class is copied to the identified GDIR CAM row as indicated at a block 402. Then the sequential operations return following entry point 2 in FIG. 3. Otherwise when a GDIR CAM row with all tags not busy and not dirty is not found at decision block 400, then the snooped data operation is retried as indicated at a block 404. Then the sequential steps return to start block 200 in FIG. 2 as indicated at a block 406.

Referring again to FIG. 3, when determined at block 304 that all tags are busy at index (M) in the GDIR CAM, then the snooped data operation retried as indicated at a block 308. Then the sequential steps return to start block 200 in FIG. 2 as indicated at a block 310. When determined at block 304 that all tags are not busy at index (M) in the GDIR CAM and following an entry point 2 in FIG. 4, then checking for a not busy tag (N) with an invalid state is performed as indicated at a decision block 312. When a not busy (N) with tag (N) having an invalid state is not found at decision block 312, then tag (N) that is not busy and not invalid state is castout as indicated at a block 314. Then the snooped data operations' tag is written to the GDIR CAM (M) and tag (N) as indicated at a block 316. After the snooped data operations' tag is written at block 316 and when a GDIR CAM index (M) and tag (N) hit is identified at block 300, the busy (N) is set as indicated at a block 318. Then the sequential operations continue following entry point 3 in FIG. 5.

When a global directory tag (N) hit is identified at decision block 306, then checking for a GDIR CAM row not busy and not dirty is provided as indicated at a decision block 320. When a GDIR CAM row not busy and not dirty is found at decision block 320, then the congruence class is copied to the identified GDIR CAM row as indicated at a block 322. Then the steps continue at block 318 where the tag busy (N) is set. When a GDIR CAM row with all tags not busy and not dirty is not found at decision block 320, then the snooped data operation is retried as indicated at a block 324. Then the sequential steps return to start block 200 in FIG. 2 as indicated at a block 326.

Referring now to FIG. 5, following entry point 3, the snooped data operation completes as indicated at a block 500. Then it is determined whether a state change is needed as indicated at a decision block 502. When determined that a state change is needed at block 502, then the GDIR CAM is updated with the new state as indicated at a block 504. Next the index (M) is set dirty as indicated at a block 506. When determined that a state change is not needed at block 502 and after the index is set dirty at block 506, then the tag (N) busy is reset as indicated at a block 508. Then the sequential steps return to start block 200 in FIG. 2 as indicated at a block 510.

FIG. 6 illustrates writeback control flow for writing dirty entries of GDIR CAM 104 back to GDIR 102. The writeback steps begin following entry point W in FIG. 6 with selecting a dirty and not busy index to write back, index (A) as indicated at a block 600. The congruence class addressed by index (A) is written to the GDIR 102 as indicated at a block 602. Then the GDIR CAM 104 is set to not dirty for Index (A) as indicated at a block 606. Then the sequential steps return to start block 200 in FIG. 2 as indicated at a block 606.

While the present invention has been described with reference to the details of the embodiments of the invention shown in the drawing, these details are not intended to limit the scope of the invention as claimed in the appended claims.

What is claimed is:

1. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) comprising the steps of:

identifying a data operation to process;

identifying a congruence class from the N-way associative directory including a directory entry for said data operation; said congruence class directory entry including multiple (N) directory entries for each associativity class;

reading said congruence class from the N-way associative directory and writing said read congruence class into the CAM;

locking said directory entry for said data operation in CAM while said data operation is pending;

checking for a state change when said data operation is completed; and

updating said directory entry for said data operation in CAM responsive to said identified state change.

2. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 1 further includes the steps of:

accumulating a predefined number of said congruence classes including said updated directory entry in CAM; and

writing one of said congruence classes including said updated directory entry in CAM back to the N-way associative directory responsive to said accumulated predefined number of said congruence classes including said updated directory entry.

3. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 2 further includes the step of responsive to writing said congruence class including said updated directory entry in CAM back to the N-way associative directory, marking said congruence class directory entries in CAM as not busy and not dirty, whereby said CAM entry can be reused.

4. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 2 wherein said step of writing said updated congruence class directory entry in CAM back to the N-way associative directory includes the steps of:

selecting an index in CAM to write back; said selected index being an index set dirty and not busy;

writing said congruence class in CAM back to the N-way associative directory addressed by said selected index; and

resetting said dirty indication for said selected index in CAM.

5. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 1 wherein said step of locking said directory entry for said data operation in CAM while said data operation is pending includes the step of setting a busy indication for a tag associated with said data operation and resetting said busy indication for said tag associated with said data operation when said data operation is completed.

6. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 2 further includes the step of:

identifying an idle state for the N-way associative directory;

identifying a second predefined number of said congruence classes including said updated directory entry in CAM; and

writing a selected one of said congruence classes including said updated directory entry in CAM back to the N-way associative directory responsive to said identified second predefined number of said congruence classes including said updated directory entry in CAM.

7. A method for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 6 wherein said step of identifying said idle state for the N-way associative directory includes the step of identifying no data operations to process.

8. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) comprising:

means for identifying a data operation to process;

means for identifying a congruence class from the N-way associative directory including a directory entry for said data operation; said congruence class directory entry including multiple (N) directory entries for each associativity class;

means for reading said congruence class from the N-way associative directory and for writing said read congruence class into the CAM;

means for locking said directory entry for said data operation in CAM while said data operation is pending;

means for identifying a state change when said data operation is completed; and

means for updating said directory entry for said data operation in CAM responsive to said state change identifying means.

9. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 8 wherein said congruence class in CAM includes a single index.

10. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 9 wherein each said multiple (N) directory entries for each associativity class includes a tag and an associated state.

11. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 10 wherein said means for updating said directory entry for said data operation in CAM responsive to said state change identifying means includes means for updating an associated state with a tag of one of said multiple (N) directory entries for said identified data operation.

12. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 11 further includes means responsive to said state change identifying means for setting a changed indication for said index for said congruence class in CAM.

13. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 11 further includes means for accumulating a predefined number of said congruence classes including said updated directory entry in CAM; and means for writing back at least one of said congruence classes including said updated directory entry in CAM to the N-way associative directory responsive to said accumulated predefined number of said congruence classes including said updated directory entry in CAM.

14. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 13 further includes means responsive to said congruence class writing back means for marking said multiple directory entries (N) in said at least one congruence class in CAM as not busy and said at least one congruence class as not dirty, whereby said CAM index can be reused.

15. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 12 wherein said means for writing back at least one of said congruence classes including said updated directory entry in CAM to the N-way associative directory include means for selecting an index in CAM to write back; said selected index being an index set changed and said multiple directory entries (N) in said congruence class in CAM set as not busy; means for writing said congruence class directory entry in CAM back to the N-way associative directory addressed by said selected index; and means for resetting said changed indication for said selected index in CAM.

16. Apparatus for maintaining a N-way associative directory utilizing a content addressable memory (CAM) as recited in claim 15 further include means for identifying an idle state of the N-way associative directory; means for identifying a second predefined number of said congruence classes including said updated directory entry in CAM; said second predefined number being less than said first predefined number; and means for writing a selected one of said congruence classes including said updated directory entry in CAM back to the N-way associative directory responsive to said identified second predefined number of said congruence classes including said updated directory entry in CAM.

* * * * *

United States Patent [19]

Churchill, Jr.

[11] **3,949,369**

[45] **Apr. 6, 1976**

[54] **MEMORY ACCESS TECHNIQUE**

[75] Inventor: **William Philip Churchill, Jr.,** Carlisle, Mass.

[73] Assignee: **Data General Corporation,** Southboro, Mass.

[22] Filed: **Jan. 23, 1974**

[21] Appl. No.: **436,023**

[52] **U.S. Cl.** **340/172.5**
[51] **Int. Cl.²** **G06F 13/00**
[58] **Field of Search** 340/172.5

[56] **References Cited**

UNITED STATES PATENTS

3,275,991	9/1966	Schneberger	340/172.5
3,292,153	12/1966	Barton	340/172.5
3,333,252	7/1967	Shimabukuro	340/172.5
3,339,183	8/1967	Bock	340/172.5
3,344,405	9/1967	Craft	340/172.5
3,693,165	9/1972	Reiley	340/172.5

Primary Examiner—Gareth D. Shaw
Assistant Examiner—James D. Thomas
Attorney, Agent, or Firm—Jacob Frank

[57] **ABSTRACT**

In a digital computer system having a main memory operable at a first speed, a high speed buffer operating at a second speed for temporarily storing selected portions of the main memory, an associative memory for temporarily storing selected main memory addresses and comparing the stored addresses with a newly received address in a read/write operation to generate comparison data, a read only memory a bit configuration reflecting an algorithm, connected to the associative memory for generating a new order of priority for the memory address stored in the associative memory, and a storage unit connected from the read only memory for storing that order of priority for subsequent feedback to the read only memory in a subsequent cycle as a previous order of priority.

7 Claims, 6 Drawing Figures

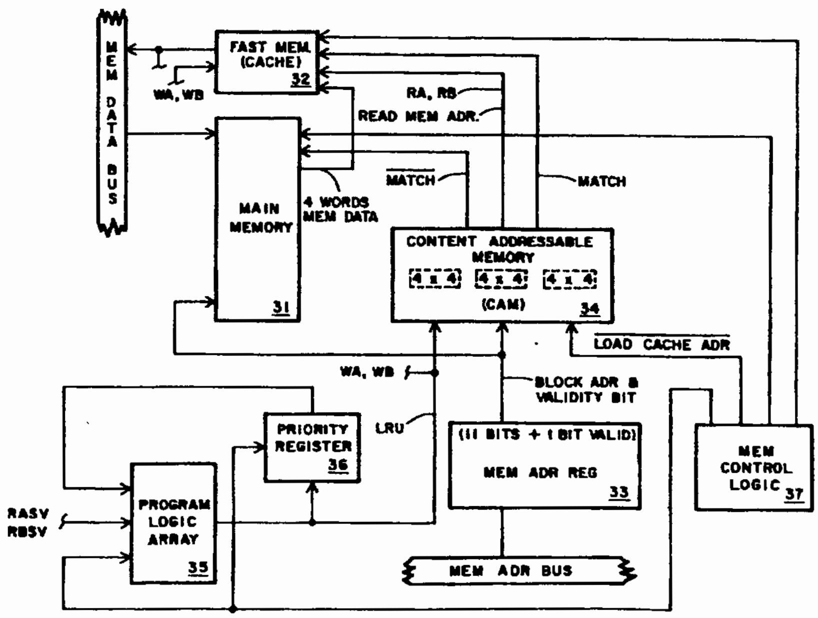
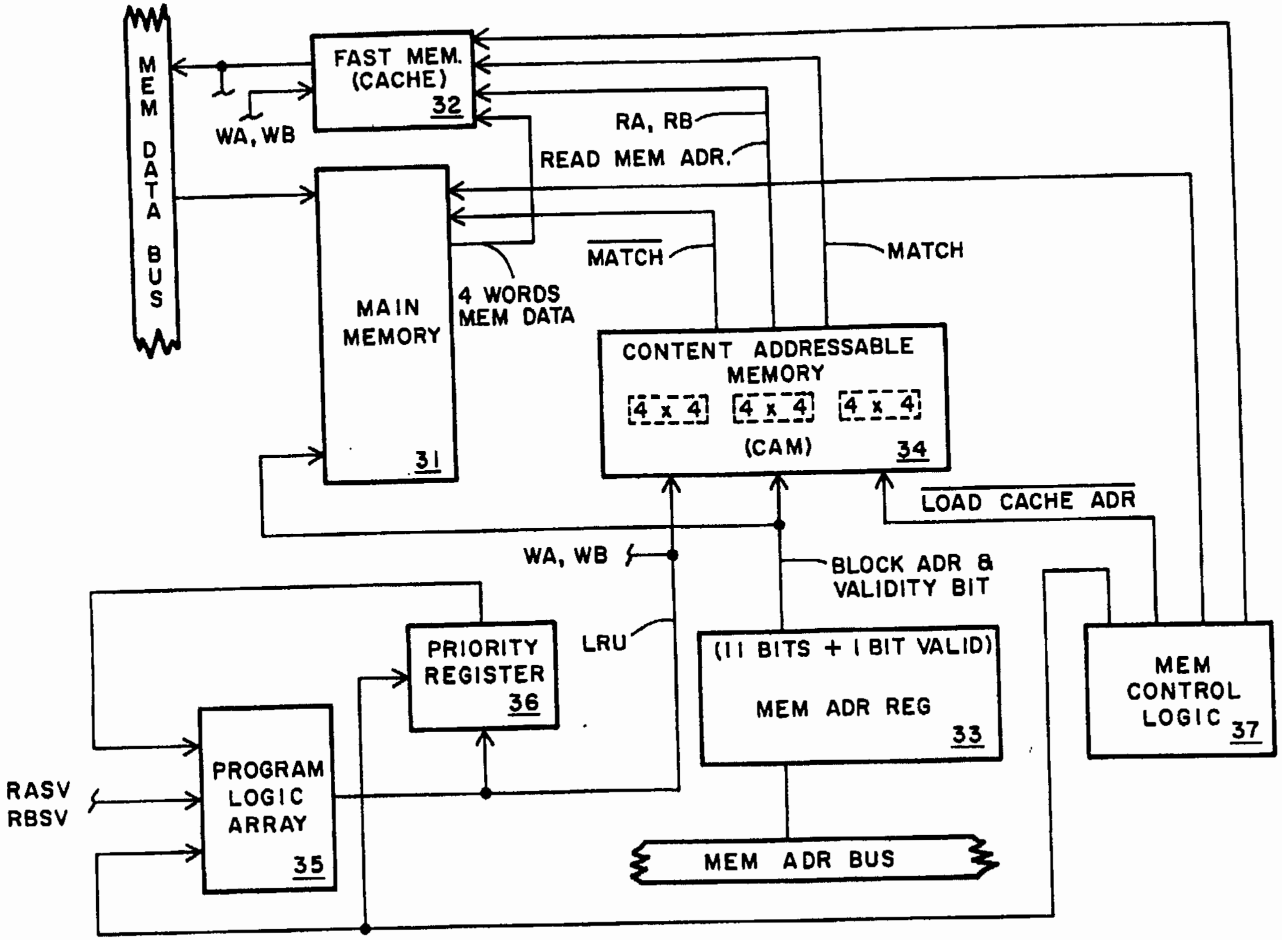


FIG. 1


MEM DATA BUS
FAST MEM. (CACHE) 32
WA, WB
RA, RB
READ MEM ADR.
MAIN MEMORY 31
4 WORDS MEM DATA
MATCH
MATCH
CONTENT ADDRESSABLE MEMORY
4 x 4
4 x 4
4 x 4
(CAM) 34
LOAD CACHE ADR
WA, WB
BLOCK ADR & VALIDITY BIT
LRU
PRIORITY REGISTER 36
(11 BITS + 1 BIT VALID)
MEM ADR REG 33
MEM CONTROL LOGIC 37
RASV
RBSV
PROGRAM LOGIC ARRAY 35
MEM ADR BUS

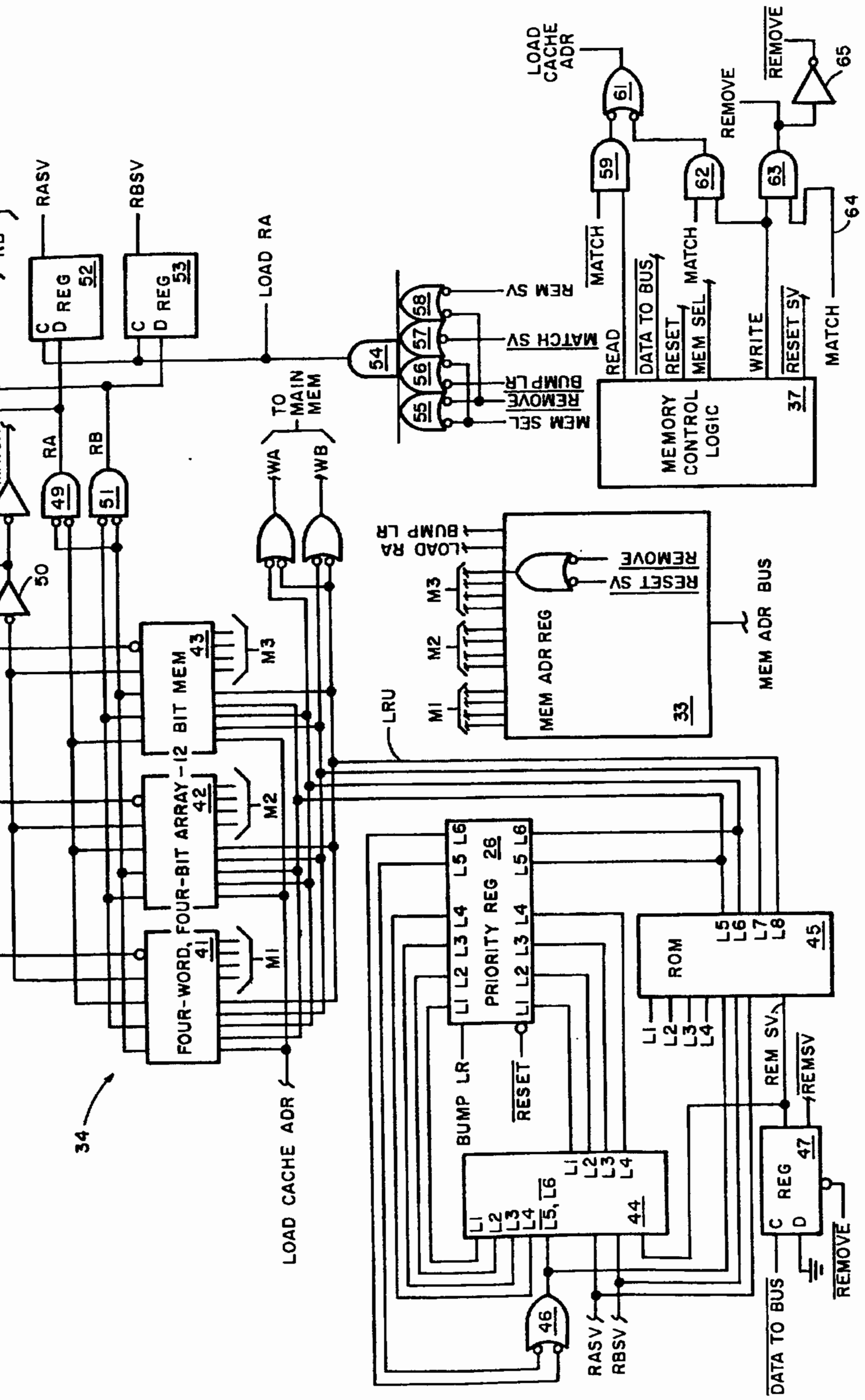

TO CACHE 32
RA
RB
RASV
RBSV
REG 52
REG 53
LOAD RA
MATCH
48
50
49
51
TO MAIN MEM
WA
WB
FOUR-WORD, FOUR-BIT ARRAY - 12 BIT MEM
41
42
43
M1
M2
M3
34
LOAD CACHE ADR
LRU
MEM ADR REG
33
MEM ADR BUS
BUMP LR
REMOVE
RESET SV
54
55
56
57
58
MEM SEL
MATCH SV
REM SV
MEMORY CONTROL LOGIC
37
READ
DATA TO BUS
RESET
MEM SEL
WRITE
RESET SV
59
61
62
63
64
65
LOAD CACHE ADR
PRIORITY REG 28
ROM
45
L1 L2 L3 L4 L5 L6
L5 L6 L7 L8
44
46
RESET
REG 47
REM SV
REMSV
DATA TO BUS

FIG. 2

FIG. 3

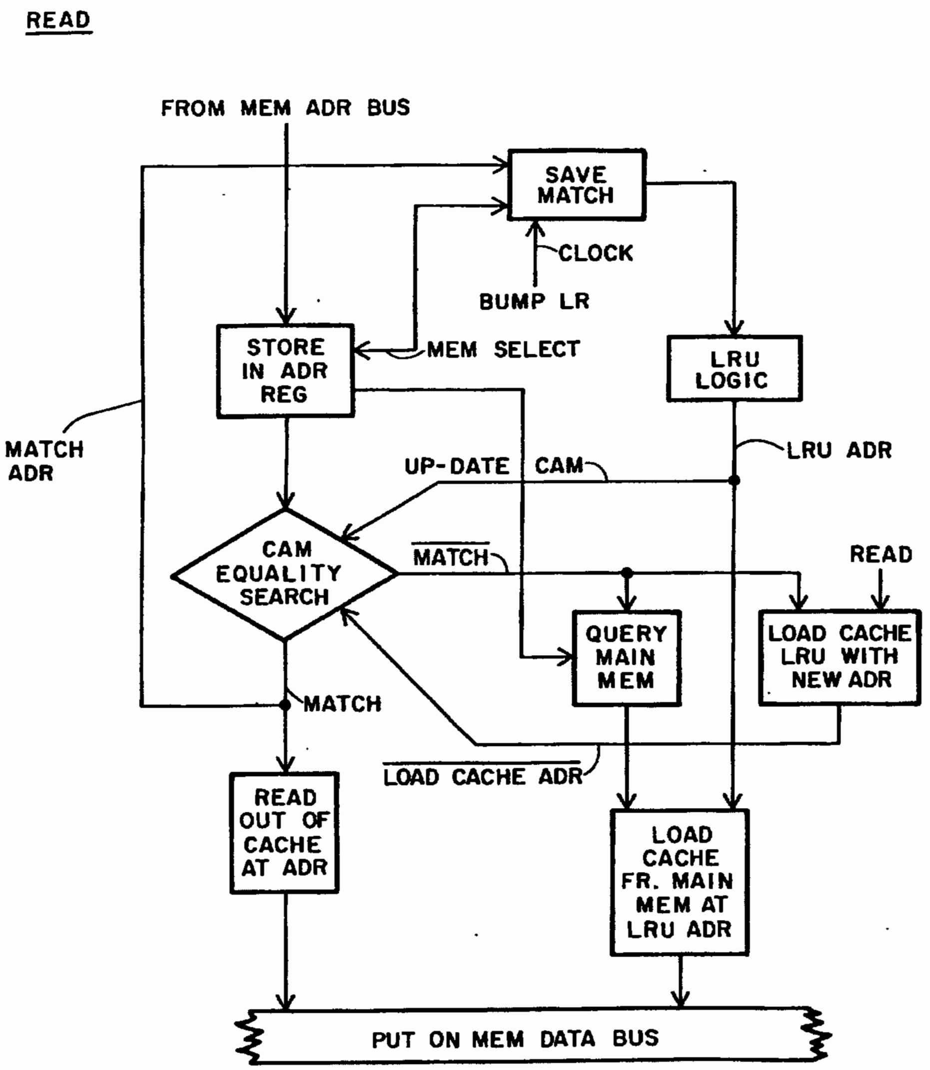

READ
FROM MEM ADR BUS
SAVE MATCH
CLOCK
BUMP LR
STORE IN ADR REG
MEM SELECT
LRU LOGIC
MATCH ADR
LRU ADR
UP-DATE CAM
CAM EQUALITY SEARCH
MATCH
READ
QUERY MAIN MEM
LOAD CACHE LRU WITH NEW ADR
MATCH
LOAD CACHE ADR
READ OUT OF CACHE AT ADR
LOAD CACHE FR. MAIN MEM AT LRU ADR
PUT ON MEM DATA BUS

FIG. 4

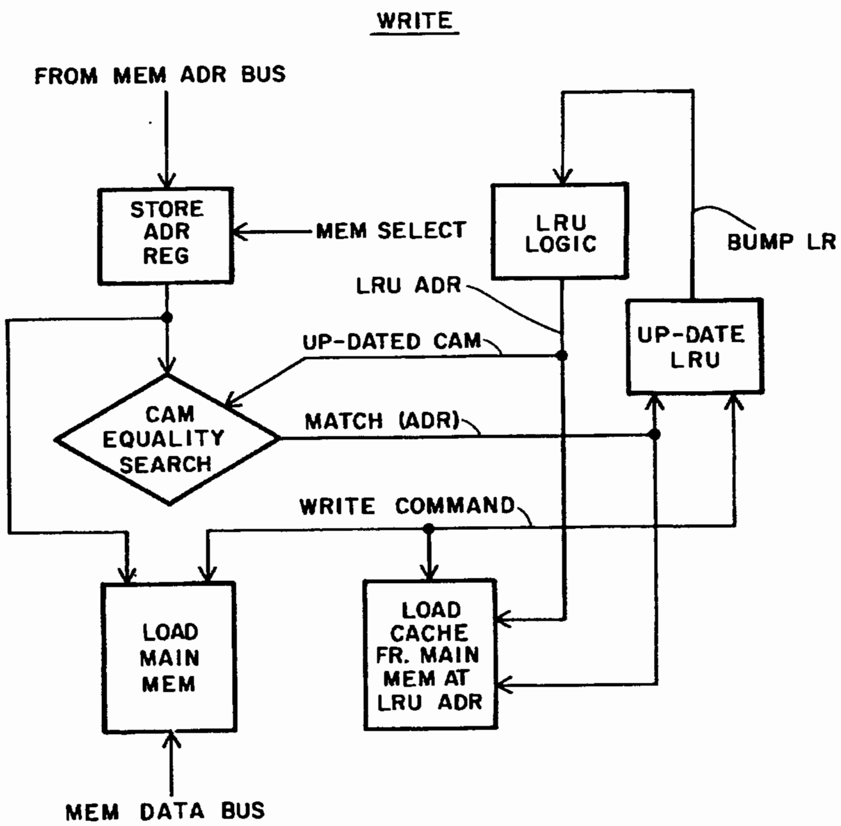

WRITE
FROM MEM ADR BUS
STORE ADR REG
MEM SELECT
LRU LOGIC
BUMP LR
LRU ADR
UP-DATED CAM
UP-DATE LRU
CAM EQUALITY SEARCH
MATCH (ADR)
WRITE COMMAND
LOAD MAIN MEM
LOAD CACHE FR. MAIN MEM AT LRU ADR
MEM DATA BUS

FIG. 5

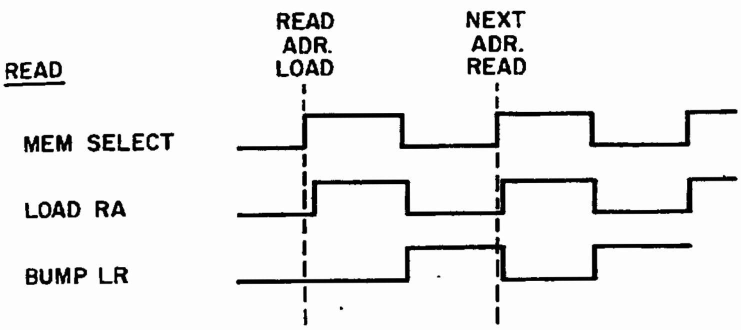
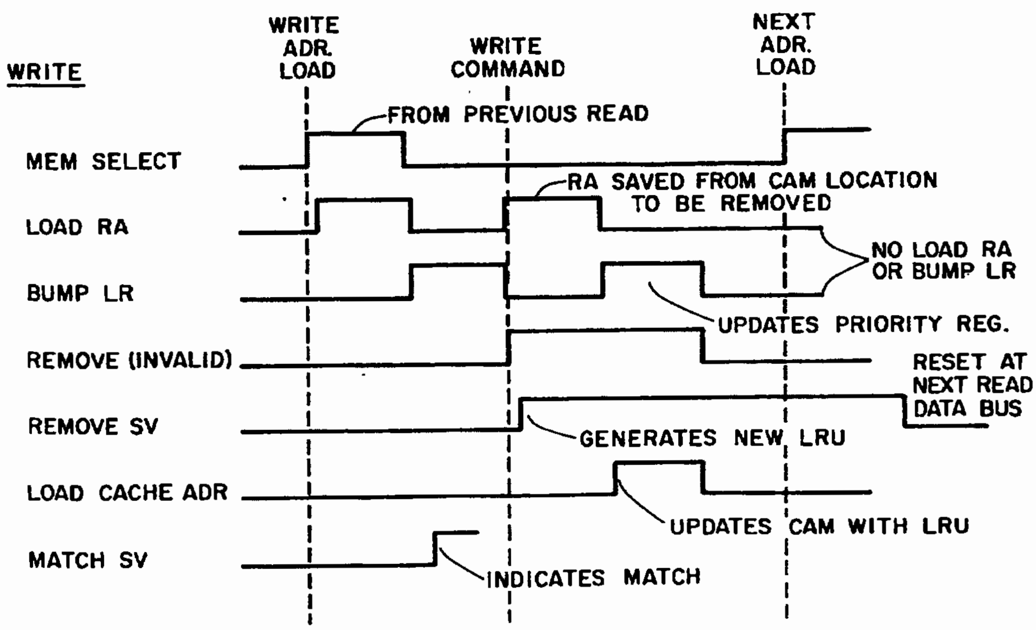

READ
READ ADR. LOAD
NEXT ADR. READ
MEM SELECT
LOAD RA
BUMP LR

FIG. 6


WRITE
WRITE ADR. LOAD
WRITE COMMAND
NEXT ADR. LOAD
FROM PREVIOUS READ
MEM SELECT
RA SAVED FROM CAM LOCATION TO BE REMOVED
LOAD RA
NO LOAD RA OR BUMP LR
BUMP LR
UPDATES PRIORITY REG.
REMOVE (INVALID)
RESET AT NEXT READ DATA BUS
REMOVE SV
GENERATES NEW LRU
LOAD CACHE ADR
UPDATES CAM WITH LRU
MATCH SV
INDICATES MATCH

MEMORY ACCESS TECHNIQUE

CROSS REFERENCE TO RELATED APPLICATIONS

Filed simultaneously with this application is a patent application assigned to the same assignee as this application and is identified as Ser. No. 436,022 filed Jan. 23, 1974 for Automatic Data Priority Technique and, entitled Automatic Data Priority Technique by Joseph Thomas West.

1. Field of the Invention

The field of art to which the present invention pertains is to memory systems in general and, in particular, to the improvement of memory systems utilizing high speed buffers for establishing a storage hierarchy.

2. Description of the Prior Art

Access to memories of high speeds is of utmost concern in order to provide for the rapid processing of data and to take advantage of the high speed CPU systems available today. One manner of achieving increased memory speed is providing for a memory hierarchy scheme where a large slow memory and a small fast memory are connected to a central processing unit (CPU). The fast memory, commonly known as a cache, serves as a window for the CPU to look at slow memory. Data from slow memory is loaded in the cache in quantities of usually several words (or bytes) at once in anticipation that subsequent memory request will be for that data. If so, then memory speed is increased by serving the CPU from the cache.

A memory system of this type requires management which has to determine: first, whether a CPU request for memory is in cache and if so, where; second, if not in cache, at what location in cache is the data from the slow memory to be loaded; third, how does the CPU modify fast and slow memory, and; fourth, how is the system to be initialized on power-up.

Inherent in the cache scheme is an associative memory which contains the address of data in the cache as related to the slow memory. This associative memory is effectively implemented as a content addressable memory (CAM) which provides for a simultaneous search of all its locations to determine if the data desired by the CPU is in the cache, and if so, where.

Among the several items governing the performance of a memory system of the type being discussed, is the ratio of speed between the slow memory and the cache. This also may be determined by the relative size of the cache and slow memory. Once a cache size and speed is selected that provides the desirable performance, the problem arises as to how to derive an efficient method of replacement of old words in cache with new ones.

If the system is to operate efficiently, replacement of data in the cache must be carefully accomplished. Although a complete knowledge of program behavior would produce the ideal replacement, this may be impractical because of the economics involved. A good approximation is to replace the least recently used entry. This will require maintaining a priority which is updated at each memory access. Efficiency can be further improved if invalidated addresses can be placed at the bottom of the schedule so they can be replaced first without destroying the valid entries.

SUMMARY OF THE INVENTION

Accordingly, an object of the present invention is to provide an improved high speed memory system by implementation of a least recently used technique having a bit configuration representing an algorithm, with an associative memory to keep track not only of the least recently used word, but in addition, to establish an order of word state priority for manipulating cache stored data, allowing a data priority locating scheme to be dynamically updated as new usage information becomes available.

Another object of this invention is to provide a programmed word state priority order based on usage that is normally not affected by effecting storage operations in main memory.

A further object of the present invention is to provide a programmed word state priority based on usage, which when containing an address location in an associative memory that is subsequently written into in main memory, invalidates the associative memory.

DESCRIPTION OF THE DRAWINGS

FIG. 1 is an overall block diagram of the preferred embodiment of the invention.

FIG. 2 is a circuit diagram of the CAM 34, program logic array 35 and priority register 36, and portions of memory control logic 37, shown in FIG. 1.

FIG. 3 is a flow diagram depicting the sequence of events in the present invention in a read cycle.

FIG. 4 is a flow diagram depicting the sequence of events in the present invention in a write cycle.

FIG. 5 is a series of time based waveforms illustrating, with certain signals, the manner of operation of the invention during a read cycle.

FIG. 6 is a series of time based waveforms illustrating, with certain signals, the manner of operation of the invention during a write cycle.

DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference to the drawings, a block diagram generally describing the present invention is illustrated at FIG. 1, wherein there is shown a main semi-conductor memory 31 having a cycle time, for example, of 600 ns and a smaller fast semi-conductor memory 32, generally referred to as a cache, having a cycle time, for example, of 100 ns to 200 ns. Main memory 31 is connected from the memory data bus and, in addition, from a memory address register 33, the latter in turn connected from the memory bus.

Connected from the memory address on register 33 is an associative memory in the form of a content addressable memory (CAM) 34, which is designed to compare data on its inputs with data already stored in its memory and indicates a match when these data are identical. This equality search is performed on all bits in parallel. The stored data is four 12-bit words and the signal input is one eleven-bit word from the memory address register 33 and a validity bit 33. The outputs of CAM 34 include a match signal to a cache memory 32 and the main memory 31 and, in addition, an address denoted as RA and RB, designating a fast memory location in cache 32.

The main memory 31 is also connected for loading the cache 32 with four words or one block of memory data when instructed to do so. The WA and WB signals which are supplied to the cache 32 will always denote the cache address where the data from main memory is to be written, which is to be described in greater detail. This signal might also be called the LRU, as it identifies the location of the least recently used data in the cache system and then the cache address to be loaded, should a situation call for loading of the cache. The LRU signal is also supplied to an input of the CAM to update the least recently used data location of the CAM with the main memory address of the new data that is loaded in the cache.

The LRU is derived from a program logic array 35 which might comprise of a selected combination of discrete gates or a read only memory (ROM). The program logic array defines an LRU algorithm for the four word associative memory or CAM 34. The LRU algorithm is such that not only will the least recently used word be known, but also the next to least recently used word and so forth. This allows the LRU algorithm to be dynamically updated in terms of a time and usage basis as newly used information becomes available. In the present embodiment, since four words of data are to be used with the CAM 34 and cache 32, these might be defined as the MRU (most recently used), NMRU (next most recently used), NLRU (next least recently used) and LRU (least recently used). It is evident, that for these four words there are 24 possible states of the algorithm defining 24 distinct combinations of four word arrangements, depending upon the order of priorities ascertained.

In order to dynamically update the algorithm, it is necessary to know the state or order of priority of the immediately previous combination of four words, as well as the address in CAM 34 of the new information loaded from the memory address register 33. The WA and WB signals on the LRU lead denote the address location in the CAM 34 of the newly entered main memory address and the corresponding location in cache 52 of that address data for the newly entered information. The RASV and RBSV signals are delayed versions of the RA and RB signals, as will be discussed hereinafter, to identify the locations in the CAM, if any, which the new information matches. The information as to the absence of a match or if a match was matched, all contribute to re-establish the new order of priority for determining the new LRU data. As may be seen, the priority register is utilized for temporarily storing the immediately previous order of priority information, enabling it to be re-circulated during the next cycle back into the program logic array 35.

The memory control logic 37 is connected to each, cache 32, main memory 31, CAM 34, program logic array 35, and priority register 36, to ensure that the proper sequence of information handling is maintained, as will become evident hereinafter.

A more detailed description of CAM 34, priority register 36, and program logic array 35, may be seen with reference to FIG. 2, wherein there is shown a four word, four-bit array and 12 bit CAM 34 comprising units 41, 42 and 43. Four input LRU leads to each of these units contain LRU information and four other leads to these respective units comprise three sets of four bit inputs mutually denoted as M_1, M_2 and M_3. The outputs of the memory address register 33 comprise 11 bits, representing the signal received from the memory address bus, identifying the location in main memory 31 at which data is to be read in or written out. The twelfth bit is a validity bit to denote a validity condition of the signal written in and therefore if written invalid, the other 11 bits will be ignored. Each of the three units 41, 42 and 43 are also fed with a LOAD CACHE ADR signal which, when enabled, allows the LRU identified address in CAM 34 to receive the newly entered main memory address from memory address register to update the units 41, 42 and 43.

The program logic array 35 is shown in the form of two read only memories, ROM 44 and ROM 45, each having common inputs including: L_1 through L_4 from the priority register; $\bar{L}_5 + \bar{L}_6$ RASV and RBSV, and; REMSV. The signal REMSV to be discussed hereinafter will indicate whether a CAM stored main memory address is to be invalidated or not. One possible program logic array table for the ROM's is shown on the following page, where given each of the 24 different word state orders of priority is a binary output on leads L_1 through L_8. An octal output is provided for the eight binary output values on the leads of the combined ROM's 44 and 45.

PROGRAM LOGIC ARRAY WORD STATE TABLE

Actual Word States	Output L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	Output Octal Code
1230	0	0	0	1	0	1	1	1	027
1320	0	0	1	0	0	1	1	1	047
2130	0	0	1	1	0	1	1	1	067
2310	0	1	0	0	0	1	1	1	107
3120	0	1	0	1	0	1	1	1	127
3210	0	1	1	0	0	1	1	1	147
0231	1	0	0	1	1	0	1	1	233
0321	1	0	1	0	1	0	1	1	253
2031	1	0	1	1	1	0	1	1	273
2301	1	1	0	0	1	0	1	1	313
3021	1	1	0	1	1	0	1	1	333
3201	1	1	1	0	1	0	1	1	353
0132	0	0	0	1	1	1	0	1	036
0312	0	0	1	0	1	1	0	1	055
1032	0	0	1	1	1	1	0	1	075
1302	0	1	0	0	1	1	0	1	115
3012	0	1	0	1	1	1	0	1	135
3102	0	1	1	0	1	1	0	1	155
0213	1	0	0	1	1	1	1	0	236
0213	1	0	1	0	1	1	1	0	256
1023	1	0	1	1	1	1	1	0	276
1203	1	1	0	0	1	1	1	0	316
2013	1	1	0	1	1	1	1	0	336
2103	1	1	1	0	1	1	1	0	356

In addition, there is a portion of one possible ROM truth table on the following page showing previous priority state possibilities and the variations of the inputs RASV, RBSV AND REMSV along with the octal output code for each output state depending on the variation of the input signals. The octal input on the following page is based upon the following input signals in a left to right order: LR1; LR2; LR3; LR4; LR5 + LR6; RB; RA; REM. For example, in word order 0132, the Octal Input for Octal Output 047 would read left to right 000 10 000.

ROM TRUTH TABLE

Word Order	Octal Input	Octal Output	Word Order	Octal Input	Octal Output
0132	020	047	1032	060	047
	021	035		061	035
	022	253		062	353
	023	075		063	075
	024	035		064	075
	025	336		065	356
	026	236		066	276
	027	135		067	155
1230	030	027	2130	070	067
	031	236		071	256
	032	313		072	313
	033	027		073	027
	034	115		074	115
	035	067		075	067
	036	316		076	356
	037	127		077	147
0312	040	127	1302	100	047
	041	055		101	035
	042	253		102	333
	043	075		103	115
	044	055		104	115
	045	273		105	067
	046	236		106	276
	047	135		107	155
1320	050	047	2310	110	107
	051	035		111	233
	052	353		112	313
	053	047		113	027
	054	115		114	155
	055	067		115	107
	056	316		116	356
	057	127		117	147

When REMSV is true, it indicates a write instruction had occurred and address was matched at the zero location in the CAM 34 so that the zero location had to be invalidated and made the LRU as new information is to be written into that main memory address.

With reference to the above table, it will be seen that given an order of priority of 0132 for locations in the CAM 34 and cache 32, a different order or priority output (octal code) will result for different RASV, RBSV and REMSV signals. If RASV and RBSV are both zeros and REMSV is true, the new order of priority is changed to 1320 represented by octal code 047.

If this were not done, it can be readily observed that confusion might occur during the reading of subsequent information. When REMSV is false, information is not to be invalidated. However, since the zero location is the one that is matched and active, the same order of priority 0132 is maintained as is represented by the octal output 035 which can be verified by looking at the illustrated program logic array word state table above.

The four outputs from ROM 44 and the two outputs L_5 and L_6 from ROM 45, are connected back into the priority register 26 L_1 to L_6 to the inputs of ROM's 44 and 45, for allowing this information to be used during a subsequent cycle to establish a new set order priority should the signals RASV, RBSV and REMSV require such.

As will be noted, the signals L_5 and L_6 in being returned to ROM's 44 and 45 are returned via a NOR gate 46. Furthermore, the REMSV signal from a register 47 is entered into ROM's 44 and 45 only upon the

presence of a change in the order of priority, as when the address of information to be written in main memory matches a CAM address that CAM address is to be invalidated and made the LRU. The DATA TO BUS signal is used to clear the REMSV on the next cache access.

It is also noted, signal BUMP LR triggers the priority register to enter into the ROM's the old priority order and then receive the new priority order for the next cycle in a manner to be hereinafter discussed.

The output of the CAM 34, including units 41, 42 and 43 provide, via an inverter coupling OR gate 50 and inverter 49, a MATCH AND $\overline{\text{MATCH}}$ indication respectively denoting whether or not the 11-bit address received from the memory address register is common to any one of the four word, 12 bit arrays stored in the CAM. Signals RA, RB denote the CAM location of the address of the data as to which a match has been detected. The signals RA and RB which are respectively derived from NAND gates 49 and 51 are mutually routed to registers 52 and 53, so that the signals RA and RB can be stored and supplied as RASV and RBSV during a successive cycle depending on whether the conditions entered into C input of the registers 52 and 53 are met.

As will be seen, memory control logic 37, upon the presence of a read and match false signal, will enable an AND gate 59 connected to one input of a NOR gate 51, the second input of NOR gate 61 supplied from an AND gate 62 having write and match inputs. These two inputs to NOR gate 61 generate a LOAD CACHE ADR signal. The output of AND gate 62 also provides a signal which may be denoted as BUMP LR. The write signal supplied to an input of an AND gate 63 is a write signal and MATCH signal to generate an output REMOVE. Other signals that are conventional put out by the memory control logic include a $\overline{\text{DATA TO BUS}}$ signal denoting that data has been put on the memory data bus. A reset signal for a resetting condition is also generated and a MEM SEL signal is generated denoting the loading of an address from the memory address bus into the memory address register 33. A $\overline{\text{RESET SV}}$ signal is also delayed for a subsequent cycle.

The BUMP LR signal from AND gate 62 occurs when the CAM has indicated a match in a WRITE condition. BUMP LR will also occur from the memory address register 33 in the form of a delayed load RA, whereby load RA denotes a previous READ operation with data loaded into the memory address register from the memory address bus. Thus, BUMP LR always enables the priority register to load the ROM's each time an operation has been effected in CAM 34 in a READ condition and a match occurs in a WRITE condition. No BUMP LR signal occurs when in a WRITE operation and match is false.

The various signals fed into the NOR gates 55 to 58 which are connected to AND gate 54, establish the condition $\overline{\text{LOAD RA}}$ which is generated immediately following the MEM SEL signal that occurs with a READ or WRITE signal at the loading of the memory address register.

OPERATION

The operation of the present invention will now be discussed in connection with the flow diagrams for READ and WRITE conditions respectively depicted in FIGS. 3 and 4 and the waveform diagrams for the READ and WRITE conditions respectively depicted in

FIGS. 5 and 6.

First, an assumption will be made that a READ condition exists where the computer is reading the address of a data word that is stored in the cache. The memory address of the data is read into the memory address register 33 from the memory address bus and then fed into the CAM 34 on level line M1, M2 and M3. The CAM has already been updated at the leading edge of LOAD RA with the previous LRU address information from ROM 45. In the CAM, an equality search is made between address and the four memory addresses stored in the CAM to ascertain whether or not a match exists.

Assuming a match is detected, this indicates that the memory address is already in the CAM and therefore the corresponding memory data is stored within the cache. Upon occurrence of a match, a match signal is generated at the output of Inverter 48 and signals RA and RB are also generated to identify at which one of four locations in the CAM a match occurred. The match location in terms of RA and RB is set into registers 52 and 53 to be saved for updating the priority register after this read cycle. At the same time, signals RA and RB identify the location of the data in the cache 32 which is to be read out onto the memory data bus. The BUMP LR signal, as may be seen from FIG. 5, which is LOAD RA delayed, enables the priority register 36 to store the order of priority generated during the present cycle. At the beginning of the next cycle, the RA and RB CAM location match saved from the previous cycle is generated and together with the signals L_1 through L_4 from priority register 36 are fed along with REMSV to ROM's 44 and 45. The information at the output LRU leads of ROM 45 is represented by WA and WB and is available for input to the CAM 34 allowing the CAM 34 location of the LRU information to be identified for loading in a memory address of new information upon the presence of a LOAD CACHE ADR signal. The LRU information represented by the WA and WB signals also is available for input to the cache 32 to identify the location in the cache at which data is to be read into from the main memory 31, in a manner hereinafter to be discussed.

Next, assuming that the computer reads a word which is not in cache, instead of having a MATCH output, a MATCH output is generated at the output of CAM 34. This output enables main memory to load the data at the address specified at the memory address register into the cache 32. The cache location in which the data is loaded is indicated by WA and WB which represent the location of the LRU information from the last cycle. This data is then read out of the cache onto the memory data bus. The $\overline{\text{MATCH}}$ signal also in turn generates the signal LOAD CACHE ADR to load the CAM with the new memory address information in the LRU/CAM location. This, of course, occurs before the BUMP LR signal causes the priority register to store the new order of priority.

If REMSV is false, no invalidity of the address occurs and then the priority of the signals is changed so that the previous least recently used location in the CAM is provided with the new memory address and made the most recently used location and the previous next to least recently used location is now denoted as the LRU location.

Next, assuming that a WRITE condition exists, if the memory address information is not matched in the CAM 34, the data is written into the main memory address from the memory address register, but the priority register is not changed at all.

This, however, will not be the case when a MATCH occurs in the CAM during a WRITE operation. Again, a loading of the main memory 31 at the memory address from the memory address register. As may be seen with reference to FIGS. 4 and 6 at the MEM SEL signal, the memory address register is loaded. If a match occurs, the signals RASV and RBSV denoting the CAM location of the match cause that location to be made the LRU location upon the presence of a REMSV signal. At the same time, the $\overline{\text{REMOVE}}$ signal at the twelfth bit of the memory address register causes the address loaded into the CAM at that location where a match occurred to be invalidated, as the same memory address has now been used for a write entry.

An interesting aspect of the machine may be seen with reference to when the computer would say "write something in a location" and then "read from that same location." What happens to the priority table in this case is that it never changes. For example, if one would consider the case where the computer reads that location, it puts the address read in the CAM and makes it the most recently used in terms of priority. The immediately next period when it goes to write in that same memory address location, it determines that the memory address location is in the CAM and invalidates that location to make it the least recently used in the priority truth table. The next occasion it goes to read that same location, it will now read from the same main memory address and load that CAM location (which is now the least recently used) and make it the most recently used location. As is evident here, the sequence goes back and forth, but what is important is that the other entries in the other three addresses in the CAM are undisturbed so that once a program stream is finished with this sort of re-cycling operation, it can proceed with previously stored information occurring before the re-cycling already in the cache.

It should be noted, that in a "power-up" condition, all the data in the cache is automatically invalidated by automatically setting all valid bits to false. This is effected for the reason that when power-up condition occurs, because of the fact that the cache and CAM used are semi-conductor memories and therefore will power up in a random state. It should be evident from the occurrence of power-up, that although the CAM is completely invalidated, it is forced to a pseudo-priority so that one can never have the same two words in cache simultaneously.

This occurs as a consequence of the proper use of the determinations MATCH and $\overline{\text{MATCH}}$, whereby in a CAM match, the order of priority of the addresses already within the priority register is properly updated by the ROM's 44 and 45 which consider the new location of the newly entered memory address which caused the $\overline{\text{MATCH}}$ signal to occur.

As may be observed from the above, the two bits RASV and RBSV comprise information for causing the ROM's 44 and 45 to the arrangement of the order of word state priority stored in the priority register, whereas the last bit or REMSV is used to invalidate, if necessary, information stored in a specific location of the CAM.

What is claimed is:

1. In a digital computer system having a main memory means operable at a first speed, a high speed buffer means operating at a second and higher speed for tem-

porarily storing selected portions of the main memory means, and associative memory means for temporarily storing selected main memory addresses and comparing the stored addresses with a newly received address in a read/write operation to generate comparison data, the improvement comprising

read only memory means having a bit configuration representing an algorithm and connected to said associative memory means and responsive in a read operation to both said comparison data and data representative of a previous order of priority for said stored address, to provide an output representing a new order of priority for the memory addresses stored in the associative memory means, and;

storage means connected from said read only memory means for storing said output and connected for subsequent feed back to said read only memory means as the previous order of priority.

2. In a digital computer system according to claim 1 including

logical circuit means responsive to a write operation in main memory and a comparison output indicative of an associative memory matched address comparison for generating an output, and;

said read only memory means responsive to said logical circuit means output, for defining the matched address location in the associative memory means as the least recently used location during a successive read operation.

3. In a digital computer system according to claim 2 including

invalidating means for invalidating the address stored in the associative memory means and identified as the least recently used in response to an output generated by said logical circuit means output.

4. In a digital computer system according to claim 1 wherein said means for generating comparison data includes logic means for identifying for a matched address, both its presence and the associative memory means location.

5. In a digital computer system according to claim 4 where the logic means includes

register means connected to said read only memory means for storing the location identified in the associative memory means of a matched address from a first read/write cycle for a subsequent read/-write cycle.

6. In a digital computer system according to claim 1 wherein the output representing the new order of priority provided by said read only memory means is defined by

a first set of signals on a first set of leads connected to said storage means, denoting an order of priority of the memory addresses in the associative memory means, and;

a second set of signals on a second set of leads connected to said storage means and associative memory means, denoting the least recently used location of the associative memory means.

7. In a digital computer system according to claim 6 wherein said first set of leads is connected to said storage means and said second set of leads is connected to said associative memory means.

* * * * *

United States Patent [19]

Houseman et al.

[11] **Patent Number: 4,559,618**

[45] **Date of Patent: Dec. 17, 1985**

[54] **CONTENT-ADDRESSABLE MEMORY MODULE WITH ASSOCIATIVE CLEAR**

[75] Inventors: **David L. Houseman**, West Chester, Pa.; **Paul Bowden**, Raleigh, N.C.

[73] Assignee: **Data General Corp.**, Westborough, Mass.

[21] Appl. No.: **417,801**

[22] Filed: **Sep. 13, 1982**

[51] **Int. Cl.⁴** **G11C 13/00**

[52] **U.S. Cl.** **365/49**; 365/230

[58] **Field of Search** 365/49, 230

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,997,882 12/1976 Goyal 365/49

4,296,475 10/1981 Nederlof et al. .... 365/49 X

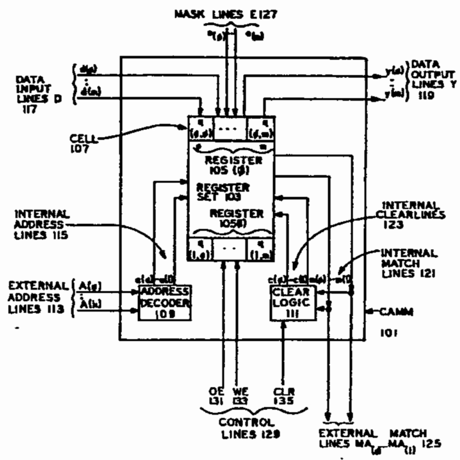
Primary Examiner—Joseph A. Popek

Attorney, Agent, or Firm—Gerald Cechony; Joel Wall

[57] **ABSTRACT**

A content-addressable memory module which performs an associative clear operation in response to a clear signal provided on a clear line. The associative clear operation simultaneously clears all registers in the content-addressable memory module whose contents match bits in a pattern input to the content-addressable memory module. A mask input along with the pattern determines which bits of the pattern are significant for the match. Each register in the content-addressable memory module has a bidirectional match line associated with it. A register's bidirectional match line carries a match signal only if that register contains data matching the pattern bits specified by the mask and the bidirectional match line is receiving a match signal from an external source. Clearing logic associated with each register clears the register when a clear signal appears on the clear line while the register's bidirectional match line is carrying a match signal. In content-addressable memories constructed of such content-addressable memory modules, memory match lines connect match lines associated with a number of registers. The memory match line and all of the match lines connected to it carry match signals only if each of the registers associated with the match lines contains data matching the pattern and mask input to the content-addressable memory module containing the register. The content-addressable memory module further contains logic allowing the use of encoded addresses to address individual registers in the content-addressable memory module.

38 Claims, 14 Drawing Figures



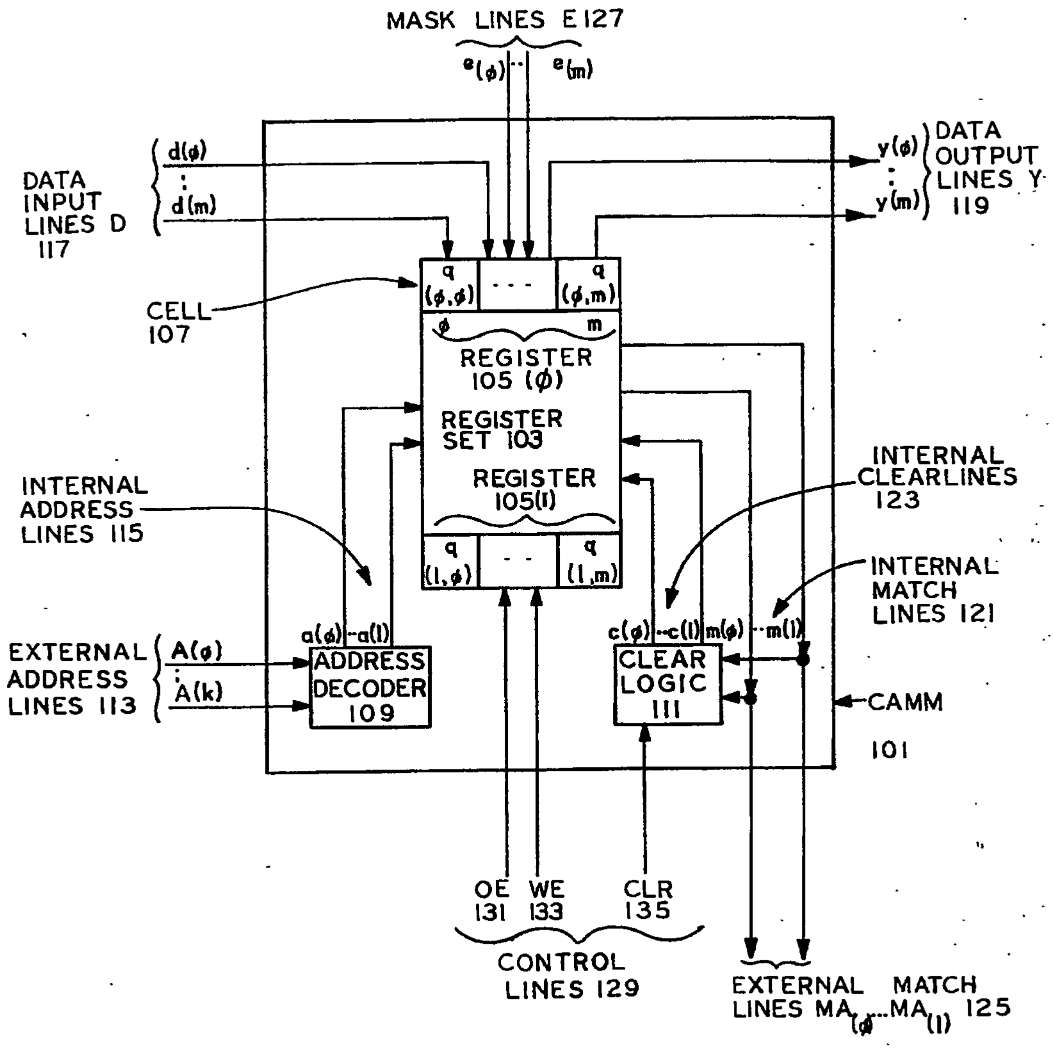
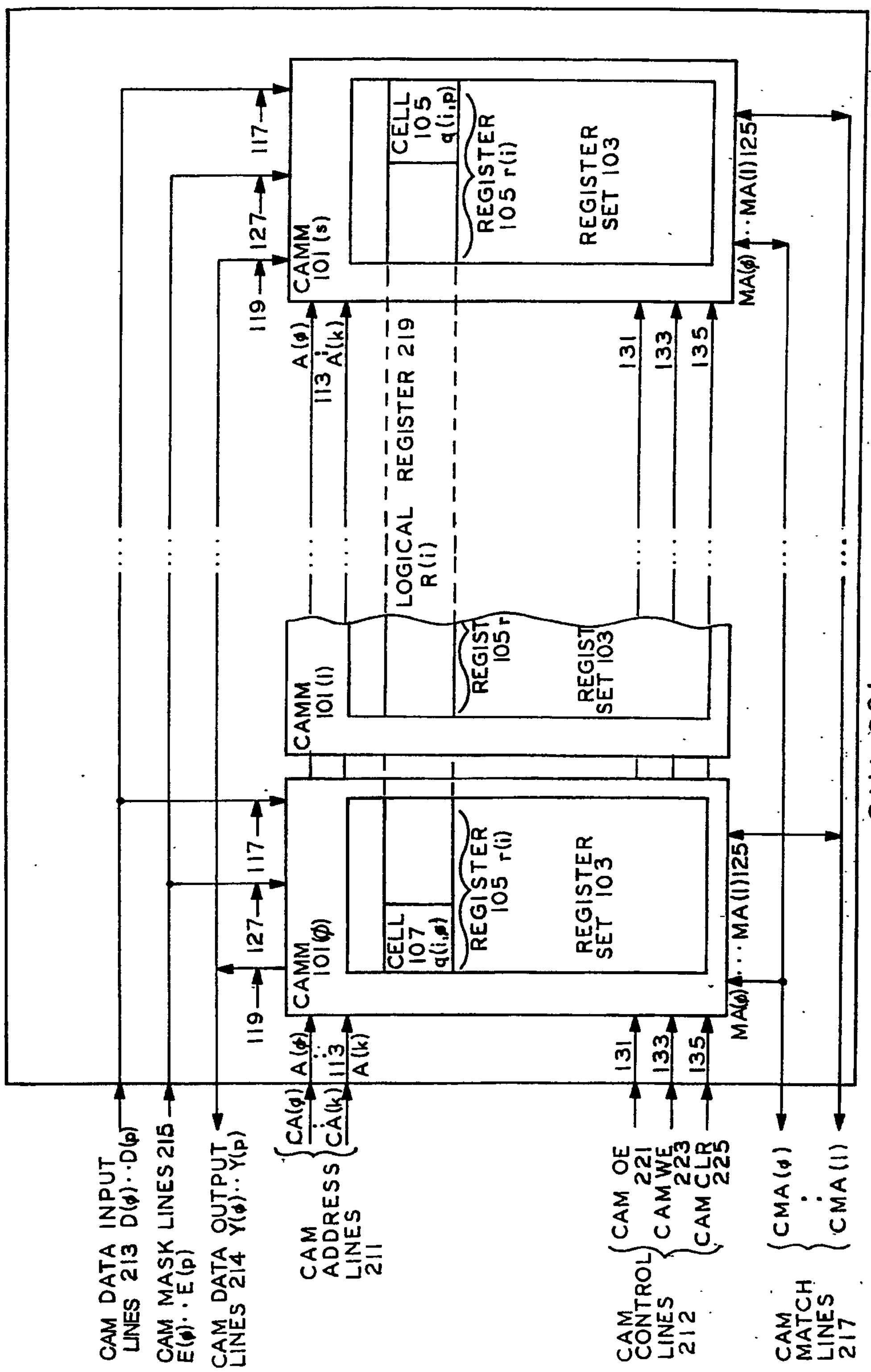

MASK LINES E127
e(φ)
e(m)
DATA INPUT LINES D 117
d(φ)
d(m)
y(φ)
y(m)
DATA OUTPUT LINES Y 119
CELL 107
q (φ,φ)
q (φ,m)
REGISTER 105 (φ)
REGISTER SET 103
REGISTER 105(l)
q (l,φ)
q (l,m)
INTERNAL ADDRESS LINES 115
INTERNAL CLEARLINES 123
INTERNAL MATCH LINES 121
EXTERNAL ADDRESS LINES 113
A(φ)
A(k)
a(φ)··a(l)
ADDRESS DECODER 109
c(φ)··c(l)
m(φ)··m(l)
CLEAR LOGIC 111
CAMM 101
OE 131
WE 133
CLR 135
CONTROL LINES 129
EXTERNAL MATCH LINES MA(φ)...MA(l) 125

FIG. 1


CAM DATA INPUT LINES 213 D(ϕ)··D(p)
CAM MASK LINES 215 E(ϕ)··E(p)
CAM DATA OUTPUT LINES 214 Y(ϕ)··Y(p)
CAM ADDRESS LINES 211
CA(ϕ)
CA(k)
CAM CONTROL LINES 212
CAM OE 221
CAM WE 223
CAM CLR 225
CAM MATCH LINES 217
CMA(ϕ)
CMA(l)
CAM 201
CAMM 101(ϕ)
CAMM 101(i)
CAMM 101(s)
CELL 107 q(i,ϕ)
CELL 105 q(i,p)
REGISTER 105 r(i)
REGISTER SET 103
REGIST 105r
REGIST SET 103
LOGICAL REGISTER 219 R(i)
119
127
117
113
A(ϕ)
A(k)
131
133
135
MA(ϕ)
MA(l) 125
FIG. 2

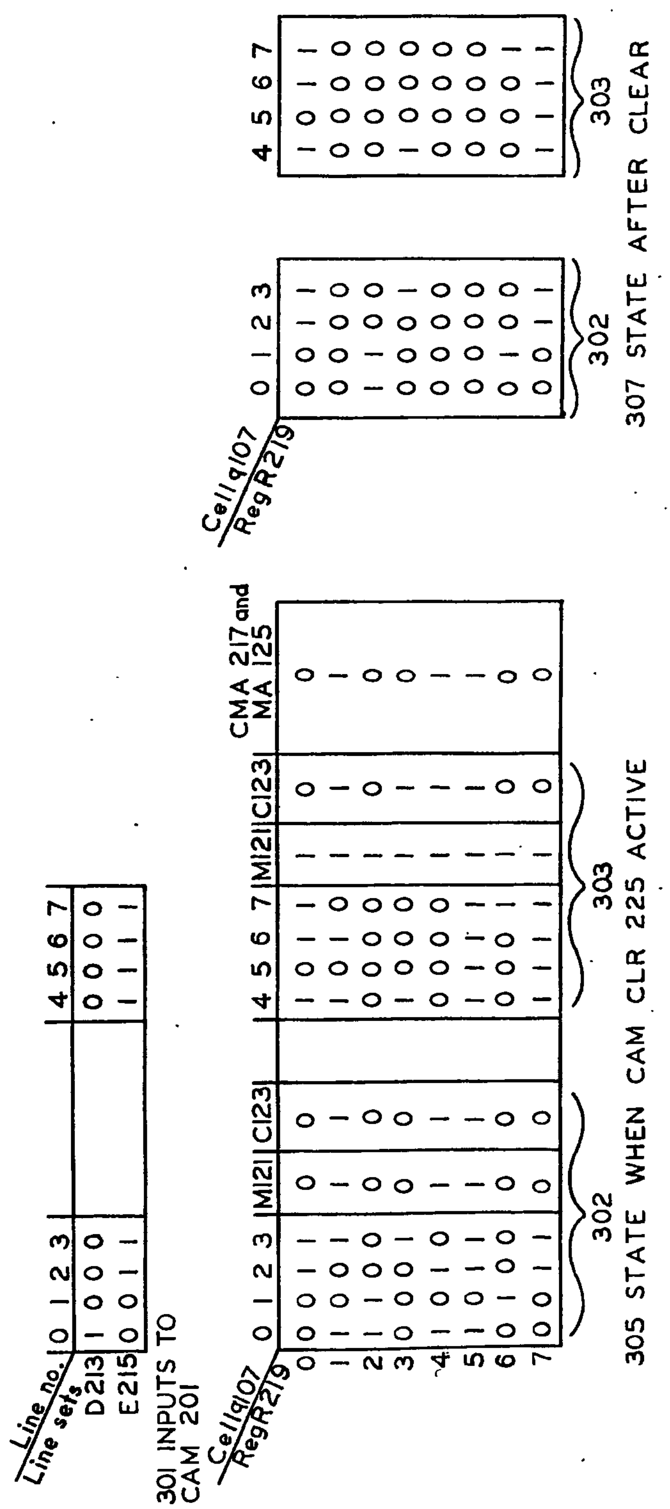

Line no.
Line sets
D213
E215
301 INPUTS TO CAM 201
Cell q107
Reg R219
MI21
CI23
CMA 217 and MA 125
302
303
305 STATE WHEN CAM CLR 225 ACTIVE
307 STATE AFTER CLEAR

FIG. 3

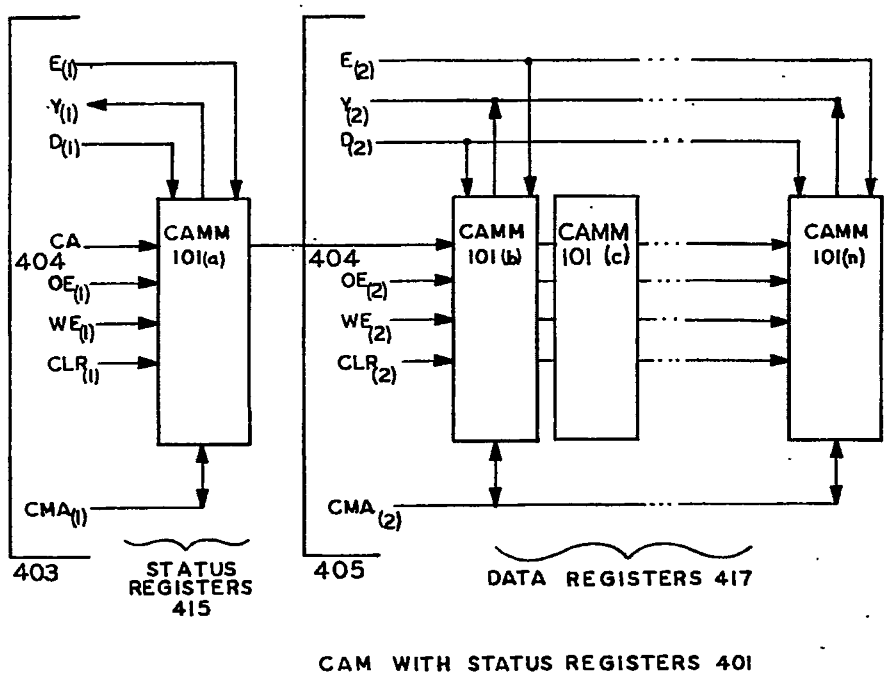

E(1)
Y(1)
D(1)
CA
404
OE(1)
WE(1)
CLR(1)
CAMM 101(a)
CMA(1)
403
STATUS REGISTERS 415
E(2)
Y(2)
D(2)
404
OE(2)
WE(2)
CLR(2)
CAMM 101(b)
CAMM 101 (c)
CAMM 101(n)
CMA(2)
405
DATA REGISTERS 417
CAM WITH STATUS REGISTERS 401

FIG. 4

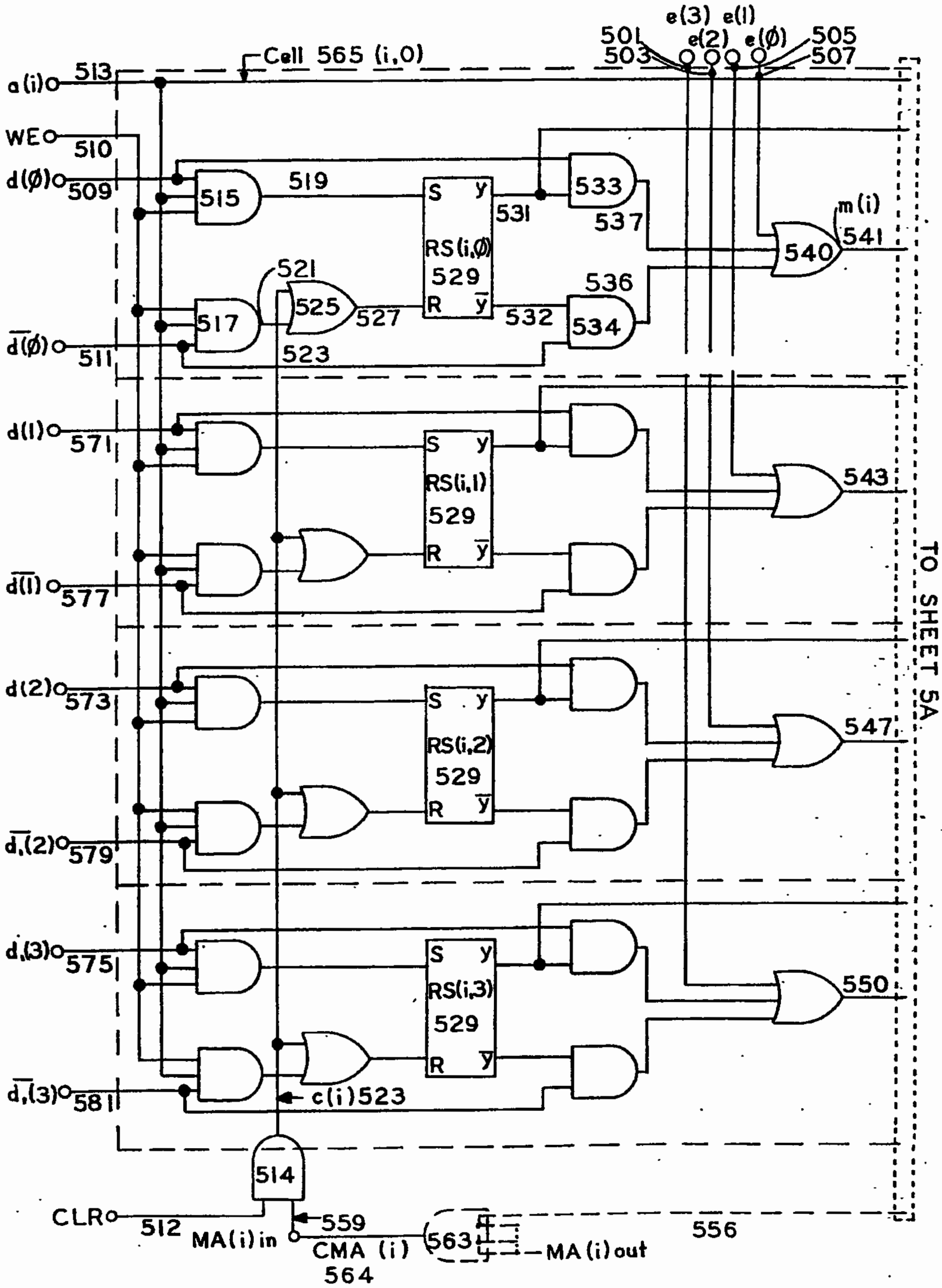

e(3) e(1)
501 e(2) e(Ø) 505
503 507
Cell 565 (i,0)
a(i) 513
WE 510
d(Ø) 509
515
519
S y
531
RS(i,Ø)
529
R ȳ
533
537
m(i)
540
541
521
525
527
532
536
534
517
d̄(Ø) 511
523
d(1) 571
RS(i,1)
529
543
d̄(1) 577
d(2) 573
RS(i,2)
529
547
d̄(2) 579
d(3) 575
RS(i,3)
529
550
d̄(3) 581
c(i) 523
514
CLR 512
MA(i) in
559
CMA (i)
564
563
MA (i) out
556
TO SHEET 5A

FIG. 5

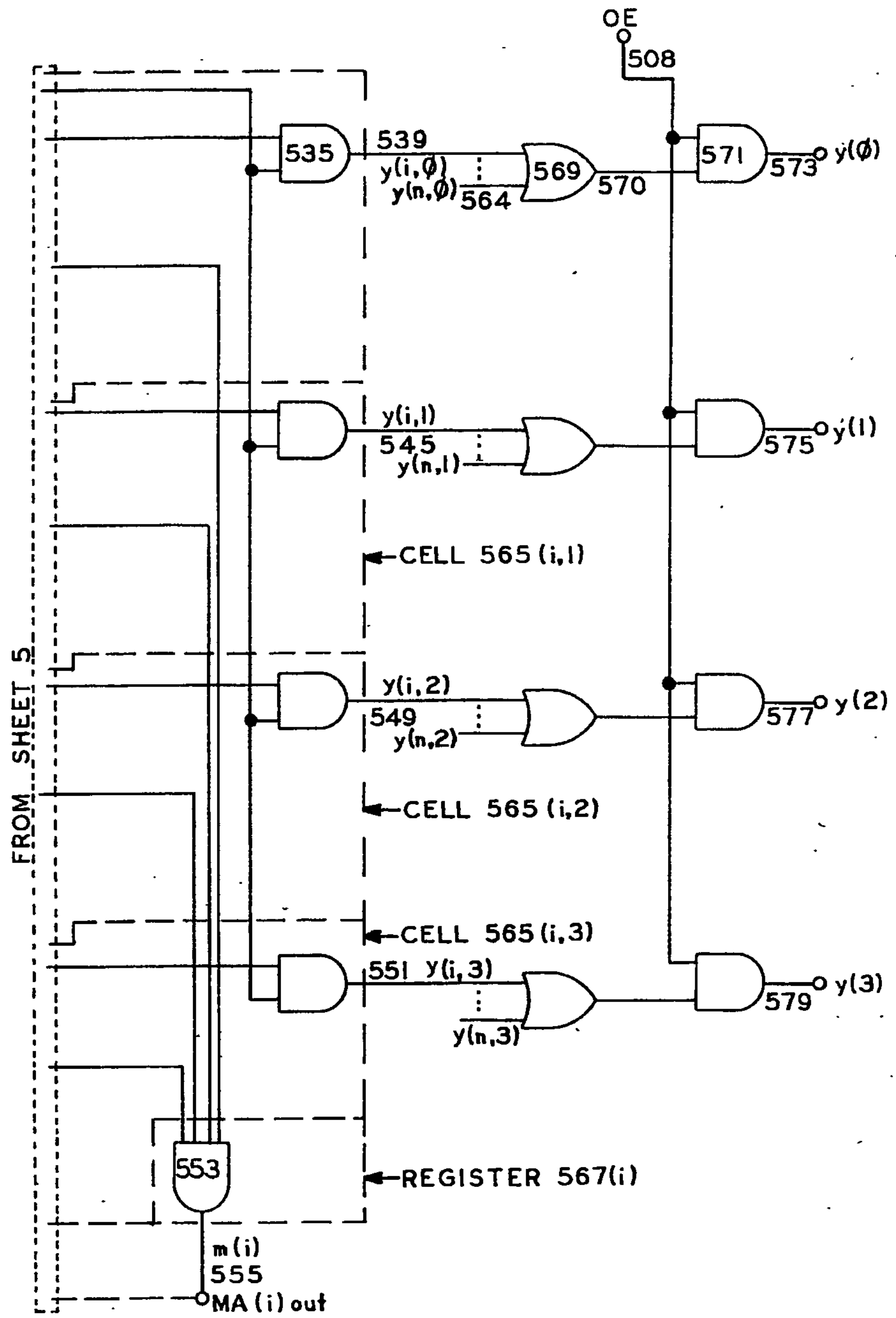
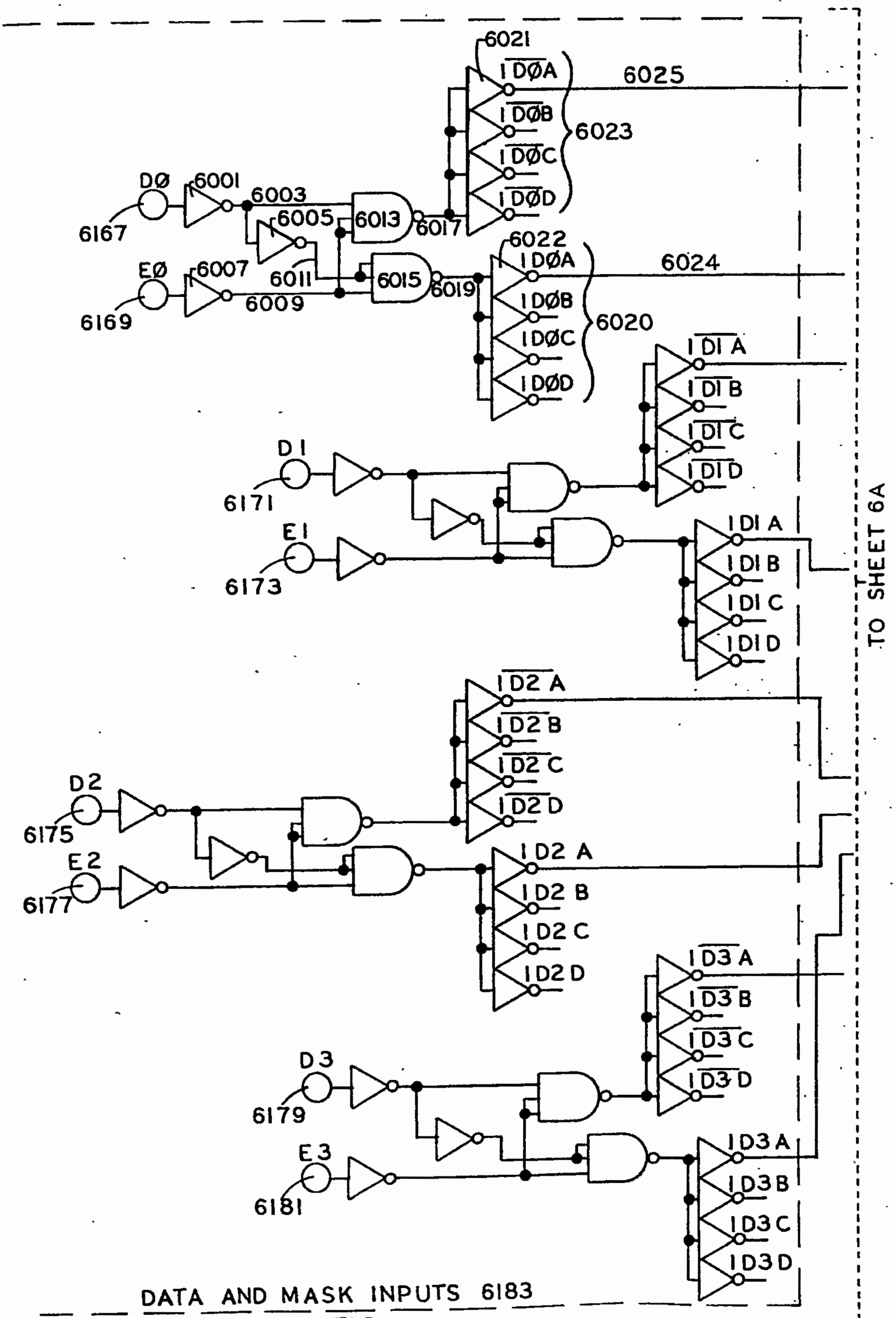
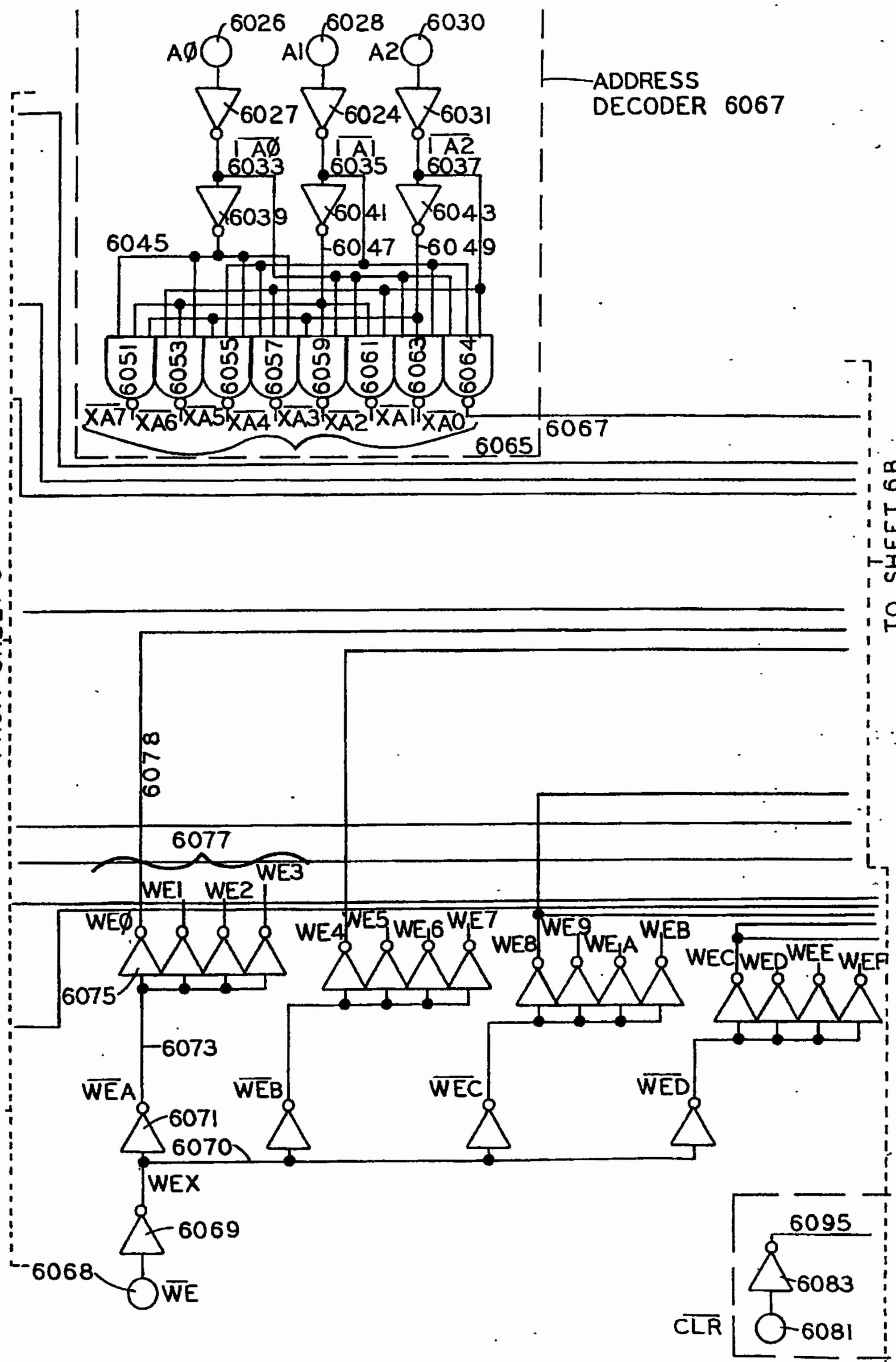

OE
508
535
539
y(i,Ø)
y(n,Ø)
564
569
570
571
573
y(Ø)
y(i,1)
545
y(n,1)
575
y(1)
CELL 565 (i,1)
y(i,2)
549
y(n,2)
577
y(2)
CELL 565 (i,2)
CELL 565 (i,3)
551 y(i,3)
y(n,3)
579
y(3)
FROM SHEET 5
553
REGISTER 567(i)
m(i)
555
MA (i) out

FIG. 5A



6021
1DØA
6025
1DØB
6023
1DØC
1DØD
DØ
6001
6003
6005
6013
6017
6167
6022
1DØA
6024
EØ
6007
6011
6015
6019
1DØB
6009
6169
6020
1DØC
1DØD
1D1 A
1D1 B
1D1 C
1D1 D
D1
6171
E1
1D1 A
1D1 B
1D1 C
1D1 D
6173
TO SHEET 6A
1D2 A
1D2 B
1D2 C
1D2 D
D2
6175
E2
6177
1D2 A
1D2 B
1D2 C
1D2 D
1D3 A
1D3 B
1D3 C
1D3 D
D3
6179
E3
1D3 A
1D3 B
1D3 C
1D3 D
6181
DATA AND MASK INPUTS 6183

FIG. 6



6026
6028
6030
AØ
A1
A2
6027
6024
6031
ADDRESS DECODER 6067
1 AØ
1 A1
1 A2
6033
6035
6037
6039
6041
6043
6045
6047
6049
6051
6053
6055
6057
6059
6061
6063
6064
XA7
XA6
XA5
XA4
XA3
XA2
XA1
XA0
6067
6065
TO SHEET 6B
FROM SHEET 6
6078
6077
WE3
WE1
WE2
WEØ
WE4
WE5
WE6
WE7
WE8
WE9
WEA
WEB
WEC
WED
WEE
WEF
6075
6073
WEA
WEB
WEC
WED
6071
6070
WEX
6069
6068
WE
6095
6083
CLR
6081

FIG. 6A

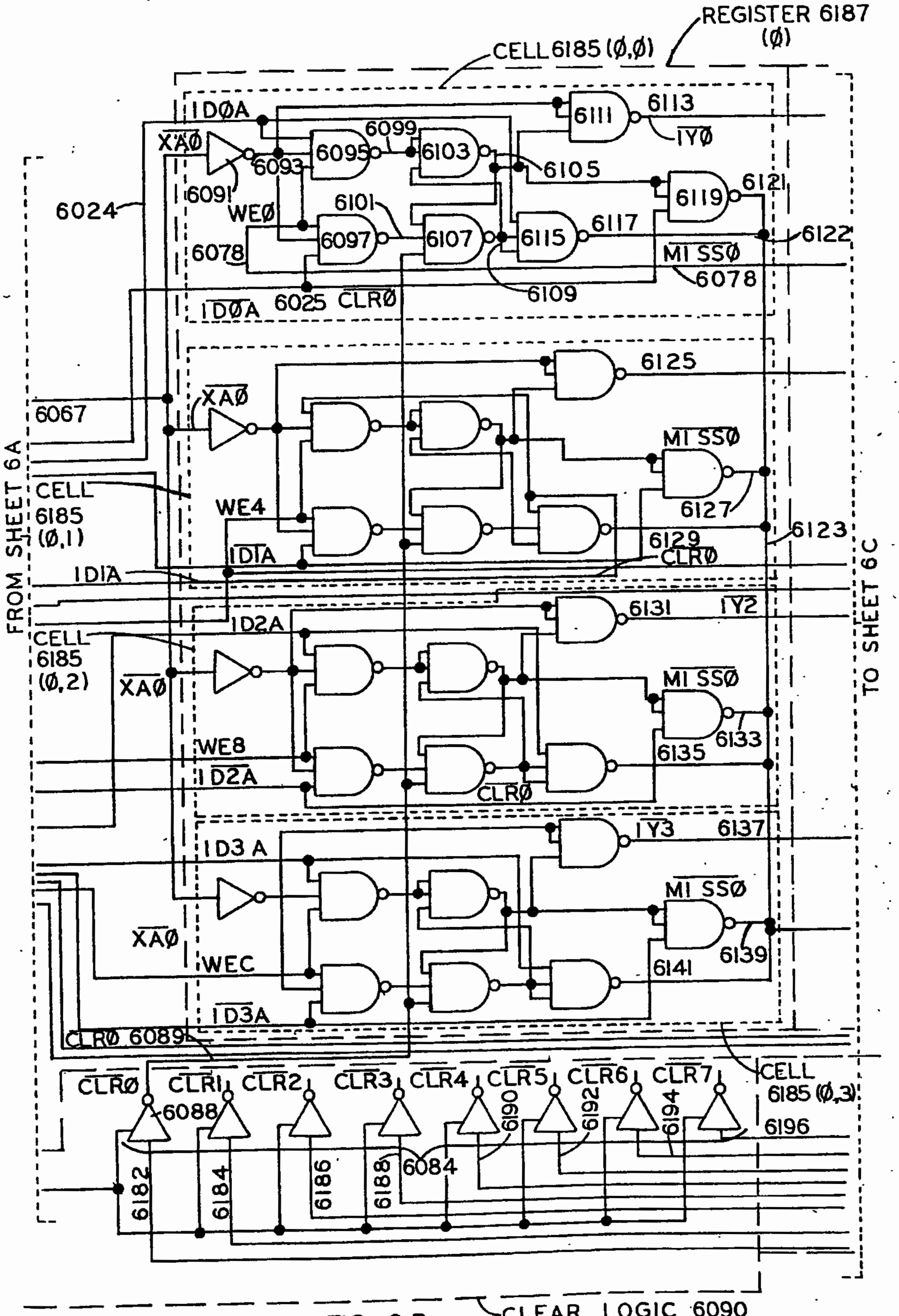

REGISTER 6187 (Ø)
CELL 6185 (Ø,Ø)
IDØA
XAØ
6091
6093
6095
6099
6103
6105
6111
6113
IYØ
6119
6121
6122
6024
WEØ
6101
6097
6107
6115
6117
6078
MI SSØ
6109
IDØA
6025
CLRØ
6125
6067
XAØ
FROM SHEET 6A
CELL 6185 (Ø,1)
WE4
MI SSØ
6127
6123
6129
IDIA
CLRØ
IDIA
6131
IY2
TO SHEET 6C
ID2A
CELL 6185 (Ø,2)
XAØ
MI SSØ
WE8
6135
6133
ID2A
CLRØ
IY3
6137
ID3 A
MI SSØ
XAØ
WEC
6141
6139
ID3A
CLRØ 6089
CLRØ
CLR1
CLR2
CLR3
CLR4
CLR5
CLR6
CLR7
CELL 6185 (Ø,3)
6088
6190
6192
6194
6196
6182
6184
6186
6188
6084
CLEAR LOGIC 6090

FIG. 6 B

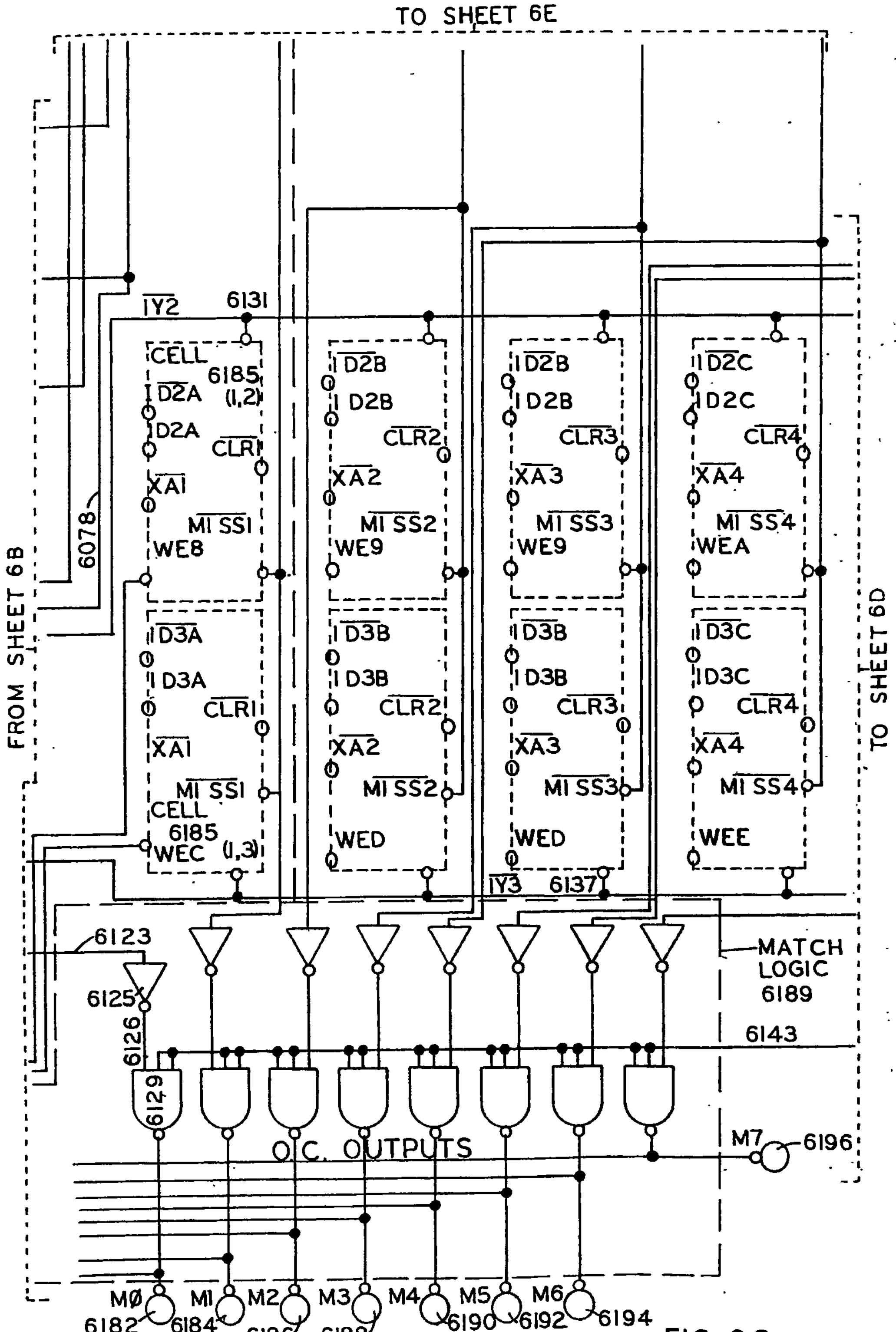

TO SHEET 6E
FROM SHEET 6B
TO SHEET 6D
6078
IY2
6131
CELL
6185
(1,2)
D2A
CLR1
XA1
MI SS1
WE8
D3A
CELL
6185
WEC
(1,3)
D2B
CLR2
XA2
MI SS2
WE9
D3B
WED
CLR3
XA3
MI SS3
D2C
CLR4
XA4
MI SS4
WEA
D3C
WEE
IY3
6137
6123
6125
6126
6129
MATCH
LOGIC
6189
6143
O.C. OUTPUTS
M7
6196
MØ
M1
M2
M3
M4
M5
M6
6182
6184
6186
6188
6190
6192
6194

FIG. 6C

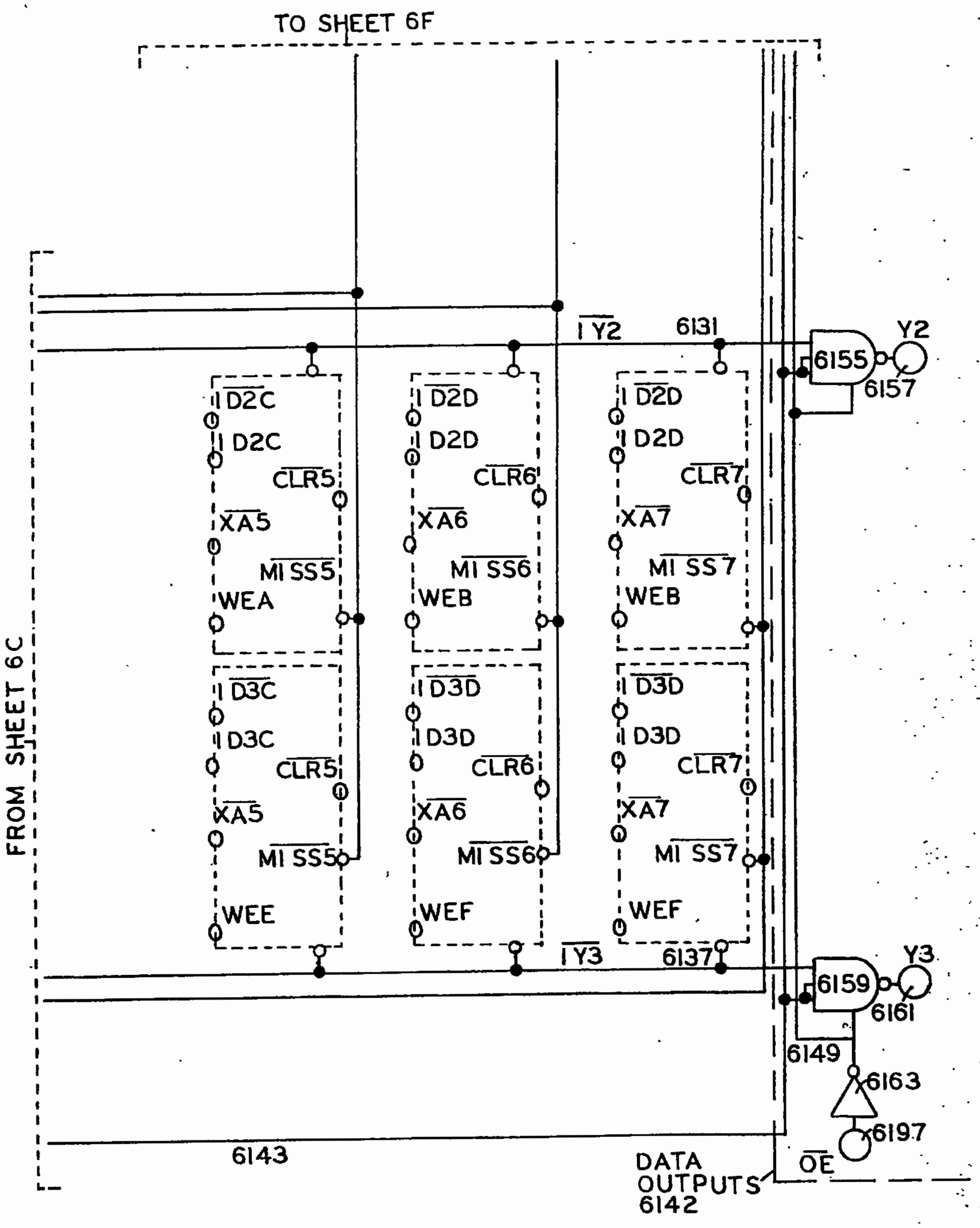
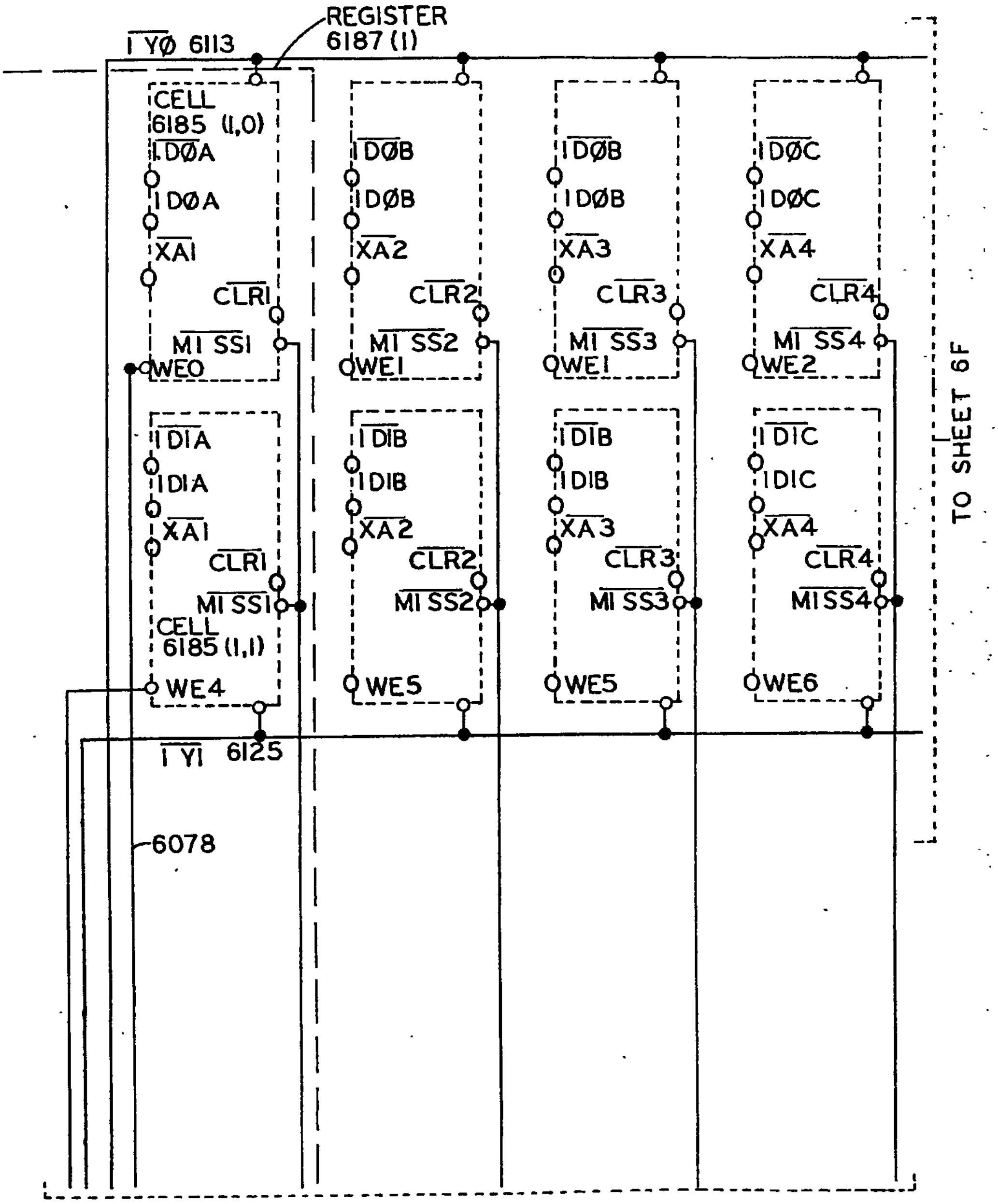

TO SHEET 6F
FROM SHEET 6C
1 D2C
1 D2C
CLR5
XA5
MISS5
WEA
1 D2D
1 D2D
CLR6
XA6
MISS6
WEB
1 D2D
1 D2D
CLR7
XA7
MISS7
WEB
1 D3C
1 D3C
CLR5
XA5
MISS5
WEE
1 D3D
1 D3D
CLR6
XA6
MISS6
WEF
1 D3D
1 D3D
CLR7
XA7
MISS7
WEF
1 Y2
6131
6155
Y2
6157
1 Y3
6137
6159
Y3
6161
6149
6163
6197
OE
6143
DATA
OUTPUTS
6142

FIG. 6 D



REGISTER 6187 (1)
IYØ 6113
CELL 6185 (1,0)
IDØA
IDØA
XA1
CLR1
MISS1
WE0
IDØB
IDØB
XA2
CLR2
MISS2
WE1
IDØB
IDØB
XA3
CLR3
MISS3
WE1
IDØC
IDØC
XA4
CLR4
MISS4
WE2
ID1A
ID1A
XA1
CLR1
MISS1
CELL 6185 (1,1)
WE4
ID1B
ID1B
XA2
CLR2
MISS2
WE5
ID1B
ID1B
XA3
CLR3
MISS3
WE5
ID1C
ID1C
XA4
CLR4
MISS4
WE6
IY1 6125
6078
TO SHEET 6F
FROM SHEET 6C

FIG. 6E

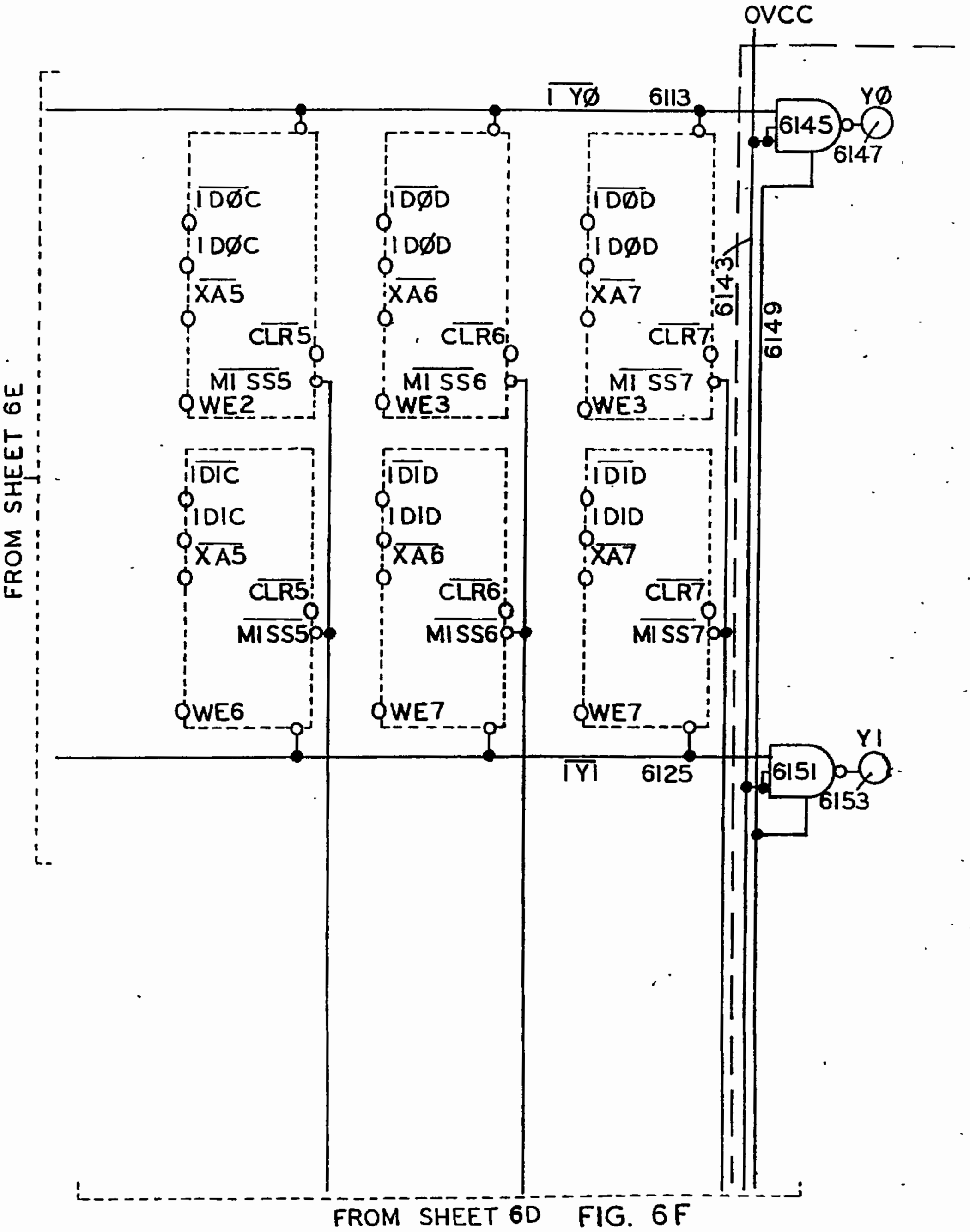

OVCC
I YØ
6113
6145
YØ
6147
IDØC
IDØC
XA5
CLR5
MISS5
WE2
IDØD
IDØD
XA6
CLR6
MISS6
WE3
IDØD
IDØD
XA7
CLR7
MISS7
WE3
6143
6149
IDIC
IDIC
XA5
CLR5
MISS5
WE6
IDID
IDID
XA6
CLR6
MISS6
WE7
IDID
IDID
XA7
CLR7
MISS7
WE7
IYI
6125
6151
YI
6153
FROM SHEET 6E
FROM SHEET 6D

FIG. 6F

VALUES ON ADDRESS LINES			NAND GATES AND INPUT LINES							
			6064	6063	6061	6059	6057	6055	6053	6051
A0	A1	A2	6033 6035 6037	6033 6035 6049	6033 6047 6037	6033 6047 6049	6045 6035 6037	6045 6035 6049	6045 6047 6037	6045 6047 6049
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

ADDRESS DECODER 6066 TRUTH TABLE

FIG. 7

CONTENT-ADDRESSABLE MEMORY MODULE WITH ASSOCIATIVE CLEAR

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to memory circuits for use in digital computer systems and more specifically to content-addressable memory circuits.

2. Description of Prior Art

In the prior art, content-addressable memory modules (CAMMs) have been developed which perform match operations in addition to the read and write operations performed by standard memory circuits. In read and write operations, memory modules respond to addresses. In the read operation, an address is presented to the memory module and the memory module returns the data stored at that address; in the write operation, an address and data are presented to the memory module and the data is stored at the address.

In the match operation, on the other hand, an item of data is input to a CAMM, and if a matching item of data is contained in the CAMM, the CAMM indicates its location by activating a match line corresponding to the register containing the matching item of data. The degree of match required to activate the match line may be controlled by presenting a CAMM with mask bits as well as with the input data. Each mask bit corresponds to an input data bit; if the mask bit is set, the corresponding input data bit is ignored when data in the registers is compared with the item of data presented to the CAMM. Examples of such prior art CAMMs are the Intel(R) 3104, the Signetics 10155, and the Fairchild F100142. Such CAMMs are generally designed so that they may be easily combined together to form content-addressable memories (CAMs). A CAM has the same properties as a CAMM, except that a single CAM register is made up of a corresponding register from each of the CAMMs making up the CAM.

CAMs as described above may be used in digital computer systems to construct caches allowing fast access to frequently-used values by means of keys representing the values. For example, an operand in an instruction stream may contain information from which a memory address may be calculated. Once the memory address has been calculated, the memory address may be loaded into a cache and the operand may be used as a key to access the memory address in the cache. Such a cache may be constructed by combining a CAM with a fast-access memory. In the combination, each register of the fast-access memory may correspond to a register of a CAM, and a match line from the CAM register may serve to address the corresponding register of the fast-access memory. The CAM registers contain operands, and the corresponding registers of the fast-access memory contain the memory addresses corresponding to the operands. When an operand appears in the instruction stream, it is presented to the CAM. If the CAM contains the operand, the match line for the CAM register containing the operand becomes active and thereby addresses the corresponding register of the fast-access memory. The fast-access memory then responds by providing the memory address contained in the corresponding register. If the CAM does not contain the operand, a fault occurs to which the digital computer system responds by calculating the memory address represented by the operand and loading the operand into a CAM register and the memory address into the corresponding register of the fast-access memory.

The use of prior-art CAMs in applications such as that just described has been hindered by the amount of time required to clear the registers of prior-art CAMs. Such clearing is often necessary when a call or return operation is performed or when one process is removed from a processor and another loaded onto a processor. Such operations occur frequently in modern digital data processing systems, and the amount of time required to perform them has an important impact on overall system performance. In CAMs of the prior art, a register may be cleared only by performing a write operation to the register to be cleared. Thus, clearing an entire CAM requires separate write operations to each register in the CAM and clearing a CAM entry for a given operand requires presenting the operand to the CAM to obtain the address of the register containing the CAM and then performing a write operation to the register specified by the address.

The foregoing problem of the prior art and other problems as well are solved by the the invention described below.

SUMMARY OF THE INVENTION

The present invention provides a CAMM in which all registers which contain data matching a pattern input as modified by a mask input are simultaneously cleared when a clear signal is received in the CAMM. The mask input modifies the pattern input by specifying that certain bits of the pattern input be ignored when testing for a match between the pattern input and data stored in the registers. If the mask input specifies that all bits of the pattern input are to be ignored, all data contained in the registers matches the pattern input and all registers of the CAMM are simultaneously cleared on receipt of the clear signal.

The CAMM includes input lines for receiving data to be stored in the registers and the pattern input, mask input lines for receiving a mask, a clear line for receiving a clear signal, registers for storing data, and bidirectional match lines associated with each register for providing and receiving a match signal. The bidirectional match lines carry a match signal only when the register associated with the match line contains stored data matching the pattern input and the match line is simultaneously receiving a match signal from an external source.

The registers have three principal components: logic forming flip-flops for storing individual bits of data, match detection logic responsive to the data stored in the register, the data input lines, and the mask input lines for detecting a matching data item and providing a match signal to the bidirectional match line associated with the register, and clearing logic responsive to the clear line and the bidirectional match line for clearing the register in response to the simultaneous occurrence of a match signal on the bidirectonal match line and a clear signal on the clear line.

CAMMs of the present invention may be combined to form CAMs with the properties of the CAMM. In such CAMs, clear lines from the CAMMs making up the CAM are connected to a memory clear line and match lines from registers in the CAMMs are connected to memory match lines. A memory match line carries a match signal only if all match lines connected to the memory match line are providing match signals. Consequently, the match lines connected to a memory match line provide a match signal to the clearing logic only if the match detection logic of each register in the CAM register detects a match. CAMM registers whose match lines are connected to a common memory match line are therefore cleared only if each of the registers connected to the memory match line contain data matching the pattern input to the CAMM containing that register.

It is thus an object of the present invention to provide an improved digital computer system.

It is a further object of the present invention to provide an improved CAMM for use in digital computer systems.

It is another object of the present invention to provide a CAMM having an associative clear operation.

It is a still further object of the present invention to provide a CAMM wherein all CAMM registers may be simultaneously cleared.

It is yet another object of the present invention to provide a CAMM wherein a set of CAMM registers may be simultaneously cleared.

It is a yet further object of the present invention to provide a CAMM having encoded addressing.

It is still another object of the present invention to provide an improved CAM.

It is a yet further object of the present invention to provide a CAM having an associative clear operation.

It is a final object of the present invention to provide a CAM wherein sets of registers or the entire CAM may be simultaneously cleared.

Other objects, advantages, and features of the present invention will be understood by those of ordinary skill in the art after referring to the following detailed description of the preferred embodiment and drawings, wherein:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing an illustrative embodiment of a content-addressable memory module having the properties of the present invention;

FIG. 2 is a block diagram showing an illustrative embodiment of a content-addressable memory module employing content-addressable memory modules having the properties of the present invention;

FIG. 3 is a representation of the contents of a content addressable memory employing content-addressable memory modules having the properties of the present invention before and after a clear operation;

FIG. 4 is a block diagram showing a second illustrative embodiment of a content-addressable memory employing content-addressable memory modules having the properties of the present invention;

FIG. 5 and 5A are a simplified logic diagram of a single register of a preferred embodiment of the content-addressable memory module of the present invention;

FIGS. 6 and 6A through 6F together make up a complete logic diagram of a TTL gate array implementation of a preferred embodiment of a content-addressable memory module of the present invention; and

FIG. 7 is a truth table showing the decoding of the encoded addresses used in the TTL gate array implementation of FIG. 6.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

1 Introduction

In the following description of the preferred embodiments of the present invention, content-addressable memories are first described in general. Next functional descriptions of a content-addressable memory module of the present invention and of content-addressable memories formed from content-addressable memory modules of the present invention are presented. Finally, a detailed implementation of a content-addressable memory module of the present invention is disclosed.

1.1 General Description of Content Addressable Memories

A content-addressable memory (CAM) is a memory which not only stores data, but also performs a match operation. In this operation, the CAM is given an item of data as input and if the CAM contains a matching item of data, i.e., one in which the values of certain bits are the same as that of corresponding bits of the item of data provided as input, the CAM indicates which register of the CAM contains the matching data. In many CAMs, a mask input selects the bits of the input data which are compared with the corresponding bits of the data contained in the CAM register. A data item stored in a CAM register matches the input data if the bits in the data item in the CAM register corresponding to the bits of the input data item selected by the mask input match the selected bits of the data input item. Other bits in the data item stored in the CAM do not affect the match.

1.2 CAMs of the Present Invention

Besides performing match operations with or without masking, CAMs of the present invention perform an associative clear operation. In a clear operation, all bits in a register of a CAM are set to 0; in an associative clear operation, all bits in a given register of a CAM are set to 0 if there is a match between the data input to a CAM as masked by the mask input and the contents of a given CAM register. Finally, CAMs of the present invention perform read and write operations like those of standard memories.

2 Content-Addressable Memory Modules of the Present Invention—FIG. 1

A CAM of the present invention may include one or more CAM modules (CAMMs). Referring to FIG. 1, there is disclosed a block diagram of a single CAMM 101 of the present invention. CAMM 101 contains a plurality of registers 105 for storing data. CAMM 101 further receives inputs of data to be stored in registers 105 from data input lines 117, masking inputs from mask lines 127, addresses of registers 105 from external address lines 113, and control signals from control lines 129. Control lines 129 include output enable (OE) line 131 for enabling output of data fom CAMM 101, write enable (WE) line 133 for enabling the storage of data on data input lines 117 in CAMM 101, and clear (CLR) line 135 for enabling the associative clearing of registers 105. CAMM 101 provides outputs of data stored in registers 105 on data output lines 119. Finally, CAMM 101 both receives inputs and provides outputs on bidirectional external match lines 125. Each external match line 125 corresponds to a register 105 in CAMM 101 and a external match line 125 may be connected to external match lines 125 of other CAMMs 101. The input received on a external match line 125 for a given register 105 indicates whether the contents of registers 105 of other CAMMs 101 whose external match lines 125 are connected to the external match line 125 of a given CAMM register 105 match the data inputs to those CAMMS 101 as masked by the mask inputs. The output of an external match line 125 for a given register 105 indicates whether the contents of that register matches the data and mask inputs received by its CAMM 101.

3 Internal Structure of CAMM 101

Internally, CAMM 101 is made up of register set 103 consisting of registers 105, address decoder 109 for decoding addresses of registers 105 received on external address lines 113, internal address lines 115 for transmitting decoded addresses from address decoder 109 to registers 105, clear logic 111 for performing the associative clear operation, internal match lines 121 for transmitting match signals between registers 105, clear logic 111, and external match lines 125, and internal clear lines 123 for transmitting clear signals between clear logic 111 and registers 105.

Each register 105 consists of a plurality of cells 107 for storing a single bit of data. Each cell 107 in a given register 105 corresponds to a single data input line 117, a single data output line 119, and a single mask line 127. Thus, if each register 105 has 0 . . . m cells 107, there are 0 . . . m data input lines 117, data output lines 119, and mask lines 125. In FIG. 1, the plurality of data input lines 117 is indicated by d(0) . . . d(m), the plurality of mask lines by e(0) . . . e(m), and the plurality of data output lines by y(0) . . . y(m). Data input line d(0) carries data to cell 107 (0) of a register 105 specified by an address on external address lines 113, data output line y(0) carries data from cell 107 (0) of a register 105 specified by an address, and mask line e(0) masks data input line d(0).

Each register 105 corresponds to a single internal address line 115, a single internal match line 121, and a single internal clear line 123. In FIG. 1, the plurality of registers 105 is indicated by r(0) . . . r(1), the plurality of internal address lines 115 by a(0) . . . a(1), the plurality of internal match lines 121 by m(0) . . . m(1), the plurality of internal clear lines 123 by c(0) . . . c(1), and the plurality of external match lines 125 by MA(o) . . . MA(1). If i is in 0 . . . 1, then internal address line 115 a(i), internal match line 121 m(i), internal clear line 123 c(i), and external match line 125 MA(i) all correspond to register r(i) 105. Further, a given cell 107 in registers 105 is indicated by q(i,j), where i specifies register 105 to which cell 107 q belongs and j specifies a single cell of 107 of cells 107 0 . . . m in register i. Thus, cell 107 (0) of register 105 r(1) is specified by q(1,0).

Internal match line 121 m(i) and external match line 125 MA(i) are related as follows: if either is inactive, the other is also inactive. Internal match line 121 m(i) is inactive if its corresponding register 105 r(i) does not match the data on data input lines 117 as masked by the inputs on mask lines 125. The electrical properties of external match lines 125 are such that corresponding external match lines from a plurality of CAMMs 101 may be connected together; since each such connected external match line 125 MA(i) is inactive if its corresponding internal match line 121 m(i) is inactive, all such connected external match lines 125 MA(i) are inactive if any of the corresponding internal match lines 121 m(i) is inactive, and if an external match lines 125 MA(i) is inactive, all internal match lines 125 m(i) connected thereto are also inactive. In logical terms, therefore, the state of an external match line 125 MA(i) is the logical product of the states of all internal match lines 121 m(i) in the CAMMs 101 whose external match lines 125 are connected.

Clear logic 111 determines the state of an individual clear line 123 c(i) in response to external match line 125 MA(i) and CLR line 125. If external match line 125 MA(i) and CLR 135 are simultaneously active, clear logic 111 actives clear line 123 c(i), thereby setting cells 107 q(i,0 . . . m) of register 105 r(i) to a value indicating a binary 0. As mentioned above, external match line MA(i) is active only if its corresponding internal match line m(i) is active. Where external match lines 125 MA(i) of a plurality of CAMMs 101 are connected together, therefore, no register 105 r(i) in any of the plurality of CAMMs 101 is cleared unless internal match lines m(i) 121 in all of the plurality of CAMMs 101 are active, that is, unless the contents of each register 105 r(i) in the plurality of CAMMs 101 matches the inputs on data input lines 117 as masked by mask lines 125 in that CAMM 101.

External address lines 113 consist of a plurality of address lines A(0) . . . A(k) which transmit a binary encoded address specifying a register 105 to address decoder 109. Address decoder 109 decodes the address and activates internal address line 115 corresponding to register 105 specified on external address lines 113. For example, in a CAMM 101 with 8 registers 105, the external address lines 113 may consist of lines A(0) . . . A(2) and internal address lines 115 may consist of lines a(0) . . . a(7). The three external address lines 113 allow a binary representation of the integers 0 through 7 and address decoder 109 decodes this binary representation and activates internal address line 115 for register 105 specified by the integer represented by external address lines 113.

4 Operations Performed by CAMM 101

As mentioned above, CAMM 101 performs four operations: a read operation, a write operation, a match operation, and a clear operation. In a read operation, OE 131 is active, external address lines 113 specify a register 105 r(i), and data output lines 119 y(0) . . . y(m) are set to the values of cells 105 q(i,0) . . . q(i,m). In a write operation, WE 133 is active, external address lines 113 specify a register 105 r(i), and cells 105 q(i,0) . . . q(i,m) are set to the values on data input lines 117 d(0) . . . d(m).

In a match operation, WE 133 and CLR 135 are both inactive. The inputs are data on data lines 117 d(0) . . . d(m) and mask enable signals on mask lines 127 e(0) . . . e(m). If a mask line 127 e(j) is active, then the value of data line 117 d(j) is disregarded when testing for a match. If the contents of cells 107 q(i,0) . . . q(i,m) for a given register 105 r(i) match all values on data lines 117 d(0) . . . d(m) which are not masked by active mask lines 127, then internal match line 121 m(i) becomes active. In logical terms, this may be defined as follows:

$$m(i) = \underset{j=0}{\overset{m}{P}} [(q(i,j) \cdot d(j)) + e(j)]$$

where P is the logical product.

In the associative clear operation, finally, WE 133 is inactive and CLR 135 is active. As previously mentioned, if CLR 135 c(i), internal match line 121 m(i), and external match line 125 MA(i) are all active, match and clear logic 111 clears register r(i). Since external match line 125 MA(i) is active only if internal match lines 121 m(i) for all CAMMs 101 whose external match lines 125 MA(i) are connected together are active, a clear takes place only if there are matches for all CAMMs 101 whose external match lines 125 MA(i) are connected.

3 CAMs Composed of CAMMS 101—FIG. 2

In most applications, an individual CAMM 101 like the one just described is combined with other CAMMs 101 to make a CAM. FIG. 2 is a block diagram representing a CAM 201 made up of a plurality of CAMMs 101. Inputs to CAM 201 include data on CAM data input lines 213, masks on CAM mask lines 215, control signals on CAM control lines 211, and encoded addresses on CAM address lines 211. Outputs include data on CAM data output lines 214 and CAM match signals on CAM match lines 217.

4.3.1 Behavior of CAM 201

The behavior of CAM 201 is determined by the manner in which CAMMs 101 making up CAM 201 are connected by CAM address lines 211, CAM control lines 212, and CAM match lines 217. CAM address lines 211 CA(0) . . . CA(k) are connected to external address lines 113 A(0) . . . A(k) of all CAMMs 101 in CAM 201, and consequently, an address i on CAM address lines 211 specifies register 105 r(i) in all CAMMs 101 making up CAM 201. CAM control lines 212 consist of CAM OE line 221, connected to OE line 131 of all CAMMs 101 making up CAM 201, CAM WE line 223, connected to WE line 133 of all CAMMs 101 in CAM 201, and CAM CLR line 225, connected to CLR line 135 of all CAMMs 101 in CAM 201. As a consequence of these connections, when a CAM control line in CAM control lines 212 becomes active, its corresponding control line in control lines 129 in all CAMMs 101 making up CAM 201 becomes active. CAM match lines 217 CMA(0) . . . CMA(1), finally, are connected to external match lines 125 MA(0) . . . MA(1) in all CAMMs 101 making up CAM 201. As previously explained, when external match lines 125 corresponding to a register 105 r(i) in a plurality of CAMMs 101 are connected together, a failure of the contents of a register 105 r(i) to match the values of register 205 r(i)'s data inputs 117 as masked by its mask inputs 125 deactivates its external match line 125 MA(i), and this in turn deactivates all external match lines 125 MA(i) connected to it. Consequently, CAM match line 217 CMA(i) is active only if for each register 105 r(i) in the group of CAMMs 101 forming CAM 201, the value of data inputs 117 as masked by mask inputs 127 of each register 105 r(i) matches the contents of that register 105 r(i).

As a result of these connections between CAMMs 101 making up CAM 201, corresponding registers 105 r(i) in CAMMs 101 making up CAM 201 behave as a single logical register 219 R(i), indicated by dashed lines in FIG. 2. if CAM 201 contains s CAMMs 101 and each register r(i) contains n cells 107, then logical register 219 R(i) contains sn cells 107. In FIG. 2 these cells are specified as cells 107 q(i,0) . . . q(i,p), where p=sn−1. Just as all registers 105 r(i) in CAMMs 101 making up CAM 201 form a logical register R(i) 219, so do all data input lines 117 in these CAMMs 101 form CAM data input lines 213, all data output lines 119 form CAM data output lines 214, and all mask lines 127 form CAM mask lines 215. There are as many CAM data input lines 213, CAM data output lines 214, and CAM mask lines 215 as there are cells 107 q in a logical register 219. In FIG. 2, the lines comprising CAM data input lines 213 are specified by D(0) . . . D(p), those comprising CAM data output lines 214 by Y(0) . . . Y(p), and those comprisin9 CAM mask lines 215 by E(0) . . . E(p), where p=sn−1 as before.

4.3.2 Operations Performed by CAM 201

As a conseqence of the manner in which CAMMs 101 are connected to form CAM 201, all of the reading, writing, matching, and clearing functions performed by a CAMM 101 can be performed by CAM 201.

In a read operation, CAM OE line 221 is active and CAM address lines 211 specify an address. Consequently, control line OE 131 of each CAMM 101 is active, external address lines 113 of each CAMM 101 specify a corresponding register 105 r(i), and data output lines 119 are set to the values of the cells 105 making up register 105 r(i). Since all the registers 105 r(i) together make up logical register 219 R(i), and all of the data output lines together make up CAM data output lines 214, the result is to set CAM data output lines 214 Y(0) . . . Y(p) to the values of cells 105 q(i,0) . . . q(i,p) in logical register 219 R(i). Similarly, in the write operation, CAM WE line 223 is active, CAM address lines 211 specify an address, and cells 105 q(i,0) . . . q(i,p) in logical register 219 R(i) indicated by the address are set to the values of CAM data input lines 213 D(0) . . . D(p).

In a match operation, CAM data input lines 213 D(0) . . . D(p) specify the data to be matched with the contents of logical registers 219 and CAM mask lines 215 E(0) . . . E(p) specify which bits of the data are to be ignored in determining whether there is a match. Since CAM match line 217 CMA(i) corresponding to a logical register 219 R(i) connects all external match lines 125 MA(i) for registers 105 r(i) comprising logical register 219 R(i), CAM match line 217 CMA(i) and all external match lines 125 MA(i) are deactivated as previously described if the contents of any register 105 r(i) fail to match unmasked bits on CAM data input lines 213 corresponding to the cells 105 contained in register 105 r(i). The state of CAM match line 217 CMA(i) thus indicates whether the contents of logical register 219 R(i) match the data on CAM data input lines 213 D(0) . . . D(p). In logical terms, this may be expressed as follows:

$$CMA(i) = \mathop{P}_{j=0}^{p} [(q(i,j) \cdot d(j)) + e(j)]$$

where P is the logical product as before. As may be seen from the above equation, a match operation for a logical register 219 R(i) in CAM 201 is completely equivalent to a match operation for a register 105 r(i) in CAMM 101.

The behavior of the clear operation in CAM 201 is determined by the behavior of the match operation and by the fact that CLR lines 135 of all CAMMs 101 in CAM 201 are connected to CAM CLR line 225, and consequently, all CLR lines 135 are active when CAM CLR line 225 is active. As explained in the description of CAMMs 101, a register 105 r(i) is cleared only if CLR line 135 and external match line 125 MA(i) are both active. External match line 125 MA(i) for a register 105 r(i) in a logical register 219 R(i) is active only if internal match lines 121 m(i) for all registers 105 r(i) making up logical register 219 R(i) are active. Therefore, registers 105 r(i) making up logical register 219 R(i), and thus, logical register 219 R(i) itself, are cleared only if the contents of logical register 219 R(i) match the data on CAM data input lines 213 as masked by the input on CAM mask lines 215. As with the other operations, the clear operation on a logical register 219 R(i) is thus completely equivalent to the clear operation on a register 105 r(i).

4.3.3 Example Match and Clear Operations—FIG. 3

A concrete example of a match operation and a clear operations in a CAM 201 is provided by FIG. 3. FIG. 3 shows the state of cells 107, CAM data input lines 213, CAM mask lines 215, internal match lines 121, internal clear lines 123, and CAM match lines 217 for a CAM 201 comprised of two CAMMs 101. Each CAMM 101 contains 8 4-bit registers 105, and consequently, CAM 201 of FIG. 3 contains 8 eight-bit logical registers 219. FIG. 3 represents CAM 201 as follows: Table 301 represents the inputs to CAM 201 at the time of the match and clear operations; row D corresponds to CAM data input lines 213, and row E corresponds to CAM mask lines 215; the columns specify individual CAM data input lines 213 and CAM mask lines 215. The value at the intersection of a row and a column specifies the value on the line specified by the column in the set of lines specified by the row.

Tables 305 and 307 show the state of CAM 201 before and after an associative clear operation. In these tables, part 302 represents the state of CAMM 101 0 and part 303 the state of CAMM 101 1 making up CAM 201. In tables 305 and 307, each row corresponds to a logical register 219 and the numbered columns correspond to cells 107. The value at the intersection of a row and a numbered column is thus the value of that cell 107 specified by the column number in logical register 219 specified by the row number. Table 305 further contains lettered columns; the letters heading these columns specify lines in CAMMs 101 corresponding to registers 105 making up logical registers 219 in CAM 201 and lines in CAM 201 itself. The letter M 121 specifies internal match lines 121, the letter C 123 specifies internal clear lines 123, the letters MA specify external match line 125, and the letters CMA specify CAM match lines 215. As previously explained, the state of a CAM match line 215 is the same as the state of the external match lines 125 connected to it. Again, the value at the intersection of a row and a lettered column is the state of the line specified by the letter corresponding to the register specified by the row.

Turning now to the operation illustrated in FIG. 3, the values of CAM mask lines 215 determine which values on CAM data input lines 213 are relevant to the match. In FIG. 3, CAM mask lines E(2) . . . E(7) all have the value 1; consequently, any value in cells 107 q(i,2) . . . q(i,7) produces a match when compared with the value on the corresponding line of CAM data input lines 213 D(2) . . . D7) and only the values in cells 107 q(i,0) . . . q(i,1) may fail to match when compared with the value of the corresponding data input line of data input lines 213 D(0) . . . D(1). The effect of the masking can be seen in column m for CAMM 1 303. Since all CAM mask lines 215 corresponding to cells 107 contained in CAMM 1 303 are active, the contents of these cells are indifferent and all internal match lines 121 in CAMM 1 303 are active. In CAMM 0 302, on the other hand, only CAM mask lines 215 corresponding to cells 107 q(i,2) . . . q(i,3) are active, and thus, the contents of cells 107 q(i,0) and q(i,l) are relevant to the match. As FIG. 3 shows, only in registers 105 (1), (4), and (5) do the contents of these cells match the values on the corresponding CAM data lines D(0) . . . D(1), and only internal match lines 121 corresponding to these registers 105 are active.

Further, since all internal match lines 121 m(i) in registers 105 r(i) making up a logical register 219 R(i) must be active in order for the CAM match line 217 corresponding to a logical register 219 R(i) to be active, only CAM match lines 217 for logical registers 219 (1), (4), and (5) are active. Finally, an internal clear line 123 c(i) in CAMM 0 302 or CAMM 1 303 is active only if CAM CLR 225 is active and external match line MA (i) 125 is active. Since the state of external match line MA(i) 125 is identical with the state of CAM match line 217 to which it is connected and only CAM match lines 219 for logical registers (1), (4), and (5) are active, only those internal clear lines 123 in CAMM 0 302 and CAMM 1 303 are active which correspond to registers 105 making np logical registers 219 1, 4, and 5. As shown in Table 307 of FIG. 3, showing the state of the cells 107 in CAM 201 after the clear operation, all cells 107 making up these logical registers 219 have been set to 0.

The associative clear operation illustrated in FIG. 3 may be used to simultaneously clear all data having a certain type code from a CAM 201 while leaving data with other type codes undisturbed. For example, the leftmost two bits of the data stored in CAM 201 of FIG. 3 might be such a type code. In the example of FIG. 3, CAM mask lines 215 mask all bits but those containing the type code, and the unmasked CAM data input lines 213 have the value 10, specifying a type code. As apparent in FIG. 3, when CAM CLR line 225 is active, all CAM 201 logical registers 219 containing data with the type code 10 are cleared.

4.3.4 CAMs with Different Properties Formed from CAMMS 101—FIG. 4

By varying the manner in which CAMMs 101 are connected together, CAMs with differing properties may be formed. FIG. 4 presents an example of such a CAM, a CAM with status registers. CAM 401 has two main parts: status registers 415 and data registers 417. Data registers 417 contain data; each register in status registers 415 is associated with a data register 417 and contains status information about that data register 417. Status information might include a bit indicating that the contents of the associated data register 417 are valid or one indicating that the associated data register 417 is being loaded. The association of registers in status registers 415 with registers in data registers 417 is accomplished by connecting all CAMMs 101 in CAM 401 to common CAM address lines 404, whereby a single address refers either to a register in status registers 415 or the register in data registers 417 associated with it. The division of CAM 401 into two sets of registers is accomplished by connecting CAMMs 101 making up data registers 417 to one set 403 of CAM input, output, masking, control, and match lines and CAMM 101 making up status registers 415 to another set 405, thus making it possible to perform read, write, match, and clear operations independently on status registers 415 and data registers 417.

4.4 Implementation of a CAMM 101

The discussion now turns to an exemplary implementation of a CAMM 101. The exemplary implementation is presented merely for purposes of illustration; other implementations are possible which are capable of performing the same operations as the exemplary implementation and are thus equivalent to it. The exemplary implementation discussed herein uses TTL gate array technology. In this technology, all logic functions must be expressed by means of NAND gates and inverters. Because of the complexities introduced into the implementation by this constraint, it is advantageous to first discuss FIGS. 5 and 5A, which together present a simplified logic diagram for a single register of a CAMM 101. Thereupon, the discussion will turn to the exemplary implementation of CAMM 101 itself.

4.4.1 Simplified Logic Diagram for a Single Register of a CAMM 101—FIG. 5

The logic diagram of FIGS. 5 and 5A employs AND gates, OR gates, and RS flip-flops, that is, flip-flops having an S input whose activation sets the flip-flop to 1, an R input whose activation sets the flip-flop to 0, a y output which has the value to which the flip-flop was last set, and a $\overline{y}$ output whose value is the complement of that of the y output. FIGS. 5 and 5A represent a single register 567 (i), outlined in dotted lines, and additional elements showing register 567 (i)'s relationship to the remainder of CAMM 101 to which it belongs. Register 567 (i) is functionally equivalent to register 105 r(i) of FIG. 1. Register 567 (i) is capable of storing four bits and consequently is made up of four cells 565 (i,0) . . . (i,3), equivalent to cells 107 q(i,0) . . . q(i,m) of FIG. 1.

4.4.1.1 Inputs and Outputs of Register 567 (i)

Inputs to register 567 (i) consist of: mask lines e(0) 501 through e(3) 507, corresponding to mask lines 127 e(0) . . . e(m) of FIG. 1; data input lines d(0) 509 and d(1) 571 through d(3) 575, corresponding to lines d(0) . . . d(m) of input data lines 117, data complement lines $\overline{d(0)}$ 511 and $\overline{d(1)}$ 577 through $\overline{d(3)}$ 581, carrying values which are the logical complement of the values on corresponding data input lines d(0) 509 and d(1) 571 through d(3) 575; OE line 508, corresponding to OE line 131, WE line 510, corresponding to WE line 133, internal clear line c(i) 523 corresponding to clear line c(i) of internal clear lines 123, and internal address line a(i) 513 corresponding to line a(i) of internal address lines 115.

Register 567 (i)'s outputs include register data output lines y(i,0) 539 through y(i,3) 551 and an external match line corresponding to line MA(i) of external match lines MA 125 in FIG. 1. As previously mentioned, external match lines MA 125 are bi-directional and may be connected to other external match lines MA 125. When so connected, an external match line MA 125 is active only if all other external match lines MA 125 connected to it are active. In FIG. 5, the bidirectional nature of the external match line and its relationship to corresponding match lines of other CAMMs 101 is expressed by representing the external match line for register 567(i) as two lines, MA(i)out 556 and MA(i)in 559. MA(i)out 556 is a continuation of internal match line m(i) 555; MA(i)in 559 is connected to CAM match line CMA(i) 564, corresponding to a line in CAM match lines 217 of FIG. 2. The relationship between lines MA(i)out 556, MA(i)in 559, and their equivalents in other CAMMs 101 is shown by means of wire AND gate 563 (in dotted lines). Inputs to gate 563 are lines MA(i)out for CAMMs 101 whose external match lines MA 125 are connected, its output is CAM match line CAM(i) 564, and MA(i)in 559's value is determined by the value of CAM match line CMA(i) 564.

4.4.1.2 Detailed Discussion of Cell 565 (i,0)

Since all cells 565 in register 567 (i) are identical, only cell 565 (i,0) is discussed in detail. Cell 565 (i,0)'s inputs are mask line e(0) 507, data input line d(0) 509, data complement line $\overline{d(0)}$ 511, internal address line a(i) 513, OE line 508, WE line 510, and internal clear line cl(i) 523. Cell 565 (i,0)'s outputs are cell match line m(i) 541 and cell output data line y(i,0) 539. The logical components of cell 565 (i,0) are: AND gate 515, receiving inputs from WE line 510, data line d(0) 509, and internal address line a(i) 513; AND gate 517, receiving inputs from WE line 510, data complement line $\overline{d(0)}$ 511, and internal address line a(i) 513; OR gate 525, receiving inputs from internal clear line c(i) 523 and AND gate 517; RS flip-flop RS(i,0) 529, receiving its S input from AND gate 515 and its R input from OR gate 525; AND gate 533, receiving inputs from data line d(0) 509 and the y output of RS flip-flop RS(i,0) 529; AND gate 534, receiving inputs from data complement line $\overline{d(0)}$ 511 and the $\overline{y}$ output of RS flip-flop RS(i,0); OR gate 540, receiving inputs from AND gates 533 and 534 and mask line e(0) 507; and AND gate 535, receiving inputs from internal address line a(i) 513 and the y output of RS flip-flop RS(i,0) 529.

4.4.1.3 Operations on Register 567 (i)

When read, write, match and associative clear operations are performed on the contents of register 567 (i), the components of cell 565 (i,0) interact as follows: In a write operation to register 567 (i) to which cell 565 (i,0) belongs, WE line 510 and internal address line a(i) 513 are both active. Consequently, the states of lines 519 and 521, carrying the outputs of AND gates 515 and 517 respectively, depend on whether data input line d(0) 509 is active. If it is, then data complement line $\overline{d(0)}$ 511 is inactive, line 519 is active, and line 521 is inactive. If data input line d(0) 509 is inactive, the reverse is true. Line 519 is connected to the S input of flip-flop RS(i,0) 529, and consequently, if line 519 is active, flip-flop RS(i,0) 529 is set to 1. Line 521 is connected to OR gate 523, which in turn is connected to the R input of flip-flop RS(i,0) 529. Therefore, if line 521 is active, flip-flop RS(i,0) 529 is reset to 0. Thus, after a write operation, the value at the y output of flip-flop RS(i,0) 529 is identical to the value represented on data input line d(0) 509 at the time of the write operation.

As FIG. 5 shows, internal address line a(i) 513 and WE line 510 are connected to other cells 565 in register 567 (i) in the same fashion as they are connected to cell 565 (i,0), and each of the other cells receives inputs from its equivalents to data input line d(0) 509 and data complement line $\overline{d(0)}$ 511 in the same fashion as cell 565 (i,0). Thus, at the end of a write operation, RS flip flops 529 (i,0 . . . 3) in register 567 (i) contain the values on data input lines d(0) 509 through d(3) 575.

In a read operation, internal address line a(i) 513 and OE line 508 are active. Internal address line a(i) 513 and line 531 from the y output of flip-flop RS(i,0) 529 serve as inputs to AND gate 535, whose output is cell data line 539 y(i,0). Thus, when internal address line a(i) 513 is active, the value of the y output of flip-flop RS(i,0) 529 determines the value of cell output data line 539. Cell output data line 539 is an input to OR gate 569, along with the equivalent lines from other registers 567. Thus, if cell output data line 539 is active, line 570, the output of OR gate 569, is active. Line 570 is one input to AND gate 571; the other input is OE line 508; consequently, when address line a(i) 513 and OE line 508 are active, cell data output line y(0) 573's value is determined by the value of the y output of flip-flop RS(i,0) 529. Since internal address line a(i) 513 and OE line 508 are connected in the same fashion in all cells 565 making up register (i) 567, the values at the y outputs of these registers' RS flip-flops (i,0 . . . 3) determine the values on data output lines y(0) 573 through y(3) 579. When a register is not being addressed, the outputs of the AND gates corresponding to AND gate 535 are inactive. Consequently, only the values in cells 565 (i,0 . . . 3) of the addressed register 567 (i) determine the values of data output lines y(0) 573 through y(3) 579.

In a match operation, the value at the y output of flip-flop RS(i,0) 529 is compared with the value on data input line d(0) 509 unless mask line e(0) 517 is active. When the operation is performed, the value at the y output of flip-flop RS(i,0) 529, carried on line 531, and the value on data input line d(0) 509 are both input to AND gate 533. At the same time the value of the $\bar{y}$ output of flip-flop RS(i,0) 529, carried on line 532, and the value on data complement line $\overline{d(0)}$ 511 are both input to AND gate 534. Consequently, if the value on data input line d(0) 509 matches the value at the y output, either line 537, the output of AND gate 533, or line 536, the output of AND gate 534, is active. Line 537 is active if data input line d(0) 509 and line 531, carrying the value of the y output, are both active, that is, if the data on data input line d(0) 509 and the data in flip-flop RS(i,0) both have the value 1, and line 536 is be active if data complement line $\overline{d(0)}$ 511 and line 532, carrying the value of the $\bar{y}$ output are both active, that is, if the data on data input line d(0) 509 and the data in flip-flop RS(i,0) 529 both have the value 0. Lines 536 and 537 are inputs to OR gate 540, and consequently, OR gate 540's output, line 541, is active if either line 536 or line 537 is active. If, on the other hand, the data on data input line d(0) 509 does not match the data in flip-flop RS(i,0) 529, neither AND gate 533 nor AND gate 534 has two active inputs, and output lines 537 and 536 are both inactive.

The third input to OR gate 540 is mask line e(0) 507. When data line d(0) 509 is being masked, mask line e(0) 507 is active and OR gate 540's output line 541 is active regardless of the values of lines 536 and 537, that is, regardless of whether data line d(0) 509 has the same value as flip-flop RS(i,0) 529. Line 541 and its equivalents from the other cells 565 in register 567 serve as inputs to AND gate 553, whose output is internal match line m(i) 555, corresponding to one of internal match lines 121. Consequently, internal match line m(i) 555 for a register (i) 567 is active only if all cell match lines for register (i) 567's cells are active.

The associative clear operation takes place when CLR line 512 is activated. If external match line MA(i)in 559 is active when CLR line 512 is activated, cell (i,0) 565 is cleared. CLR line 512 and external match line MA(i)in 559 are inputs to AND gate 514, which has internal clear line c(i) 523 as its output. Internal clear line c(i) 523 provides an input to OR gate 525, whose output is connected via line 527 to the R input of flip-flop RS(i,0) 529. Thus, when CLR line 512 and external match line MA(i)in 559 are active, internal clear line c(i) 523 is active, line 527 is active, and flip-flop RS(i,0) is set to 0. Since internal clear line c(i) 523 is connected as described above to all other cells 565 in register (i) 567, all cells 565 in register (i) 567 are cleared simultaneously with cell (i,0) 565. As previously mentioned, an external match line MA(i) 125 is active only if all other external match lines MA(i) 125 from other CAMMs 101 connected to it are active, and thus, if an associative clear operation may be performed on register (i) 567, it may be performed on corresponding registers 567 whose external match lines are connected to register (i) 567.

4.5 A TTL Gate Array Implementation of CAMM 101—FIGS. 6 and 6A through 6F

FIGS. 6 and 6A through 6F together contain a logic diagram for an exemplary TTL gate array implementation of an eight-register by four-bit CAMM 101. The form of the logic in this implementation is dictated by logical and electrical characteristics of the TTL gate array. The only logical devices which may be formed from the gate array are NAND gates and inverters. Further, each NAND gate must have three inputs and a given NAND gate or inverter can drive a maximum of four other NAND gates or inverters. In FIG. 6, only the cells of a single register are shown in detail; cells of remaining registers are represented as boxes with labelled inputs and outputs; the cells and registers so represented are, however, identical to the cells and register shown in detail.

4.5.1 Inputs and Outputs of the TTL Gate Array Implementation

CAMM 101 represented in FIGS. 6 and 6A through 6F, has the following inputs: on FIG. 6, data input lines D0 6167, D1 6171, D2 6175, and D3 6179, corresponding to data input lines 117 of FIG. 1; mask lines E0 6169, E1 6173, E2 6177, and E3 6181, corresponding to mask lines 127 and serving to mask the corresponding data input line when they are active; on FIG. 6A, external address lines A0 6026, A1 6028, and A2 6030, corresponding to external address lines 113; on FIG. 6D, $\overline{OE}$ line 6197, corresponding to OE 131; and on FIG. 6A, write enable line $\overline{WE}$ 6068, corresponding to WE 133, and $\overline{CLR}$ line 6081, corresponding to CLR 135. Lines $\overline{WE}$ 6068, $\overline{OE}$ 6197, and $\overline{CLR}$ 6081 are all normally active and are inactivated to specify a write, read, or clear operation respectively. Outputs from CAMM 101 represented in FIG. 6 are data output lines Y0 6147, Y1 6153, Y2 6157, and Y3 6161, on FIGS. 6D and 6F corresponding to data output lines 119 and bidirectional external match lines M0 6182 through M7 6196 on FIG. 6C corresponding to external match lines 125 in FIG. 1. As specified on FIG. 6C, external match lines M0 6182 through M7 6196 are connected to open collector outputs. When one such external match line M0 6182 through M7 6196 is connected to external match lines from other CAMMs 101 of the type disclosed in FIG. 6, the result is a wire AND: none of the connected external match lines will be active unless all of them are.

4.5.2 Functional Subdivisions of the TTL Implementation

CAMM 101 of FIG. 6A has the following functional subdivisions, outlined in dashed lines: on FIG. 6, data and mask input 6183, for receiving inputs from data input lines D0 6167 through D3 6179 and mask lines E0 6169 through E3 6181; on FIG. 6A, address decoder 6067, corresponding to address decoder 109, for receiving external address lines A0 6026 through A2 6028 and decoding addresses received on these lines; on FIGS. 6D and 6E, data outputs 6142 for outputting data received from registers 6176; on FIG. 6B, clear logic 6090, corresponding to clear logic 111, for clearing individual registers 6176; and on FIG. 6B, match logic 6189, for detecting matches. In addition, one register, register (0) 6187, on FIG. 6B, is outlined with dashed lines, and one cell of register (0) 6187, cell (0,0) 6185, is so outlined. Registers 6187 correspond to registers 105 of FIG. 1, and cells 6185 correspond to cells 107. The discussion deals first with each of these functional divisions and then with their interaction in the read, write, match, and associative clear operations.

4.5.2.1 Data and Mask Inputs 6183

Data and mask inputs 6183 on FIG. 6 include data input lines D0 6167 through D3 6179, mask lines E0 6169 through E3 6181 paired with the data lines, and associated logic. Since each data input line-mask line pair has the same logic, only that for data input line D0 6167 and mask line E0 6169 is discussed in detail. Beginning with D0 6167, the logic includes inverter 6001, with D0 6167 as its input and line 6003 as its output; inverter 6005, with line 6003 as its input and line 6011 as its output; inverter 6007, with mask line E0 6169 as its input and line 6009 as its output; NAND gate 6013, with inputs from lines 6003 and 6009 and an output to line 6017; inverters 6023, having line 6017 as their input and lines to cells 6185 as their outputs; NAND gate 6015, with inputs from lines 6009 and 6011 and an output to line 6019, and inverters 6020, with inputs from line 6019 and lines to cells 6185 as their outputs. In the following, only $\overline{\text{ID0A}}$ line 6025, the output of inverter 6021, and ID0A line 6024, the output of inverter 6022, are discussed in detail.

In the portion of data and mask inputs 6183 associated with data input line D0 6167 and mask line E0 6169, the inputs D0 6167 and E0 6169 and the outputs ID0A 6024 and $\overline{\text{ID0A}}$ 6025 have the following relationships: if data input line D0 6167 is not being masked, that is, if mask line E0 6169 is inactive, ID0A line 6024 is set to the value of data input line D0 6167 and $\overline{\text{ID0A}}$ line 6025 is set to the complement of that value; if data input line D0 6167 is being masked, that is, if E0 6169 is active, ID0A line 6024 and $\overline{\text{ID0A}}$ line 6025 are both inactive These relationships are achieved as follows: beginning with the case in which no masking is taking place, when mask line E0 6169 is inactive, line 6009 is active and the values of the outputs of NAND gates 6013 and 6015 depend on the values of lines 6003 and 6011 respectively. The values of lines 6003 and 6011 in turn depend on the value of data input line D0 6167. If data input line D0 6167 is active, line 6003 is inactive and line 6011 is active. Consequently, line 6019, the output of NAND gate 6015, is inactive, and its inversion, ID0A line 6024, is active, while line 6017, the output of NAND gate 6013, is active, and its inversion, $\overline{\text{ID0A}}$ line 6025, is inactive. If data input line D0 6167 is inactive, the reverse of the above is true. Thus, when mask line E0 6169 is inactive, ID0A line 6024's value is always identical with that of data input line D0 6167 and $\overline{\text{ID0A}}$ line 6025's value is always the complement of the value of data input line D0 6167. When data input line D0 6167 is being masked on the other hand, mask line E0 6169 is active, line 6009 is inactive, and consequently, NAND gates 6013 and 6015 have active outputs 6017 and 6019 and ID0A line 6024 and $\overline{\text{ID0A}}$ line 6025 are inactive regardless of the value of data input line D0 6167.

4.5.2.2 Address Decoder 6067—FIGS. 6A and 7

Turning now to address decoder 6067, on FIG. 6A, address decoder 6067's inputs are external address lines A0 6026, A1 6028, and A2 6030 and its outputs are internal address lines 6065, corresponding to internal address lines 115. Each line in internal address lines 6065 is associated with a register 6187. Lines in internal address lines 6065 are active unless register 6187 associated with a line is being addressed; in that case, the line associated with register 6187 being addressed is inactive. Thus, address decoder 6066 operates by activating all internal address lines 6065 but the one for the register specified by external address lines A0 6026 through A2 6030.

Address decoder 6066 consists of inverters 6027 through 6043 and NAND gates 6051 through 6054. Each address line A0 6026 through A2 6030 is input to an inverter and the output from that inverter is input to another inverter. Thus, for each address line A0 6026 through A2 6030, there is available from the first inverter a signal which is the complement of the signal on the corresponding external address line and from the second inverter a signal which is identical with that on the corresponding external address line. The signals obtained from the inverter outputs are then input to NAND gates 6051 through 6054. Each of these gates takes three inputs, one derived from address line A0 6026, one from address line A1 6028, and one from address line A2 6030. An input derived from a given address line is obtained from the output of either the first or second inverter following the address line. The input's value is therefore either identical with the value of the address line or the complement of that value. For example, NAND gate 6063 takes as its inputs line 6033, line 6035, and line 6049. Line 6033's value is the complement of the value of external address line A0 6026, line 6035's value is the complement of the value of external address line A1 6028, and line 6049's value is identical with that of external address line A2 6030. The inputs to NAND gates 6051 through 6064 are distributed among the gates in such fashion that a given combination of signals on external address lines A0 6026 through A2 6030 causes one of NAND gates 6051 through 6064 to have an inactive output and the remainder to have active outputs. For instance, NAND gate 6064 takes as its inputs line 6037, whose value is the complement of the value on external address line A2 6030, line 6035, whose value is the complement of the value on external address line A1 6028, and line 6033, whose value is the complement of the value on external address line 6026. NAND gate 6064's output 6067 is active unless line 6037, line 6035, and line 6028 are all simultaneously active, and the latter is true only if external address lines A0 6026 through A2 6030 are simultaneously inactive, that is, only if the values on external address lines A0 6026 through A2 6030 represent a binary 0. With all other NAND gates 6051 through 6063, when external address lines A0 6026 through A2 6030 are simultaneously inactive, at least one input line to each of NAND gates 6051 through 6063 is inactive, and consequently, all NAND gates 6051 through 6063 have active outputs.

The complete relationship between combinations of signals on external address lines A0 6026 through A2 6030 and outputs on internal address lines 6065 is illustrated in the truth table in FIG. 7. In that table, the table rows indicate the eight possible combinations of values on address lines A0 6026 through A2 6030 and the table columns indicate individual NAND gates 6051 through 6054 and their input lines. The table entries themselves show the output of the NAND gate specified by the entry's column for the values on address lines A0 6026 through A2 6030 specified by the entry's row.

4.5.2.3 Cell 6185 (0,0)

Turning now to cell 6185 (0,0), on FIG. 6B, cell 6185 (0,0) has the following inputs: data line ID0A 6024 and data complement line $\overline{\text{ID0A}}$ 6025 from data and mask inputs 6183, internal address line $\overline{\text{XA0}}$ 6067, from NAND gate 6064 of address decoder 6066, internal write enable line WE0 6078, whose value is derived from external write enable line WE 6068 by way of inverters 6069, 6071, and 6073 on FIG. 6A, and is therefore the complement of the value of external write enable line $\overline{\text{WE}}$ 6068, and internal clear line $\overline{\text{CLR0}}$ 6089, which corresponds to internal clear lines 123 except that internal clear line $\overline{\text{CLR0}}$ 6089 is inactive when an associative clear operation is taking place. Outputs from cell 6185 (0,0) are cell data line $\overline{\text{IY0}}$ 6113, whose value is the complement of the value stored in cell 6185 (0,0), and cell match lines 6117 and 6121, which are both active when either data input line D0 6167 is masked or the value contained in cell 6185 (0,0) matches the value on data input line D0 6167.

Cell 6185 (0,0) consists of: inverter 6091, receiving its input from internal address line $\overline{\text{XA0}}$ 6067; NAND gate 6095, receiving its inputs from inverter 6091, WE0 line 6078, and data line ID0A 6024; NAND gate 6097, receiving its inputs from inverter 6091, WE0 line 6078, and data complement line $\overline{\text{ID0A}}$ 6025; NAND gate 6103, receiving its inputs from NAND gate 6095 and NAND gate 6107; NAND gate 6107, receiving its inputs from NAND gate 6103, NAND gate 6097, and internal clear line $\overline{\text{CLR0}}$ 6089; NAND gate 6111, receiving its inputs from NAND gate 6105 and inverter 6091; NAND gate 6115, receiving its inputs from data line ID0A 6024 and NAND gate 6107, and NAND gate 6119, receiving its inputs from NAND gate 6103 and data complement line $\overline{\text{ID0A}}$ 6025. Finally, connection point 6122, connecting the outputs of NAND gates 6115 and 6119, is a wire AND; consequently, if either or both of lines 6117 and 6119 is inactive, line 6123 is inactive.

The components of cell 6185 (0,0) perform the same logical functions as the components of cell 565 (i,0) in FIG. 5. NAND gates 6095 and 6097 take inputs which are equivalent to those for AND gates 515 and 517 in FIG. 5 and provide outputs which are the complements of those of AND gates 515 and 517. Line 6099, the output of NAND gate 6095, is active unless line 6093, line ID0A 6024, and line WE0 6078 are all active. Line 6093 is the complement of internal address line $\overline{\text{X0A}}$ 6067, and consequently, is active only when register 6187 is being addressed, while line WE0 6078 is active only when a write operation is taking place. Therefore, line 6099 is inactive only when a write operation to register 6187 (0) is taking place and line ID0A 6024 is active. During a write operation to register 6187 (0), line 6099's value is thus the complement of the value of line ID0A 6024. NAND gate 6097's inputs are line 6093, line WE0 6078, and line $\overline{\text{ID0A}}$ 6025, and like NAND gate 6097, its output 6101 is inactive only when a write operation to register 6187 (0) is taking place and line $\overline{\text{ID0A}}$ 6025 is active. During a write operation, therefore, Line 6101's value is the complement of the value of line $\overline{\text{ID0A}}$ 6025 and also the complement of the value of line 6099. At other times, both line 6101 and line 6099 are active.

NAND gates 6103 and 6107 function as an RS flip-flop with R and S inputs which change the flip-flop's state when they become inactive. NAND gates 6103 and 6107 and NAND gates 6095 and 6097 together thus are logically equivalent to AND gates 515 and 517 and RS flip-flop 529 in FIG. 5. In the RS flip-flop formed by NAND gates 6103 and 6107, line 6105, the output of NAND gate 6103, is the Y output and line 6109, the output of NAND gate 6107 is the $\overline{\text{Y}}$ output. The set operation works as follows: line 6099 is the S input. As the output of NAND gate 6095, it is inactive only when input data line ID0A 6024, write enable line WE0 6078, and line 6093, the complement of internal address line $\overline{\text{XA0}}$ 6067, are active. When line 6099 is inactive, line 6105 becomes active, i.e., the Y output is set to 1. At the same time, line 6109 becomes inactive, i.e., the $\overline{\text{Y}}$ output is set to 0. This action takes place as follows: line 6105, line 6101 and $\overline{\text{CLR0}}$ line 6089 are inputs to NAND gate 6107. On a write operation, $\overline{\text{CLR0}}$ line 6089 is active. If line ID0A 6024 is active, lines 6105 and 6101 are also active; consequently, line 6109, the $\overline{\text{Y}}$ output, is inactive. If, on the other hand, line ID0A 6024 is inactive, line 6099 is active, lines 6105 and 6101 are inactive; and line 6109 is active. Thus, in this case, the Y output has the value 0 and the $\overline{\text{Y}}$ output the value 1.

$\overline{\text{CLR}}$ line 6089 acts as the R input to the flip-flop formed by NAND gates 6103 and 6107 only when no write operation is taking place. Under these circumstances, write enable line WE0 6078 is inactive, and consequently, lines 6099 and 6101 are active. When the flip-flop formed by NAND gates 6103 and 6107 contains the value 0, line 6105 is inactive and line 6109 is active regardless of the value of $\overline{\text{CLR}}$ line 6089. When the flip-flop formed by NAND gates 6103 and 6107 contains the value 1, line 6105 is active along with line 6101 and the value of $\overline{\text{CLR}}$ line 6089 determines the value of lines 6109 and 6105. If $\overline{\text{CLR}}$ line 6089 remains active, line 6109 remains inactive and line 6105 remains active; if $\overline{\text{CLR}}$ line 6089 becomes inactive, line 6109 becomes active and line 6105 becomes inactive, giving the flip-flop's Y output the value 0 and its $\overline{\text{Y}}$ output the value 1. Since either line 6101 or 6089 can reset the flip-flop formed by NAND gates 6103 and 6107, the connection of these lines to NAND gate 6107 is functionally equivalent to OR gate 525 in FIG. 5.

NAND gate 6111 in FIG. 6A inactivates cell data line $\overline{\text{IY0}}$ 6113 when both line 6093 and line 6105 are active. Line 6093 is the complement of internal address line $\overline{\text{XA0}}$ 6067, and is therefore active when register 6187 (0,0) is being addressed. Line 6105 is the Y output of the flip-flop formed by NAND gates 6103 and 6107, and consequently, when register 6187 (0,0) is being addressed, cell data line line $\overline{\text{IY0}}$ 6113's value is the complement of the value on line 6105. As shown on FIGS. 6E and 6F, cell data line $\overline{\text{IY0}}$ 6113 receives outputs from equivalent cells of all registers in the CAMM 101 described in FIG. 6 and then serves as an input to tri-state NAND gate 6145 on FIG. 6F. It thus effectively ORs these outputs and is equivalent to OR gate 569 in FIG. 5. Tri-state NAND gate 6145's output is data output line Y0 6147. This line has three states, active, inactive, and off. It is in the latter state when $\overline{\text{OE}}$ line 6197 is inactive and its complement, line 6149, is active; otherwise, input line 6143 is at VCC and is always active, and consequently, data output line Y0 6147's value is the complement of the value of cell data line $\overline{\text{IY0}}$ 6113, or the value of the Y output of the flip-flop formed by NAND gates 6103 and 6107. Together, NAND gates 6145 and 6111 output the value of the Y output of cell 6185 (0,0) when register 6187 (0) is addressed and output has been enabled; NAND gates 6145 and 6111 are thus logically equivalent to AND gates 535 and 571 of FIG. 5.

Turning again to FIG. 6B, NAND gates 6115, 6119, and the wire AND formed by connection 6122 between the outputs of NAND gates 6115, 6119, and internal match line 6123, finally, perform the match function for cell 6185 (0,0) and are thus equivalent to AND gates 533 and 534 and OR gate 540 in FIG. 5. NAND gate 6115 takes as its inputs line ID0A 6024 and line 6109 from the $\overline{Y}$ output of the flip-flop formed by NAND gates 6103 and 6107. NAND gate 6119 takes as its inputs line $\overline{ID0A}$ 6025 and line 6105 from the Y output of the flip-flop. If mask line E0 6169 is inactive, then, as described in the discussion of data and mask inputs 6183 above, the values on line ID0A 6024 and line $\overline{ID0A}$ 6025 are complementary. As also explained above, the values on lines 6105 and 6109 are always complementary. Consequently, when the value on line ID0A 6024 is the same as the value on line 6105, NAND gates 6115 and 6119 have complementary inputs and their outputs, lines 6117 and 6121, are both active. When the value on line ID0A is different from that on line 6105, one of NAND gates 6115 and 6119 has both inputs high, and lines 6117 and 6121 have have complementary values. When lines 6117 and 6121 are both active, the output from the AND formed by connection 6122 is active, indicating a match. When lines 6117 and 6121 have complementary values, the output from the AND formed by connection 6122 is inactive, indicating no match. Thus, when mask line E0 6169 is inactive, the output from the AND formed by connection 6122 is equivalent to the output of OR gate 540 when mask line e(0) 507 is inactive.

As mentioned in the discussion of data and mask inputs 6183, when mask line E0 6169 is active, both line ID0A 6024 and line $\overline{ID0A}$ 6025 are inactive. Since line ID0A 6024 serves as an input to NAND gate 6115, and line $\overline{ID0A}$ as an input to NAND gate 6119, the outputs of the NAND gates, lines 6117 and 6121 respectively, are both active regardless of the values on lines 6105 and 6109 and the output from the AND formed by connection 6122 is active, indicating a match. Thus, data and mask inputs 6183, NAND gates 6115 and 6119 and the AND formed by connection 6122 produce the same results when mask line E0 6169 is active as OR gate 540 in FIG. 5.

4.5.2.4 Register 6187 (0)

Cell 6185(0,0) and three equivalent cells 6185 form register 6187(0). All cells 6185 in register 6187 (0) take internal address line $\overline{XA0}$ 6067, and internal clear line $\overline{CLR0}$ 6089 as inputs and output to internal match line 6123. Because the cells in register 6187 share internal address line $\overline{XA0}$ 6067, internal clear line $\overline{CLR0}$ 6089, and internal match line 6123, they act as a single unit in read, write, match, and associative clear operations.

4.5.2.5 Data Outputs 6142

Data outputs 6142, on FIGS. 6D and 6F, outputs data contained in CAMM 101 registers 6187 to data output lines Y0 6147 through Y3 6161. Data to be output is received from lines $\overline{IY0}$ 6113, $\overline{IY1}$ 6125, $\overline{IY2}$ 6131, and $\overline{IY3}$ 6137. As previously explained, when a read operation is being performed, the values on these lines are the complements of the values in cells 6185 (i,0) through (i,3) of register 6187 (i) currently being addressed. Each of these lines is one input to one of NAND gates 6145 through 6159. NAND gates 6145 through 6159 are tri-state, that is, their outputs have three states, active, inactive, and off. The off state is controlled by $\overline{OE}$ line 6197. When $\overline{OE}$ line 6197 is active, line 6149 is inactive, and NAND gates 6145 through 6159 have no output; otherwise, their outputs are the NAND of their inputs. The other input to each of NAND gates 6155 through 6159 is line 6143, which is always active. Consequently, when $\overline{OE}$ line 6197 is inactive, the outputs of NAND gates 6145 through 6159 are the complements of the values on lines 6113, 6125, 6131, and 6137, that is, identical with the values contained in cells 6185 (i,0) through (i,3) in register 6187 (i).

4.5.2.6 Match Logic 6189

Match logic 6189 for register 6187 (0), on FIG. 6C, consists of internal match line 6123, inverter 6125, NAND gate 6129, and external match line M0 6182. The match logic for the other registers 6187 is identical, and consequently, only that for register 6187(0) is explained in detail.

Internal match line 6123 connects the output of wire AND 6122 with the outputs of equivalent wire ANDs in the other cells 6185 of register 6187 (0) and thereby forms another wire AND taking the output of wire AND 6122 and the outputs of its equivalents as inputs. Thus, internal match line 6123 is active only if the outputs of wire AND 6122 and its equivalents are all active, that is, only if each cell 6185 in register 6187 (0) indicates a match. Internal match line 6123 thus performs the function of AND gate 553 of FIG. 5.

Internal match line 6123 then serves as an input to inverter 6125, whose output, line 6126, is an input to NAND gate 6129. The other input to NAND gate 6129, line 6143, is at Vcc and therefore always active. In consequence, NAND gate 6129's output is inactive unless line 6126 is inactive, that is, unless internal match line 6123 is active. As indicated on FIG. 6A, external match line M0 6182 is an open collector output; hence, it acts as the output of a wire AND connecting the outputs of the equivalents of NAND gate 6129 in all CAMM registers 6187 whose equivalents to external match line M0 6182 are connected to external match line M0 6182, and if any of these external match lines are inactive, external match line M0 6182 is inactive.

4.5.2.7 Clear Logic 6090

Clear logic 6090 on FIGS. 6A and 6B activates internal clear line $\overline{CLR0}$ 6089 and its equivalents in other registers 6187. Inputs to clear logic 6090 are $\overline{CLR}$ line 6081, which is active except when an associative clear operation is being performed, and external match lines M0 6182 through M7 6196. Clear logic 6090 includes inverter 6083 and inverters 6084. Inverters in inverters 6084 are all identical to inverter 6088, and consequently, only that inverter is described in detail. Inverter 6088 has a control input, entering at the side of inverter 6088, as well as an input for the signal being inverted. As long as the control input is inactive, inverter 6088's output is active; when the control input is active, inverter 6088's output is the complement of the value of the signal being inverted. Inverter 6088 thus behaves like a NAND gate in that inverter 6088's output is inactive only if the control input and the input signal are both active. The control input for inverter 6088 is line 6095, which is the output of inverter 6083 and the signal input is external match line M0 6182. Line 6095's value is thus the complement of the value of $\overline{CLR}$ line 6081, and internal clear line $\overline{CLR0}$ 6089 is inactive, clearing register 6187(0), only if $\overline{CLR}$ line 6081 is inactive when external match line M0 6182 is active. Taken together, therefore, inverter 6083 and inverter 6088 are equivalent to AND gate 514 of FIG. 5.

4.5.3 Operations in the TTL Gate Array Implementation

Operations in the TTL gate array implementation are analogous to those discussed in reference to FIG. 5. On a write operation to register 6187 (0), on FIG. 6B, $\overline{WE}$ line 6068 is inactive and address lines A0 6026 through A3 6030 specify register 6187(0). Consequently, in each cell 6185 of the register, WE0 line 6078 is active, internal address line $\overline{XA0}$ 6097 is inactive, the line corresponding to line ID0A 6024 in cell 6185 (0,0) has the value of the line corresponding to data input line D0 6167, and the line corresponding to line $\overline{ID0A}$ 6025 has that value's complement. As explained in the discussion of cell 6185 (0,0), when WE0 line 6078 is active and internal address line $\overline{XA0}$ 6097 is inactive, the RS flip-flop contained in each cell 6185 is set to the value on the data input line of data input lines D0 6167 through D3 6179 corresponding to that cell 6185.

In a read operation on register 6187 (0), output enable line $\overline{OE}$ 6197 is inactivated and external address lines 6026 through 6030 specify register 6187 (0), deactivating internal address line $\overline{XA0}$ 6067. As explained in the discussion of cell 6185 (0,0), when internal address line $\overline{XA0}$ 6067 is inactive, line IY0 6113 and its equivalents in the other cells 6185 making up register 6187(0) have values which are the complement of the value at the Y output of cell 6185's flip-flop. The discussion of data outputs 6142 further showed that when output enable line $\overline{OE}$ 6197 is inactivated, the complements of the values of line 6113 and its equivalents in the other cells 6185 making up register 6187 (0) are output at data outputs Y0 6147 through Y(3) 6161. Since the values output at data outputs Y(0) 6147 through Y(3) 6161 are the complements of the values on line 6113 and its equivalents, they are identical with the values at the Y outputs of cells 6185 making up register 6187(0).

Turning now to a match operation, as previously explained with regard to cell 6185 (0,0), whenever a value on a data line D0 6167 through D3 6179 matches the value of its corresponding cell 6185 or whenever mask line E0 6169 through E3 6181 is active, the output of the connection in cell 6185 corresponding to connection 6122 in cell 6185 (0,0) is active. All of the connections corresponding to connection 6122 in cells 6185 belonging to a register 6187 (i) are connected by the line in register 6187 (i) corresponding to internal match line 6123 of register 6187 (0). As explained in the discussion of match logic 6189, internal match line 6123 and its equivalents function as wire ANDs taking the outputs from connection 6122 and its equivalents as inputs. The equivalent of internal match line 6123 for a register 6187 (i) is therefore active only if all outputs from connections equivalent to connection 6122 are active. If the equivalent of internal match line 6123 for a register 6187 (i) is active, then, as explained in the discussion of match logic 6189, external match line M0 6182 through M7 6196 corresponding to register 6187 (i) is active unless external match line M0 6182 through 6196 corresponding to register 6187 (i) is connected to external match lines M0 6182 through 6196 belonging to other CAMMs 101 and one of these external match lines M0 6182 through 6196 is inactive.

An associative clear operation, finally, is executed for a register 6187 (i) when external match line M0 6182 through M7 6196 corresponding to register 6187 (i) is active and $\overline{CLR}$ line 6081 is inactivated. As explained in the discussion of clear logic 6090, under these circumstances, the equivalent of line $\overline{CLR0}$ 6089 is inactive, and as explained in the discussion of cell 6185 (0,0), when this is the case, all cells 6185 belonging to register 6187 (i) are simultaneously set to 0.

Embodiments of the present invention may have specific forms other than those presented in FIGS. 1 through 7. The functions of the present invention may be performed by arrangements of logical devices other than those presented herein and different techniques may be used to implement the present invention. For example, the present invention may be implemented, using discrete devices, on a chip containing a single CAMM 101, or on a chip containing a plurality of CAMMs 201, and the devices on the chips may be formed using various technologies. Similarly, the number of bits in a register and the number of registers in a CAMM 101 may vary from implementation to implementation.

The invention may be embodied in yet other specific forms without departing from the spirit or essential characteristics thereof. Thus, the present embodiments are to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

What is claimed is:

1. A content-addressable memory module comprising:

(1) a plurality of register means, each register means of said plurality of register means containing one stored item of data;

(2) means for receiving a pattern item of data;

(3) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means, being responsive to said stored item of data contained in said one said register means and to said pattern item of data, and providing a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data; and

(4) a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module and receiving said match signal from an external source, each one of said bidirectional match signalling means being associated with one of said register means and responsive to said match signal from said match detection means associated with said associated register means and to said match signal from said external source, and acting to provide said match signal only when simultaneously receiving said match signal from said associated match detection means and from said external source.

2. In the content-addressable memory module of claim 1, and wherein

said bidirectional match signalling means is a match line connected to said associated match detection means for providing and receiving a match state and a no-match state;

said match signal is said match state; and said match line is connected to an open-collector driver circuit in said associated match detection means and said open-collector driver circuit places said connected match line in said no match state unless said stored item of data in said register means associated with said match detection means matches said pattern item of data.

3. In the content-addressable memory module of claim 2, and wherein:

said match state is a high voltage and

said no match state is a low voltage.

4. In the content-addressable memory module of claim 3, and wherein:

a pattern sequence of bits in said pattern item of data corresponds to a certain sequence of bits in each one of said stored items of data, said match detection means is responsive to said pattern sequence of bits and to said certain sequence of bits, and said stored item of data matches said pattern item of data when said bits in said certain sequence match said bits in said pattern sequence.

5. In the content-addressable memory module of claim 4, and wherein:

said content-addressable memory module further includes means for receiving a masking item of data for specifying said pattern sequence of bits and

said match detection means is further connected to said masking item receiving means and is responsive to said masking item of data.

6. In the content-addressable memory module of claim 5, and wherein:

said masking item of data further specifies a non-pattern sequence of bits in said pattern item of data;

said stored items of data further contain a second certain sequence of bits corresponding to said non-pattern sequence of bits; and

one said stored item of data matches said pattern item of data when said first certain sequence of bits matches said pattern sequence of bits, regardless of the values of bits in said second certain sequence of bits.

7. A content-addressable memory module comprising:

(1) a plurality of register means, each register means of said plurality of register means containing one stored item of data;

(2) means for receiving a pattern item of data;

(3) means for receiving a clear signal specifying that certain ones of said plurality of register means are to be cleared, said certain ones being said register means containing said stored items of data matching said pattern item of data; and

(4) means for simultaneously clearing said certain ones of said register means, said simultaneous clearing means being connected to said plurality of register means, to said pattern receiving means, and to said clear signal receiving means and responding to said stored item of data, said pattern item of data, and said clear signal by simultaneously clearing said certain ones of said register means on receipt of said clear signal in said clear signal receiving means;

wherein:

a pattern sequence of bits in said pattern item of data corresponds to a certain sequence of bits in each one of said stored items of data and said stored item of data matches said pattern item of data when said bits in said certain sequence match said bits in said pattern sequence;

and wherein said simultaneous clearing means includes:

(a) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means and being responsive to said stored item of data contained in said one said register means and to said pattern item of data, and each one of said plurality of match detection means acting to provide a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data;

(b) means for providing a register clearing signal specifying any one of said register means in response to said clear signal and to said match signal; and

(c) a plurality of means for clearing said register means, each one of said register clearing means being associated with one of said register means and being responsive to said register clearing signal.

8. In the content-addressable memory module of claim 7, and wherein:

said content-addressable memory module further includes a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module, receiving said match signal from an external source, and providing said match signal to said register clearing signal providing means, each bidirectional match signalling means of said plurality of bidirectional match signalling means being associated with one register means of said plurality of register means and being connected to said match detection means associated with said associated register means and to said register clearing signal providing means, and each said bidirectional match signalling means providing said match signal to said register clearing signal providing means only when said bidirectional match signalling means is simultaneously receiving said match signal from said connected match detection means and from said external source.

9. In the content-addressable memory module of claim 8, and

wherein:

said bidirectional match signalling means is a match line connected to said match detection means and to said register clearing signal providing means;

said match line provides and receives a match state and a no-match state;

said match signal is said match state; and

each said match line is connected to an open-collector driver circuit in said associated match detection means and said open-collector driver circuit places said connected match line in said no match state unless said stored item of data in said register means associated with said match detection means matches said pattern item of data.

10. A content-addressable memory module comprising:

(1) a plurality of register means, each register means of said plurality of register means containing one stored item of data;

(2) means for receiving a pattern item of data;

(3) means for receiving a clear signal specifying that certain ones of said plurality of register means are to be cleared, said certain ones being said register means containing said stored items of data matching said pattern item of data; and

(4) means for simultaneously clearing said certain ones of said register means, said simultaneous clearing means being connected to said plurality of register means, to said pattern receiving means, and to said clear signal receiving means and responding to said stored item of data, said pattern item of data, and said clear signal by simultaneously clearing said certain ones of said register means on receipt of said clear signal in said clear signal receiving means;

wherein:

a pattern sequence of bits in said pattern item of data corresponds to a certain sequence of bits in each one of said stored items of data and said stored item of data matches said pattern item of data when said bits in said certain sequence match said bits in said pattern sequence.

and wherein:

said content-addressable memory further includes means for receiving a masking item of data for specifying said pattern sequence of bits and said simultaneous clearing means is further connected to said masking item receiving means and is responsive to said masking item of data;

and wherein:

said masking item of data further specifies a non-pattern sequence of bits in said pattern data item;

said stored items of data further contain a second certain sequence of bits corresponding to said non-pattern sequence of bits; and

one said said stored item of data matches said pattern data item when said first certain sequence of bits matches said pattern sequence of bits, regardless of the values of bits in said second certain sequence of bits;

and wherein:

said masking item of data specifies all said bits in said pattern item of data as said non-pattern sequence of bits, whereby all said stored items of data match said pattern item of data, all said register means in said plurality of register means are said certain ones of said plurality of register means, and said simultaneous clearing means simultaneously clears all said register means in said plurality of register means upon receipt of said clear signal in said clear signal receiving means.

11. In the content-addressable memory module of claim 10, and wherein:

said simultaneous clearing means further includes

(a) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means, being responsive to said stored item of data contained in said one said register means, to said pattern item of data, and to said mask item of data, and providing a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data,

(b) means for providing a register clearing signal to any one of said register means in response to said clear signal and to said match signal,

(c) a plurality of means for clearing said register means, each one of said register clearing means being associated with one of said register means and being responsive to said register clearing signal.

12. In the content-addressable memory module of claim 11, and wherein:

said content-addressable memory module further includes a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module, receiving said match signal from an external source, and providing said match signal to said register clearing signal providing means, each bidirectional match signalling means of said plurality of bidirectional match signalling means being associated with one register means of said plurality of register means and being connected to said match detection means associated with said associated register means and to said register clearing signal providing means, and each said bidirectional match signalling means providing said match signal to said register clearing signal providing means only when said bidirectional match signalling means is simultaneously receiving said match signal from said connected match detection means and from said external source.

13. In the content-addressable memory of claim 7, 10, or 1, and wherein said content-addressable memory module further comprises:

address receiving means connected to said plurality of register means for receiving an encoded address specifying an addressed register means of said plurality of register means from an external source, decoding said encoded address to generate an address signal for said addressed register means specified by said encoded address, and providing said address signal to said addressed register means, each register means of said plurality of register means being responsive to said address signal.

14. In the content-addressable memory module of claim 13 and wherein:

said address receiving means includes

(a) encoded address receiving means for receiving an encoded address specifying said addressed register means from said external source;

(b) decoding means connected to said encoded address receiving means and responsive to said encoded address for decoding said encoded address and generating said address signal for said addressed register means; and

(c) means connected to said decoding means and said plurality of register means for providing said address signal to said addressed register means.

15. In the content-addressable memory module of claim 7, 10, or 1, and wherein said content-addressable memory module further comprises:

data input means for receiving an input item of data from an external source;

means for receiving an address specifying an addressed register means of said plurality of register means from an external source and providing an address signal for said addressed register means;

data output means for outputting one said stored item of data from said content-addressable memory module;

means for receiving an output enable signal from an external source;

means for receiving a write enable signal from an external source;

data writing means connected to said plurality of register means, said address receiving means, said data input means, and said write enable signal receiving means for setting said stored item of data in said addressed register means to the value of said input data item in response to said input item of data, said address signal, and said write enable signal;

data reading means connected to said plurality of register means, said address receiving means, said data output means, and said output enable signal receiving means for providing said stored item of data in said addressed register means to said data output means in response to said address signal and said output enable signal.

16. A content-addressable memory comprising:

(1) a plurality of content-addressable memory modules, each content-addressable memory module of said plurality of content-addressable memory modules including

(a) a plurality of register means, each register means of said plurality of register means containing one stored item of data;

(b) means for receiving a pattern item of data;

(c) means for receiving a clear signal specifying that certain ones of said plurality of register means are to be cleared, said certain ones being said register means containing said stored items of data matching said pattern item of data; and

(d) means for simultaneously clearing said certain ones of said register means, said simultaneous clearing means being connected to said plurality of register means, to said pattern item receiving means, and to said clear signal receiving means and responding to said stored item of data, said pattern item of data, and said clear signal by simultaneously clearing said certain ones of said register means on receipt of said clear signal in said clear signal receiving means; and

(2) memory clear signal providing means connected to said clear signal receiving means in each one of said plurality of memory modules for simultaneously providing said clear signal to all said content-addressable memory modules in said plurality of content-addressable memory modules;

wherein:

said simultaneous clearing means includes

(i) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means and being responsive to said stored item of data contained in said one said register means and to said pattern item of data, and each one of said plurality of match detection means acting to provide a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data;

(ii) means for providing a register clearing signal to any one of said register means in response to said clear signal and to said match signal; and

(iii) a plurality of means for clearing said register means, each one of said register clearing means being associated with one of said register means and being responsive to said register clearing signal;

said content-addressable memory module further includes a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module, receiving said match signal from an external source, and providing said match signal to said register clearing signal providing means, each bidirectional match signalling means of said plurality of bidirectional match signalling means being associated with one register means of said plurality of register means and being connected to said match detection means associated with said associated register means and to said register clearing signal providing means, and each said bidirectional match signalling means providing said match signal to said register clearing signal providing means only when said bidirectional match signalling means in simultaneously receiving said match signal from said connected match detection means and from said external source; and

said content-addressable memory further includes a plurality of memory match signalling means for receiving said match signal from said bidirectional match signalling means and serving as said external source for providing said match signal to said bidirectional match signalling means, each one of said memory match signalling means corresponding to one of said bidirectional match signalling means, being connected to said corresponding said match signalling means in each of said content-addressable memory modules, and providing said match signal to said connected bidirectional match signalling means only when all of said connected bidirectional match signalling means are providing said match signal;

whereby said content-addressable memory responds to said clear signal provided by said memory clear signal providing means by clearing said register means only when said register means contain said stored items of data matching said pattern item of data and said register means are associated with said bidirectional match signalling means which are receiving said match signal from said memory match signalling means.

17. In the content-addressable memory of claim 16, and wherein said content-addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional said memory clear signal providing means connected to said clear signal receiving means in each one of said additional plurality of memory modules,

and wherein each one of said plurality of memory match signalling means is further connected to said corresponding bidirectional match signalling means in each content-addressable memory module of said additional plurality of content-addressable memory modules.

18. In the content-addressable memory of claim 16, and wherein said content-addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional plurality of memory match signalling means, each one of said additional plurality of memory match signalling means being connected to said corresponding said match signalling means in each of said content-addressable memory modules of said additional plurality of content-addressable memory modules; and

wherein said memory clear signal providing means is further connected to said clear signal receiving means in each content-addressable memory module of said additional plurality of memory modules.

19. In the content-addressable memory of claim 16, and wherein:

a pattern sequence of bits in said pattern item of data corresponds to a certain sequence of bits in each one of said stored items of data and said stored item of data matches said pattern item of data when said bits in said certain sequence match said bits in said pattern sequence.

20. In the content-addressable memory of claim 19, and wherein:

said simultaneous clearing means includes

(i) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means and being responsive to said stored item of data contained in said one said register means and to said pattern sequence of bits, and each one of said plurality of match detection means acting to provide a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data,

(ii) means for providing a register clearing signal to any one of said register means in response to said clear signal and to said match signal, and

(iii) a plurality of means for clearing said register means, each one of said register clearing means being associated with one of said register means and being responsive to said register clearing signal;

said content-addressable memory module further includes a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module, receiving said match signal from an external source, and providing said match signal to said register clearing signal providing means, each bidirectional match signalling means of said plurality of bidirectional match signalling means being associated with one register means of said plurality of register means and being connected to said match detection means associated with said associated register means and to said register clearing signal providing means, and each said bidirectional match signalling means providing said match signal to said register clearing signal providing means only when said bidirectional match signalling means is simultaneously receiving said match signal from said connected match detection means and from said external source; and

said content-addressable memory further includes a plurality of memory match signalling means for receiving said match signal from said bidirectional match signalling means and serving as said external source for providing said match signal to said bidirectional match signalling means, each one of said memory match signalling means corresponding to one of said bidirectional match signalling means, being connected to said corresponding said match signalling means in each of said content-addressable memory modules, and providing said match signal to said connected bidirectional match signalling means only when all of said connected bidirectional match signalling means are providing said match signal,

whereby said content-addressable memory responds to said clear signal provided by said memory clear signal providing means by clearing said register means only when said register means contain said stored items of data matching said pattern item of data and said register means are associated with said bidirectional match signalling means which are receiving said match signal from said memory match signalling means.

21. In the content-addressable memory of claim 19, and wherein:

said content-addressable memory module further includes means for receiving a masking item of data for specifying said pattern sequence of bits and

said simultaneous clearing means is further connected to said masking item receiving means and is responsive to said masking item of data.

22. In the content-addressable memory of claim 21, and wherein:

said masking item of data further specifies a non-pattern sequence of bits in said pattern data item;

said stored items of data further contain a second certain sequence of bits corresponding to said non-pattern sequence of bits; and

one said stored item of data matches said pattern item of data when said first certain sequence of bits matches said pattern sequence of bits, regardless of the values of bits in said second certain sequence of bits.

23. In the content-addressable memory of claim 22, and wherein:

said masking item of data specifies all said bits in said pattern item of data as said non-pattern sequence of bits,

whereby all said stored items of data match said pattern item of data, all said register means in said plurality of register means are said certain ones of said plurality of register means, and said simultaneous clearing means simultaneously clears all said register means in said plurality of register means upon receipt of said clear signal in said clear signal receiving means.

24. In the content-addressable memory module of claim 21, and wherein:

said simultaneous clearing means further includes

(i) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means, being responsive to said stored item of data contained in said one said register means, to said pattern item of data, and to said mask item of data, and providing a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data,

(ii) means for providing a register clearing signal to any one of said register means in response to said clear signal and to said match signal, and

(iii) a plurality of means for clearing said register means, each one of said register clearing means being associated with one of said register means and being responsive to said register clearing signal;

said content-addressable memory module further includes a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module, receiving said match signal from an external source, and providing said match signal to said register clearing signal providing means, each bidirectional match signalling means of said plurality of bidirectional match signalling means being associated with one register means of said plurality of register means and being connected to said match detection means associated with said associated register means and to said register clearing signal providing means, and each said bidirectional match signalling means providing said match signal to said register clearing signal providing means only when said bidirectional match signalling means is simultaneously receiving said match signal from said connected match detection means and from said external source; and

said content-addressable memory further includes a plurality of memory match signalling means for receiving said match signal from said bidirectional match signalling means and serving as said external source for providing said match signal to said bidirectional match signalling means, each one of said memory match signalling means corresponding to one of said bidirectional match signalling means, being connected to said corresponding said match signalling means in each of said content-addressable memory modules, and providing said match signal to said connected bidirectional match signalling means only when all of said connected bidirectional match signalling means are providing said match signal,

whereby said content-addressable memory responds to said clear signal provided by said memory clear signal providing means by clearing said register means only when said register means contain said stored items of data matching said pattern item of data and said register means are associated with said bidirectional match signalling means which are receiving said match signal from said memory match signalling means.

25. In the content-addressable memory of claim 24, and wherein said content-addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional said memory clear signal providing means connected to said clear signal receiving means in each one of said additional plurality of memory modules,

and wherein each one of said plurality of memory match signalling means is further connected to said corresponding bidirectional match signalling means in each content-addressable memory module of said additional plurality of content-addressable memory modules.

26. In the content-addressable memory of claim 24, and wherein said content addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional plurality of memory match signalling means, each one of said additional plurality of memory match signalling means being connected to said corresponding said match signalling means in each of said content-addressable memory modules of said additional plurality of content-addressable memory modules; and

wherein said memory clear signal providing means is further connected to said clear signal receiving means in each one of said additional plurality of memory modules.

27. A content-addressable memory comprising:

(1) a plurality of content-addressable memory modules, each one of said plurality of content-addressable memory modules including

(a) a plurality of register means, each register-means of said plurality of register means containing one stored item of data;

(b) means for receiving a pattern item of data;

(c) a plurality of means for detecting said register means containing said stored items of data matching said pattern item of data, each one of said plurality of match detection means being associated with one said register means, being responsive to said stored item of data contained in said one said register means and to said pattern item of data, and providing a match signal when said one said register means associated with said one match detection means contains said stored item of data matching said pattern item of data; and

(d) a plurality of bidirectional match signalling means for providing said match signal from said content-addressable memory module and receiving said match signal from an external source, each one of said bidirectional match signalling means being associated with one of said register means and responsive to said match signal from said match detection means associated with said associated register means and to said match signal from said external source, and acting to provide said match signal only when simultaneously receiving said match signal from said associated match detection means and from said external source; and

(2) a plurality of memory match signalling means for receiving said match signal from said bidirectional match signalling means and serving as said external source for providing said match signal to said bidirectional match signalling means, each one of said memory match signalling means corresponding to one of said bidirectional match signalling means, being connected to said corresponding said match signalling means in each of said content-addressable memory modules, and providing said match signal to said connected bidirectional match signalling means only when all of said connected bidirectional match signalling means are providing said match signal.

28. In the content-addressable memory of claim 27, and wherein said content-addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional plurality of said memory match signalling means, each one of said additional plurality of memory match signalling means being connected to said corresponding said match signalling means in each of said content-addressable memory modules of said additional plurality of content-addressable memory modules.

29. In the content-addressable memory of claim 27, and wherein:

said bidirectional match signalling means is a match line connected to said match detection means and clearing signal providing means;

said memory match signalling means is a memory match line connected to a corresponding said match line in each one of said content-addressable memory modules;

said match line and said memory match line provide and receive a match state and a no-match state;

said match signal is said match state; and

each said match line is connected to to an open-collector driver circuit in said associated match detection means and said open-collector driver circuit places said connected match line and said connected memory match line in said no match state unless said stored item of data in said register means associated with said match detection means matches said pattern item of data.

30. In the content-addressable memory of claim 29, and wherein:

said match state is a high voltage and said no match state is a low voltage.

31. In the content-addressable memory of claim 30, and wherein:

a pattern sequence of bits in said pattern item of data corresponds to a certain sequence of bits in each one of said stored items of data, said match detection means is responsive to said pattern sequence of bits and to said certain sequence of bits, and said stored item of data matches said pattern item of data when said bits in said certain sequence match said bits in said pattern sequence.

32. In the content-addressable memory module of claim 31, and wherein:

said content-addressable memory module further includes means for receiving a masking item of data for specifying said pattern sequence of bits and

said match detection means is further connected to said masking item receiving means and is responsive to said masking item of data.

33. In the content-addressable memory module of claim 32, and wherein:

said masking item of data further specifies a non-pattern sequence of bits in said pattern data item;

said stored items of data further contain a second certain sequence of bits corresponding to said non-pattern sequence of bits; and

one said stored item of data matches said pattern item of data when said first certain sequence of bits matches said pattern sequence of bits, regardless of the values of bits in said second certain sequence of bits.

34. In the content-addressable module of claim 33, and wherein:

said masking item of data specifies all said bits in said pattern item of data as said non-pattern sequence of bits,

whereby said memory match line is in said match state when first certain memory modules of said plurality of memory modules receive said masking items of data specifying all said bits in said pattern items of data received by said first certain memory modules as said non-pattern bits, second certain memory modules of said plurality of memory modules receive said masking items not specifying all said bits in said pattern item of data as said bits, and said stored items of data in said register means associated with said memory match lines in said second certain memory modules match said pattern items received by said second certain memory items.

35. In the content-addressable memory of claim 34, and wherein said content-addressable memory further comprises:

an additional plurality of said content-addressable memory modules; and

an additional plurality of said memory match signalling means, each one of said additional plurality of memory match signalling means being connected to said corresponding said match signalling means in each of said content-addressable memory modules of said additional plurality of content-addressable memory modules.

36. In the content-addressable memory of claim 16, 19, 21, or 27, and wherein:

said content-addressable memory module further includes

address receiving means connected to said plurality of register means for receiving an encoded address specifying an addressed register means of said plurality of register means from an external source, decoding said encoded address to generate an address signal for said addressed register means specified by said encoded address, and providing said address signal to said addressed register means, each register means of said plurality of register means being responsive to said address signal; and

(2) memory register address providing means connected to each said address receiving means in said plurality of memory modules for simultaneously providing said encoded address to said address receiving means in each one of said plurality of memory modules,

whereby said encoded address provided by said memory register address providing means specifies a memory register made up of said addressed register means in each one of said plurality of memory modules.

37. In the content-addressable memory of claim 17, 18, 25, 26, 28, or 35, and wherein:

each content-addressable memory module of said plurality of content-addressable memory modules and of said additional plurality of content-addressable memory modules further includes address receiving means, connected to said plurality of register means for receiving an encoded address specifying an addressed register means of said plurality of register means from an external source, decoding said encoded address to generate an address signal for said addressed register means specified by said encoded address, and providing said address signal to said addressed register means, each register means of said plurality of register means being responsive to said address signal; and

said content-addressable memory further includes memory register address providing means connected to each said address receiving means in said plurality of memory modules and to said address receiving means in said additional plurality of memory modules for simultaneously providing said encoded address to said address receiving means in each one of said plurality of memory modules and in each one of said additional plurality of memory modules.

38. In the content-addressable memory of claim 17, 18, 25, 26, 28, or 35, and wherein:

each content-addressable memory module of said plurality of content-addressable memory modules and of said additional plurality of content-addressable memory modules further includes address receiving means connected to said plurality of register means for receiving an encoded address specifying an addressed register means of said plurality of register means from an external source, decoding said encoded address to generate an address signal for said addressed register means specified by said encoded address, and providing said address signal to said addressed register means, each register means of said plurality of register means being responsive to said address signal; and

said content-addressable memory further includes

memory register address providing means connected to each said address receiving means in said plurality of memory modules for simultaneously providing said encoded address to said address receiving means in each one of said plurality of memory modules; and

additional memory register address providing means connected to each said address receiving means in said additional plurality of memory modules for simultaneously providing an additional said encoded address to said address receiving means in each one of said additional plurality of memory modules.

* * * * *

United States Patent [19]

Okamoto et al.

[11] **Patent Number: 4,910,668**

[45] **Date of Patent: Mar. 20, 1990**

[54] **ADDRESS CONVERSION APPARATUS**

[75] Inventors: **Tadashi Okamoto**, Hirakata; **Hiroshi Kadota**, Toyonaka; **Masaitsu Nakajima**, Hirakata, all of Japan

[73] Assignee: **Matsushita Electric Industrial Co., Ltd.**, Osaka, Japan

[21] Appl. No.: **100,561**

[22] Filed: **Sep. 24, 1987**

[30] **Foreign Application Priority Data**

Sep. 25, 1986 [JP] Japan 61-226697

[51] **Int. Cl.⁴** **G06F 12/10**

[52] **U.S. Cl.** **364/200**

[58] **Field of Search** 364/200, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

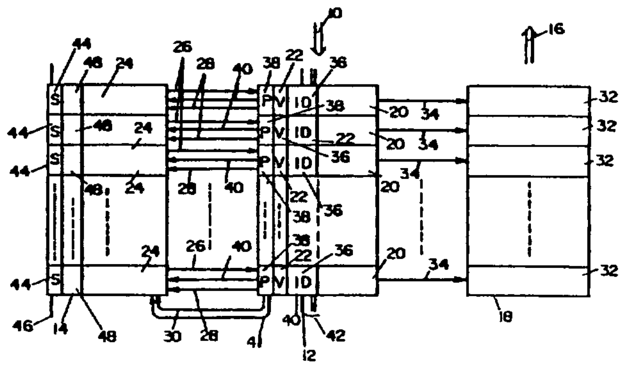
4,296,475	10/1981	Nederlof et al.	364/900
4,433,389	2/1984	York et al.	364/900
4,538,241	8/1985	Levin et al.	364/900
4,607,331	8/1986	Goodrich, Jr. et al.	364/200
4,785,398	11/1988	Joyce et al.	364/200

Primary Examiner—Gareth D. Shaw
Assistant Examiner—Debra A. Chun
Attorney, Agent, or Firm—Wenderoth, Lind & Ponack

[57] **ABSTRACT**

An address conversion apparatus includes a content addressable memory for storing a plurality of logical addresses, and a random access memory for storing a plurality of physical addresses corresponding to the logical addresses. When an input logical address is received, a search is conducted to find the same logical address stored in the memory. When the same logical address is found, the content addressable memory causes the random access memory to output a corresponding physical address. The content addressable memory includes a plurality of logical address storage units. Each unit has a plurality of data bit cells for storing address data and a process identification number cell for storing a process identification number. Thereby, a plurality of logical addresses which correspond to different processes are stored in the single content addressable memory.

2 Claims, 8 Drawing Sheets



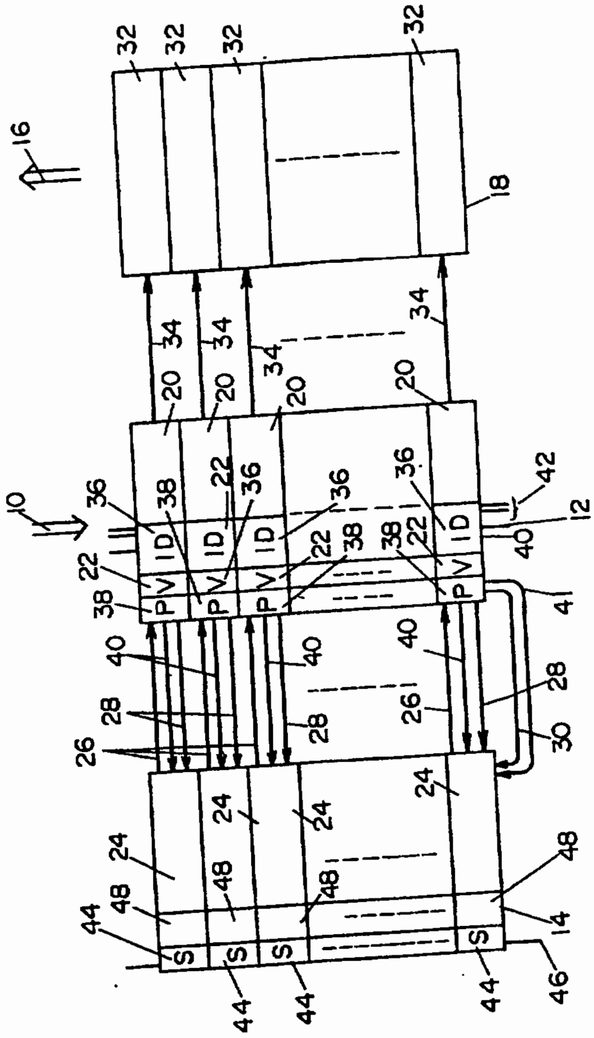


FIG. 1

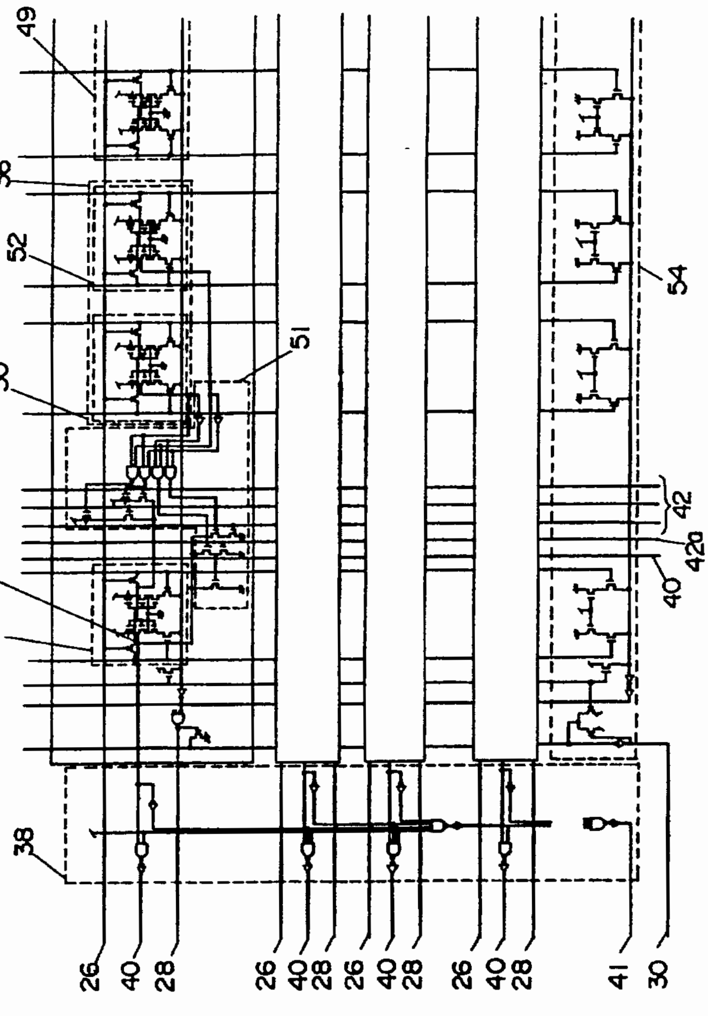

49
36
52
50
51
22a
22
38
54
42
42a
40
26
40
28
26
40
28
26
40
28
26
40
28
41
30

FIG. 2

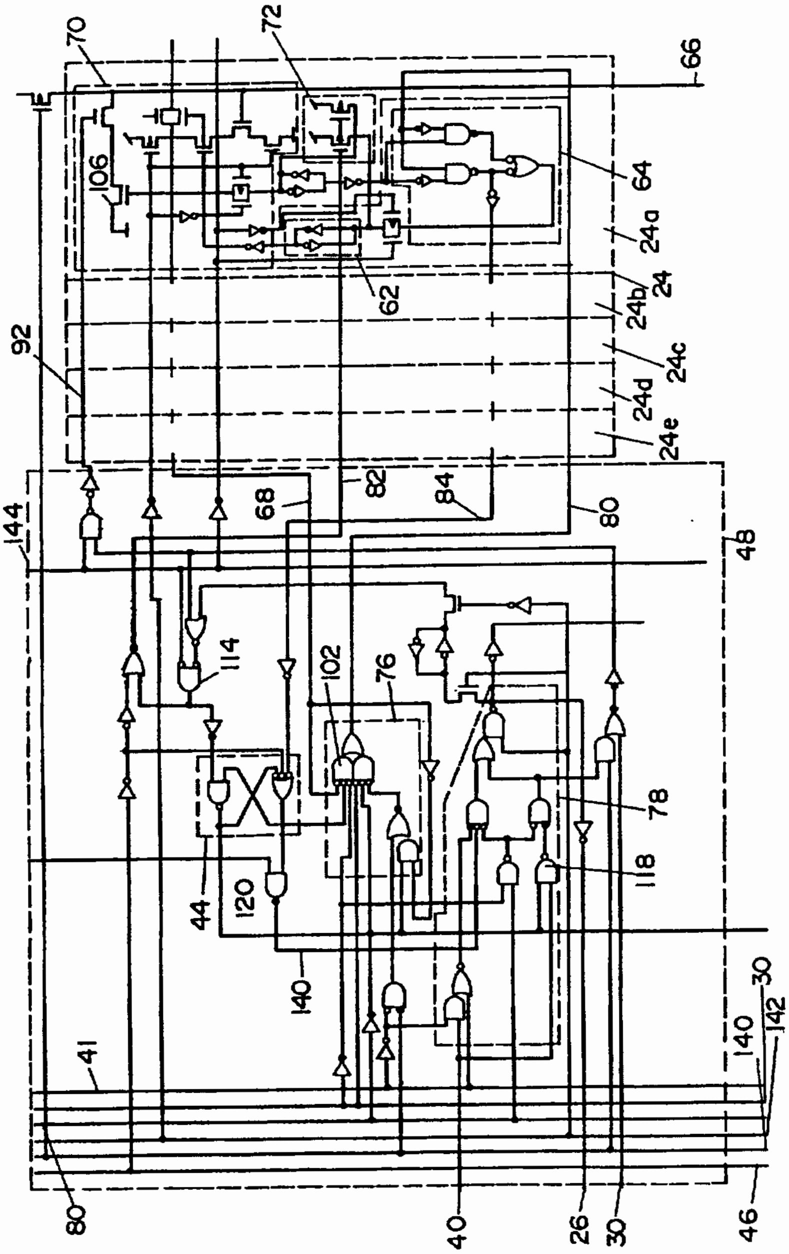

70
72
66
106
64
24a
62
24b
24
92
24c
24d
24e
144
68
82
84
80
48
114
102
76
78
118
44
120
140
41
30
142
80
40
26
30
46

FIG. 3

FIG. 4A

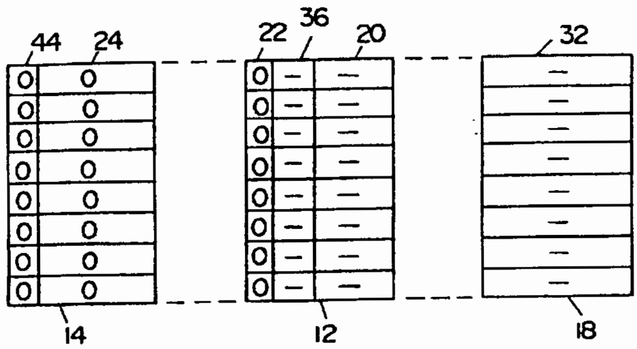


FIG. 4B

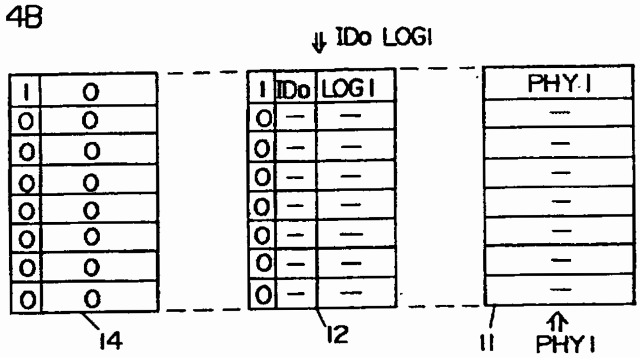


FIG. 4C

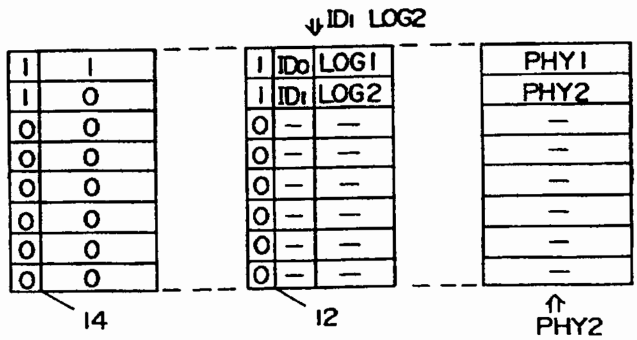


FIG. 4D

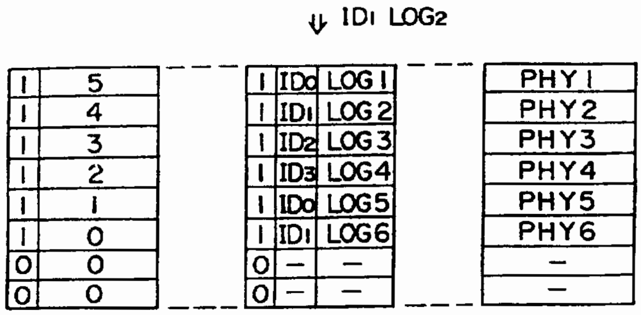


FIG. 4E

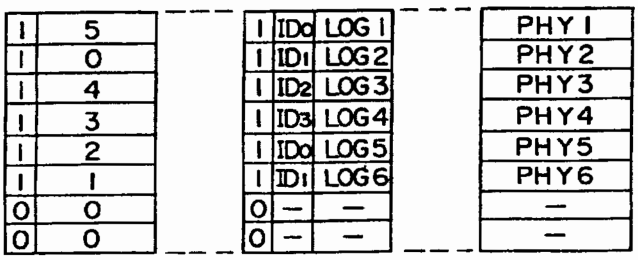


FIG. 4F

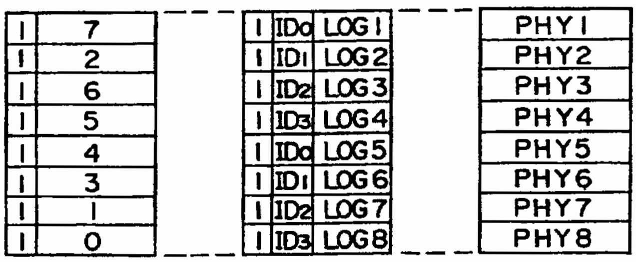


FIG. 4G

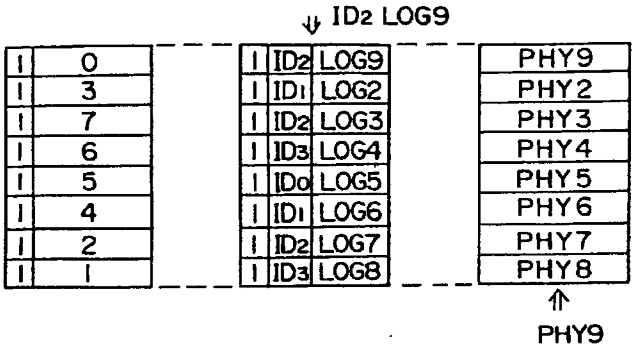


FIG. 4H

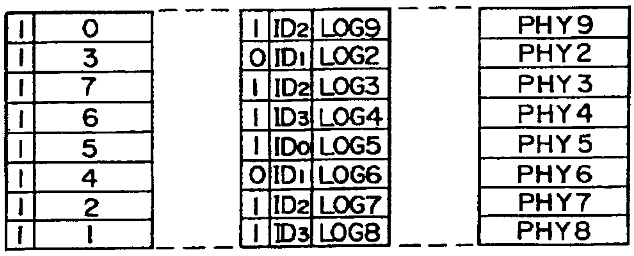


FIG. 4J

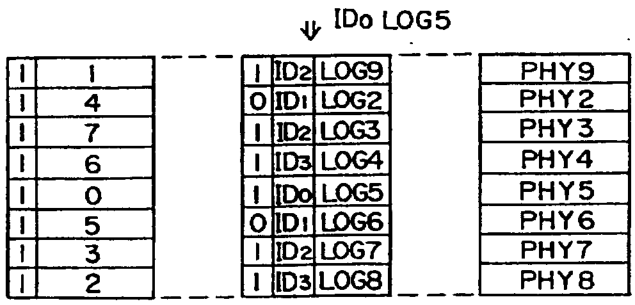


FIG. 4K

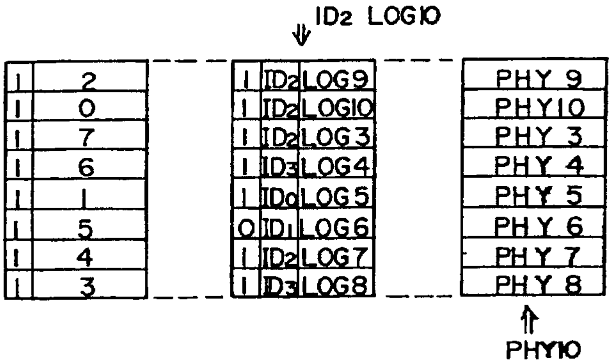

ID2 LOG10
1 2
1 0
1 7
1 6
1 1
1 5
1 4
1 3
1 ID2 LOG9
1 ID2 LOG10
1 ID2 LOG3
1 ID3 LOG4
1 ID0 LOG5
0 ID1 LOG6
1 ID2 LOG7
1 ID3 LOG8
PHY 9
PHY10
PHY 3
PHY 4
PHY 5
PHY 6
PHY 7
PHY 8
PHY10

FIG. 4L

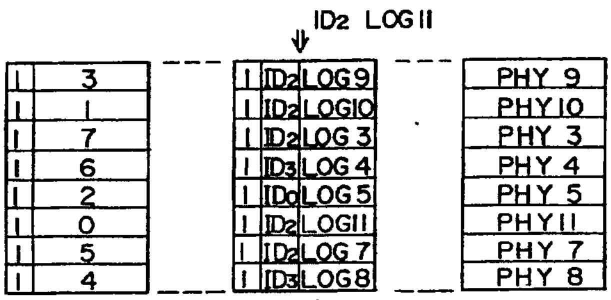
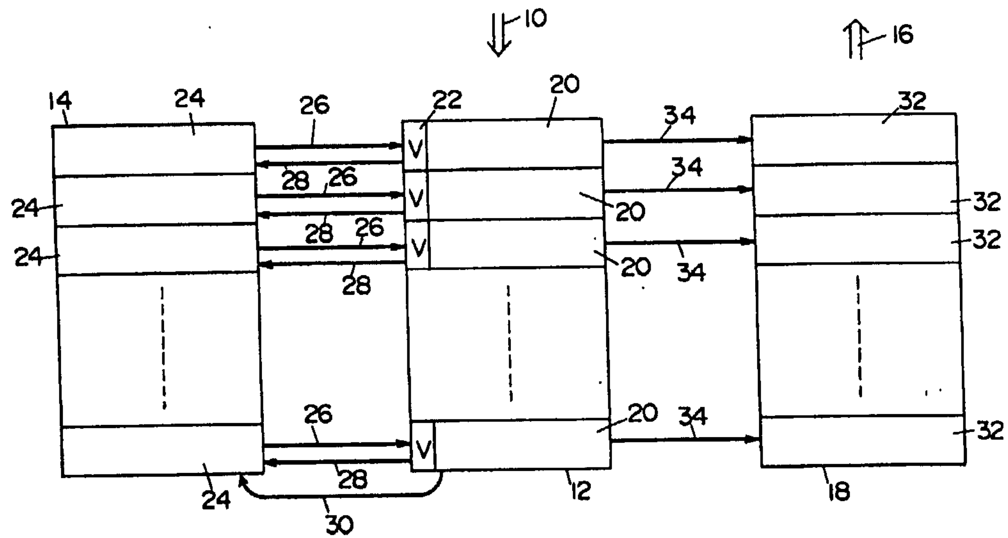

ID2 LOG11
1 3
1 1
1 7
1 6
1 2
1 0
1 5
1 4
1 ID2 LOG9
1 ID2 LOG10
1 ID2 LOG 3
1 ID3 LOG 4
1 ID0 LOG5
1 ID2 LOG11
1 ID2 LOG 7
1 ID3 LOG8
PHY 9
PHY10
PHY 3
PHY 4
PHY 5
PHY11
PHY 7
PHY 8

FIG. 5



ADDRESS CONVERSION APPARATUS

BACKGROUND OF THE INVENTION

This invention relates to an address conversion apparatus used in a computer system employing a microprocessor, and more particularly to an address conversion apparatus capable of efficiently converting from a logical address into a physical address.

In a computer system, the central processing unit outputs a logical address when executing a certain program or a process. Since this logical address merely indicates a virtual address on the program, when actually executing the program, this logical address must be converted into a physical address, that is, the address storing the instruction or data of the memory which stores the content of the practical instruction or data. It is the address conversion apparatus that converts from a logical address into a physical address.

FIG. 5 shows a block diagram of a translation lookaside buffer (TLB) as an address conversion apparatus for converting a logical address into a physical address in the conventional memory management system by paging.

This TLB is composed of a content addressable memory (CAM) 12 for storing the logical address 10 delivered from the CPU, a least recently used circuit (LRU) 14 for controlling the content thereof, and a random access memory (RAM) 18 being accessed by the CAM 12 and delivering a physical address 16. The CAM 12 possesses plural logical address storing parts 20 for storing plural logical addresses. In each logical address storing part 20, a valid bit 22 is provided, and depending on whether the valid bit 22 is 1 or 0, it is known whether the logical address stored in the corresponding logical address storing part 20 is valid (necessary) or invalid (unnecessary). The LRU 14 is composed of a number of least recently used counters 24 corresponding to the plural logical address storing parts 20, and these counters 24 and the logical address storing parts 20 are mutually linked by means of least recently used replace word wires 26 and content addressable memory word wires 28. The CAM 12 and the LRU 14 are joined by way of content addressable memory hit wires 30. The RAM 18 possesses physical address storing parts 32 corresponding to the logical address storing parts 20 of the CAM 12, and the logical address storing parts 20 and the physical address storing parts 32 are linked together by way of random memory access word wires 34.

Usually, when a certain process is executed by a processor, and its logical addresses are converted into physical addresses at a high speed ,by way of the TLB, the operation is effected according to the following procedure.

A certain logical address 10 is fed from the CPU to the CAM 12, and it compared with the logical address stored in the content addressable memory 12. Here, if a logical address coinciding with the input logical address 10 is present, the data corresponding to the physical address stored in the physical address storing part 32 of the RAM 18 corresponding to that logical address is delivered. As a result of this output of the data corresponding to the physical address, the data on that physical address is read out by the CPU or the processor, and is processed.

At the time of the above described logical address retrieval, if no coinciding logical address is present and the content addressable memory 12 is fully filled with the logical address data and it is necessary to delete the logical address data not required for the time being, the least recently used logical address storing part 20 is selected by the LRU 14, and the logical address data storing in that part is erased, and the data of the logical address to be used newly will be stored.

Thus, while a certain process is being executed, the input logical address 10 is converted at high speed by the TLB into an outputted physical address 16, but in the CAM 12 of the TLB, there was not field to recognize the process to be executed. Accordingly, when plural processes, that is, multiprocesses are executed in the processor, if a content switching occurs due to a change-over of the process to be executed, it is necessary to invalidate all data of the logical address newly in each process to update. This is because, even at the same logical address, if the process to be executed is different, the address content differs.

Furthermore, in the multiprocess environment, each process is scheduled, and the processor is used in time sharing, and therefore, in each process, it is necessary to update all logical addresses of the TLB every time changed over by the context switch until the process is completely terminated. Therefore, the system performance was lowered.

SUMMARY OF THE INVENTION

It is hence a primary object of this invention to present an address conversion apparatus capable of converting addresses efficiently even in the environment of frequent context switching.

It is another object of this invention to present an address conversion apparatus comprising a content addressable memory having a field for indicating a process identification number, and capable of storing logical addresses of different processes at the same time.

While the novel features of the invention are set forth in the appended claims, the invention both as to organization and content, will be better understood and appreciated, along with other objects and features thereof, from the following detailed description taken in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of an address conversion apparatus according to one of the embodiments of this invention;

FIG. 2 is a circuit diagram of a content addressable memory in the same apparatus;

FIG. 3 is a circuit diagram of a least recently used circuit of the same apparatus;

FIG. 4A to FIG. 4L are diagrams showing the changes of data in address conversion by using the same apparatus; and

FIG. 5 is a block diagram of a conventional address conversion apparatus.

DETAILED DESCRIPTION OF THE INVENTION

The address conversion apparatus of this invention is described below while referring to FIG. 1 which shows a translation lookaside buffer (TLB) as one of the embodiments thereof. The TLB shown in this drawing is similar in basic structure to the conventional TLB shown FIG. 5 in and identical parts are given same numerals and detailed descriptions are omitted.

The CAM 12 comprises, at the beginning of each logical address storing part 20 for storing the logical address produced from a certain process being executed by the processor, a process identification number part (process ID part) for storing the identification number of that process, a valid bit 22 for indicating the validity of one word stored in one logical address part, and a priority encoder (PENC) 38 for specifying by selecting a specific invalid word disposed physically at a higher position than the word by referring to the valid bit 22.

Furthermore, the CAM 12 also comprises a batch reset wire 40 for initializing the valid bits by simultaneously resetting all valid bits 22, a process identification number batch reset wire (process ID batch reset wire) 42 for referring to the process ID part 36, and for resetting the valid bit 22 of the plural words having the same unnecessary process identification number (process ID) when the process is unnecessary or when the processes handled by the processor exceed the preset process control number, a priority encoder word wire (PENC word wire) 40 for transmitting the specific word specified by a priority encoder (PENC) 38 to the LRU 14, and a content addressable memory full wire (CAM full wire) 41 for indicating that the CAM 12 is filled with valid data without any invalid word, to the LRU 14.

In the physical address part 32 of the random access memory 18 is stored the address for storing the physical address which stores the data or instruction corresponding to the process identification number of the process ID part 36 on each word of the CAM 12 and the logical address of the logical address storing part 20.

The LRU 14 comprises a counter 24, a counter valid bit 44 to indicate whether the value of this counter 24 is valid, a least recently used batch reset wire 46 for batch-resetting this counter valid bit 44, and a least recently used control circuit 48.

Referring now to FIG. 2 through FIG. 4, the circuit structure of the TLB is described more specifically below, and the operation of the process ID part 36 which is one of the features of this invention during operation of TLB is also explained in detail.

FIG. 2 is a circuit diagram specifically illustrating the CAM 12 in FIG. 1. The CAM 12 shown comprises logical address storing parts 20 composed of plural data bit cells 49, and a process ID part 36 for storing, for example, four process IDs 0, 1, 2, 3, and in the process ID part, for example, there are two process identification number cells (ID_1, ID_2) 50, 52 for setting the four process IDs 0 to 3.

For instance, when the least recent used replace word wire 26 becomes 1 and replacement is effected for storing new logical data, a node 22a is set High, and the valid bit 22 is set at 1. At the same time, the process identification number of the address data and the logical address are entered into process identification number cells (ID_1, ID_2) 50, 52 and plural data bit cells 49 which make up the logical address storing part 20.

When the process identification number batch reset of this invention is effected, a signal "1" is applied to an arbitrary one of the four process identification batch reset wires 42 to indicate four processes from 0 to 3, for example, the reset wire 42a corresponding to the second process. In the process identification number part 36, the data corresponding to the second process, that is, the data, for example, of which process identification number cell ID_1 50 is "1" and process identification number cell ID_2 52 is "0" outputs a control signal of "1" to the reset circuit 51, and the AND of this control signal and the control signal "1" of said reset wire 42a is obtained in this reset circuit 51, and as a result of this product, the node 22a of the valid bit 22 becomes Low, and that word becomes invalid. This processing is conducted on all words having the same process identification number, and each invalid signal is entered into the priority encoder 38. In the priority encoder 28, with respect to the input of these plural invalid signals, they are indicated to the LRU 14 as being reloadable word regions of logical addresses, sequentially one by one, from the higher ones (the words at higher positions in FIG. 2). Therefore, if the process identification numbers are reset in batch and plural reloadable words should occur in the CAM 12, only the word at the highest position is noticed to the LRU 14 as a reloadable word. At the LRU 14, receiving this notice, when the data of writing logical address is newly entered into the TLB, this new logical address data is written into the word at the highest position, and the priority encoder 38 of the CAM 12 delivers the next word in the priority order as the word at the highest position to the LRU 14.

Furthermore, when the CAM 12 is filled completely with valid data without any invalid word, the nonactive state of the priority encoder 38 is detected, and it is transmitted to the LRU 14 through the CAM full wire 49.

The content addressable memory hit wire 30 is connected to each word, and an indication as to whether each word is hit or not is forwarded to the LRU 14. Numeral 54 is a dummy word part for adjusting the timing of retrieval of the CAM 12.

FIG. 3 shows the portion of the one word of LUR 14 of the TLB. The LRU is roughly divided into the counter part 24 and the other LRU control circuit 38.

The counter part 24 is a 5-bit counter, and each bit (24a to 24e) has a counter data part 62, a carry propagation part 64 for propagating the carry of the counter, a reset part 72 for resetting the counter, and a comparator part 70 for comparing the counter value with other word. The LRU control circuit 48 comprises a counter valid bit 44 to indicate whether the counter of the word is valid, a carry generating part 76 for generating a carry only to the words in which the counter valid bit 44 has been set so far if the comparative word wire 68 becomes active or CAM mishit should occur as the retrieved logical address is not present, and an LRU replace word generating part 78 for making the LRU replace word wire 26 active for the purpose of keeping uniformity of the LRU 14 and CAM 12, RAM 18 by referring to the PENC word wire 40 and CAM full wire 41, and CAM hit wire 30 and replace enable signal 86. Numerals 140, 142, 144 are the clock wires for adjusting the timing.

The LRU control circuit 48, if there is the same as the retrieval logical address in its word and the hit signal from the CAM 12 is entered into the LRU control circuit 48 through the content addressable memory hit word wire 30, transmits the value of the counter of that word to the counter reference bit wires 66a to 66e in each bit 24a to 24e of the counter part 24.

If there is no hit, on the other hand, the value of the counter of other word being hit is received from the counter reference bit wires 66a to 66e, and it is compared with the value of the own counter in the comparator 70, and if the value of the own counter is smaller than the value of this hit counter, the comparative word wire 68 is made active, and this signal is transmitted to the carry generating part 76, and the generated carry is transmitted to the counter part 24 through the carry wire 80, and the value of the counter is incremented by 1.

These actions of the LRU circuit are practically described below. In short, it is intended to prepare for updating the content of the logical address of the word to the content of the logical address of the highest possibility of use, by always recognizing in the TLB the most recently used word, or, in other words, by always recognizing the least recently used word out of N words in the TLB. For this purpose, data of logical addresses are stored in N words in the TLB, and for example, when the data of a certain word is used at the k-th time out of N words, the value of the counter part 24 of the LRU 14 is k. Accordingly, by the next command, if the logical address of this k-th word is used, the counter part 24 of this word is set to 0, and the counters of all words having so far the values of 0 to (k-1) are increased by 1, so that the most recently used word can be always recognized as the counter value becomes 0, or the least recently used word can be recognized as the counter value becomes N.

Meanwhile, the explanation of RAM 18 is omitted because it is a very common one designed to deliver the content to a certain address.

This has been a brief explanation of an embodiment of the TLB of this invention by CAM 12, LRU 14, and RAM 18. Below is described the practical operation of the TLB capable of identifying the process by this invention, mainly relating to the LRU 14.

The circuit action, is explained in FIG. 2, FIG. 3, and data changes of the TLB in action are given in FIG. 4. Here, the TLB is explained as 8 entries (8 words).

When the TLB capable of identifying the process of this invention is operated, two cases are roughly considered.

(1) Ordinary action (not erased by process ID batch reset wire 42, and valid bit 22 of CAM 12 and counter valid bit 44 of LRU 14 are matched).

(2) Extraordinary action (erased by process ID batch reset wire 42, and valid bit 22 of CAM 12 and counter valid bit 44 of LRU 14 are not matched).

These actions are further described below.

(1) Ordinary action

For initialization of the TLB, the batch reset wire 40 for initializing the valid bit 22 is made active in the CAM 12, and the LRU batch reset wire 46 in the LRU, and the value of the counter 24 and the counter bit 44 to see if the counter is effective or not is reset, and the TLB is initialized. At this time, the data holding each element of the TLB becomes as shown in FIG. 4A. The solid line in FIG. 4A shows that the data is present, and 0 of the counter valid bit 44, valid bit 22, and counter part 24 indicates "reset" and the subsequent 1 denotes "set" (valid).

(i) When a new ID and a logical address (ID_0, LOG_1) get into the CAM 12, since there is no word in which a valid bit 22 is set in the CAM 12, the content addressable memory bit wire 30 becomes inactive, and a mis-hit is transmitted to the LRU circuit 14. At the same time, from the outside the data to be written into the RAM 18 is transferred, and a replace enable signal wire 86 becomes active. Here, at the LRU 14, the LRU replace word wire 26 is made active by the LRU replace word generating part 78 of the word located physically higher as seen from one direction, while the counter valid bit 44 of the same word is set, and as a result of this series of actions, the content of the TLB changes from FIG. 4A to FIG. 4B.

(ii) Furthermore, when a new ID and a logic address (ID_1, LOG_2) get into the CAM 12, the content addressable memory hit wire 30 becomes inactive again, and the CAM 12 indicates a mis-hit. At this time, the carry generating part 76 of the word in which the address was stored before generates a carry, and the counter is increased by 1, and, the word to be set this time is set in the same process as in i) above as a result of mis-hit, and the logical address is newly stored. At this time, the content of the TLB changes from FIG. 4B to FIG. 4C. When mis-hit is repeated several times, the same operations of i) and ii) are repeated, and the content of TLB becomes as in FIG. 4D.

(iii) Afterwards, suppose the previously stored logical addresses (ID_1, LOG_2) get in. At this time, the CAM 12 makes the CAM hit wire 30 active, and indicates that the logical address entered into the LRU 14 has been hit. The LRU 14 receives it, and the comparator part 70 of the word which has been hit by the CAM 12 transmits the data of the counter data part 62 to the counter reference bit wires 66. In the other words, the individual counter data parts 62 and the counter reference bit wires 66 are compared, and when the value of the own counter is larger than or equal to the value of the counter reference bit wires 66 to be referred to, the comparative word wire 68 is made active by this comparator part 70, and this signal is transmitted to the carry generating part 76. Receiving this signal, at the carry generating part 76, if the counter valid bit 44 has been set, a carry is generated, and the carry is propagated to the carry propagation part 64 through the carry wire 80. As a result, in the word in which a carry has occurred, the counter is increased by 1 only, but the counter value is not changed in the word having a counter value of larger than the hit word.

As for the counter of the hit word, the reset wire 82 is made active by the logical gate 114, and the value of the counter is cleared. By these actions, when the hit word is the second one from the top, the content of the TLB changes from FIG. 4D to FIG. 4E.

When several of such addresses get in and hit and mis-hit are further repeated, changing from FIG. 4E to FIG. 4F, the TLB is filled up, and the CAM full wire 41 becomes an active state, which is received by the LRU 14.

(iv) In the filled state of the CAM 12 as shown in FIG. 4F, when logical address process IDs not referred red to so far (ID_2, LOG_9) enter, the CAM 12 transmits the mis-hit to the LRU 14 in the same manner as mentioned above. When a replace signal 86 returns from outside, the LRU 14 generates carries for all words in the logical gate 102 of the carry generating part 76 because the CAM 12 is full. As a result of this carry generation, of all words of the LRU 14, the highest position carry wire 84 of only one word (in this example, the word of which counter of LRU 14 changes from 7 to 8) is made active, and the counter valid bit 44 is temporarily reset, and it is transmitted to the LRU replace word generating part 78, and the LRU replace word wire 26 is made active. At the same time, the counter valid bit 44 is set again. Later, as for the words of which the LRU replace word wire 26 is active, data is written into the CAM 12 and RAM 18. At this time, the content of the TLB changes from FIG. 4F to FIG. 4G.

(2) Extraordinary action

(i) Of the data stored so far in the TLB, if an unnecessary process should occur, for example, supposing process ID_1 is unnecessary, the state changes from FIG. 4G to FIG. 4H by using the circuit for resetting the valid bit of the plural words having the same process ID and the process ID batch reset wire 42. At this time, the reset circuit 51 of the CAM 12 turns on only the transistor connected only to the valid bit 22 having the process ID to be erased, and the valid bit 22 is reset. At this time, the PENC 38 of the CAM 12 makes active and sets only the word located at the highest position as seen from one direction in the physical configuration, out of the words being erased.

Here, suppose the logical address and process ID (ID_0, LOG_5) to hit get into the CAM 12. At this time, the LRU 14 refers to the CAM hit signal wire 30 and the counter valid bit 44, and since the values of all counters are legal, the operation iv) of the above ordinary action is effected, and the content of the TLB is changed from FIG. 4H to FIG. 4J.

(ii) Finally, in this state, suppose the logical address and process ID (ID,LOG_{10}) to mis-hit the CAM get in.

At this time, the PENC 38 of the CAM 12 makes active only the word at the highest position as seen from one direction in the physical configuration, and makes inactive the CAM full wire 41. At this time, the CAM hit wire 30 is inactive.

In this state, the signal of PENC word wire 70 and the value of counter valid bit 12 are entered into the logical gate 118 of the LRU replace word generating part 78. Here, in the word of which value of the counter valid bit 12 is "1" and PENC word wire 40 is active, the output of the logical gate 118 becomes active, and this signal makes active the LRU hit word wire 92. As a result, as mentioned in step iv) of ordinary action, the TLB sets 0 the value of the counter of LRU 14 of the word which has been hit, as if the stored logical address had been hit, and increases the counter value by 1 as for the words requiring increment.

At the same time, different from step iv) of ordinary action, when the LRU replace enable signal 86 returns, the output of the LRU replace word generating part 78, that is, the LRU replace word wire 26 becomes active. By this active LRU replace word wire 26, the data of the CAM 12 and RAM 18 are updated. At this time, the content of the TLB changes as shown in FIG. 4K. Then by repeating such mis-hit, the logical addresses causing mis-hit are stored in the place of the word where the valid bit 22 of CAM 12 is cleared, and the information of the physical addresses to be converted is stored in the process ID, RAM 11, and as the LRU 14 goes on resetting the counter value of the word, the content of the TLB changes from FIG. 4K to FIG. 4L, thereby returning to the ordinary TLB content. Hence, even after occurrence of erasure by the process ID batch reset wire 40, the uniformity of the TLB may be maintained by the operation described above.

By using this TLB, the following effects are expected.

(1) If context switching should occur, it is not necessary to reset the content of TLB.

(2) The data of only the unnecessary processes can be erased.

(3) It is possible to store the data of different processes having an identical physical address into the TLB.

Owing to these effects (1) to (3), it is possible to use a high speed translation lookaside buffer (TLB) very effectively on multiprocesses, so that the processing speed of the processor may be dramatically enhanced.

While specific embodiments of the invention have been illustrated and described herein, it is realized that other modifications and changes will occur to those skilled in the art. It is therefore to be understood that the appended claims are intended to cover all modifications and changes as fall within the true spirit and scope of the invention.

We claim:

1. An apparatus for converting a logical address outputted by a processor into an equivalent physical address, said apparatus comprising:

a content addressable memory;

a random access memory coupled to said content addressable memory; and,

a least recently used circuit coupled to said content addressable memory;

said content addressable memory including a means for providing a hit word indication to the processor, said hit word indication indicating that a corresponding identification number of a process being processed by the processor and a corresponding logical address of said process is stored in said content addressable memory, and further indicating that the equivalent physical address stored in a corresponding word location of said random access memory is accessible;

said content addressable memory further including a means for providing a miss-hit word indication to the processor, said miss-hit word indication indicating the absence of a corresponding identification number and corresponding logical address of said process, and further indicating that the processor is to search a main memory to locate the equivalent physical address of the logical address, wherein the logical address is stored in said content addressable memory at a word location indicated by said least recently used circuit, and wherein the thus located physical address is stored in a corresponding word location of said random access memory;

said content addressable memory further including: a logical address area for storing the logical address of said process in each word location of said content addressable memory; a process identification number storage area for storing the process identification number in each word location of said content addressable memory; a valid bit for providing an indication of the validity of data stored in both said logical address area and said process identification number storage area in each word location of said content addressable memory; a word line for providing a matching word located during a search of said content addressable memory; a content addressable memory bit line for indicating whether said matching word exists in said content addressable memory, and a content addressable memory full line for providing an indication as to whether words in said content addressable memory are used by referring to said valid bit area of each word location of said content addressable memory;

said random access memory including a physical address area for storing the physical address of the main memory to be accessed by said processor;

said least recently used circuit including a counter data area having a reset portion for indicating a sequence of searching and reading of data of each word stored in said least recently used circuit; a valid bit for indicating the validity of data stored in said counter data area, a counter data reference bit line for providing counter data of a matching word when searching said content addressable memory and being commonly connected to the counter data area of each word; a comparator area disposed in each word location for comparing the counter data of the reference bit and the counter data of other words; a carry area disposed in each word for receiving a value stored in said counter data area of each word, and for selectively varying said value by one, and for resetting said counter data area, and a replace word generating area for specifying a word to be input to said content addressable memory in accordance with a carry signal from said carry area;

wherein, when said least recently used circuit determines that there is an absence of a matching word in said content addressable memory on the basis of said content addressable memory bit line, and further determines that said content addressable memory is fully unoccupied on the basis of said content addressable memory full line, the value stored in said counter data area is increased by one, and the valid bit of said counter data area of an unused word is set, and the logical address and process identification number received from the processor and the valid bit corresponding to the unused word are set in said content addressable memory in a corresponding word location, and a physical address located by the processor in said main memory is set in said random access memory in a corresponding word location,

wherein, when said least recently used circuit determines that there is an absence of a matching word in said content addressable memory on the basis of said content addressable memory bit line, and further determines that said content addressable memory is fully occupied on the basis of said content addressable memory full line, the value stored in said counter data area and the carry area of each word are simultaneously increased by one, and the logical address and the process identification number received from the processor are set in the corresponding content addressable memory of the replace word generated in accordance with said replace word generating area by using a carry signal generated in said carry unit, and the physical address located by the processor in the main memory is set in the random access memory at a corresponding word location of said random access memory, and

wherein, when said least recently used circuit determines that there is a matching word in said content addressable memory on the basis of said content addressable memory bit line, said least recently used circuit receives said word line indicating the matching word from said content addressable memory, and a value of said counter data of a corresponding word is transmitted to each word through the counter data reference bit line, and the value of said counter data of said matching word and the value of the counter data of each other word are compared, and wherein the value of the counter data of a word having a value smaller than that of the counter data of the matching word is increased by one, and the counter data of the matching word is rendered to a value of an initial setting by said reset unit.

2. An apparatus as recited in claim 1, further comprising a priority encoder for selecting one of the words in which said valid bit is absent, and for transmitting the thus selected word to said least recently used circuit, wherein, when there is an absence of a matching word in said content addressable memory, said least recently used circuit inputs the thus selected word specified by said priority encoder.

* * * * *

PAT-NO: JP02003044510A

DOCUMENT-IDENTIFIER: JP 2003044510 A

TITLE: GATEWAY SYSTEM

PUBN-DATE: February 14, 2003

INVENTOR-INFORMATION:

NAME	COUNTRY
INOSHITA, AKIHITO	N/A
SUZUKI, HIROYOSHI	N/A
KUBOTA, HIROMI	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
MATSUSHITA ELECTRIC IND CO LTD	N/A

APPL-NO: JP2001225981

APPL-DATE: July 26, 2001

INT-CL (IPC): G06F017/30, G06F012/00 , G06F013/00

ABSTRACT:

PROBLEM TO BE SOLVED: To provide a gateway system that enables a network

terminal user to automatically surf valuable Web pages without any specified setting.

SOLUTION: An access monitor unit 25 of a gateway system 80 detects the URL for Webs a user frequently accesses and manages the URL with a URL management table 30. A surfing unit 40 of the gateway system automatically surfs the Webs having the URL and stores the Web data in a cache server 50. The gateway system generate a management table that includes not only the frequency of the accesses but also data for the elapsed time from the most recent accessed time to the present time and can automatically surf a Web site being judged as the high priority site based on the management table.

Form PTO-948 (Rev. 06/03)
Application No. 09/608266

U.S. DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

The drawing(s) filed (insert date) 6/30/00 are:

A. ____ approved by the Draftsperson under 37 CFR 1.84 or 1.152.

B. X objected to by the Draftsperson under 37 CFR 1.84 or 1.152 for the reasons indicated below. Corrected drawings are required.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink or Color (3 sets required).
___ Color drawings are not acceptable until petition is granted. Fig(s)________
___ Pencil and non black ink not permitted. Fig(s)____

2. PHOTOGRAPHS. 37 CFR 1.84(b)
___ One (1) full-tone set is required. Fig(s)________
___ Photographs may not be mounted. 37 CFR 1.84(e)
___ Photographs must meet paper size requirements of 37 CFR 1.84(f). Fig(s)________
___ Poor quality (half-tone). Fig(s)________

3. TYPE OF PAPER. 37 CFR 1.84(e)
___ Paper not flexible, strong, white, and durable. Fig(s)________
___ Erasures, alterations, overwritings. interlineations, folds, copy machine marks not accepted. Fig(s)________

4. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
21.0 cm by 29.7 cm (DIN size A4) or
21.6 cm by 27.9 cm (8 1/2x 11 inches)
___ All drawing sheets not the same size. Sheet(s)________
___ Drawings sheets not an acceptable size. Fig(s)____

5. MARGINS. 37 CFR 1.84(g): Acceptable margins:
Top 2.5 cm Left 2.5 cm Right 1.5 cm Bottom 1.0 cm
___ Margins not acceptable. Fig(s) 3, 14, 21
____ Top (T) X Left (L)
____ Right (R) ____ Bottom (B)

6. VIEWS. 37 CFR 1.84(h)
REMINDER: Specification may require revision to correspond to drawing changes, e.g., if Fig. 1 is changed to Fig. 1A, Fig 1B and Fig. 1C, etc., the specification, at the Brief Description of the Drawings, must likewise be changed.
___Views not labeled separately or properly. Fig(s)________

7. SECTIONAL VIEWS. 37 CFR 1.84(h)(3)
____ Sectional designation should be noted with Arabic or Roman numbers. Fig(s)________

8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)________

9. SCALE. 37 CFR 1.84(k)
___ Scale not large enough to show mechanism without crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s)________

10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
___ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (poor line quality). Fig(s)________

11. SHADING. 37 CFR 1.84(m)
___ Solid black areas pale. Fig(s) ________
___ Solid black shading not permitted. Fig(s)________

12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
___ Numbers and reference characters not plain and legible. Fig(s)________
___ Figure legends are poor. Fig(s)________
___ Numbers and reference characters not oriented in the same direction as the view. 37 CFR 1.84(p)(1) Fig(s)________
___ English alphabet not used. 37 CFR 1.84(p)(2) Fig(s)________
___ Numbers, letters and reference characters must be at least 32 cm (1/8 inch) in height. 37 CFR 1.84(p)(3). Fig(s)________

13. LEAD LINES. 37 CFR 1.84(q)
___ Lead lines missing. Fig(s)________

14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
___ Sheets not numbered consecutively, and in Arabic numbers beginning with number 1. Sheet(s)____

15. NUMBERING OF VIEWS. 37 CFR 1.84(u)
___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)____

16. DESIGN DRAWINGS. 37 CFR 1.152
___ Surface shading shown not appropriate. Fig(s)________
___ Solid black surface shading is not permitted except when used to represent the color black as well as color contrast Fig(s)________

COMMENTS:

Reviewer A. D

Date 8/29/03

If you have questions, call (703) 305-8404.

Attachment to Paper No. 6

☆ U.S GOVERNMENT PRINTING OFFICE: 2003-300-153

Form PTO-948 (Rev. 06/03)
Application No. 09/608266

U.S. DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

The drawing(s) filed (insert date) 6/30/00 are:

A. ___ approved by the Draftsperson under 37 CFR 1.84 or 1.152.

B. X objected to by the Draftsperson under 37 CFR 1.84 or 1.152 for the reasons indicated below. Corrected drawings are required.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink or Color (3 sets required).
___ Color drawings are not acceptable until petition is granted. Fig(s)___
___ Pencil and non black ink not permitted. Fig(s)___

2. PHOTOGRAPHS. 37 CFR 1.84(b)
___ One (1) full-tone set is required. Fig(s)___
___ Photographs may not be mounted. 37 CFR 1.84(e)
___ Photographs must meet paper size requirements of 37 CFR 1.84(f). Fig(s)___
___ Poor quality (half-tone). Fig(s)___

3. TYPE OF PAPER. 37 CFR 1.84(e)
___ Paper not flexible, strong, white, and durable. Fig(s)___
___ Erasures, alterations, overwritings. interlineations, folds, copy machine marks not accepted. Fig(s)___

4. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
21.0 cm by 29.7 cm (DIN size A4) or
21.6 cm by 27.9 cm (8 1/2x 11 inches)
___ All drawing sheets not the same size. Sheet(s)___
___ Drawings sheets not an acceptable size. Fig(s)___

5. MARGINS. 37 CFR 1.84(g): Acceptable margins:
Top 2.5 cm Left 2.5 cm Right 1.5 cm Bottom 1.0 cm
___ Margins not acceptable. Fig(s) 3, 14, 21
___ Top (T) X Left (L)
___ Right (R) ___ Bottom (B)

6. VIEWS. 37 CFR 1.84(h)
REMINDER: Specification may require revision to correspond to drawing changes, e.g., if Fig. 1 is changed to Fig. 1A, Fig 1B and Fig. 1C, etc., the specification, at the Brief Description of the Drawings, must likewise be changed.
___ Views not labeled separately or properly. Fig(s)___

7. SECTIONAL VIEWS. 37 CFR 1.84(h)(3)
___ Sectional designation should be noted with Arabic or Roman numbers. Fig(s)___

8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)___

9. SCALE. 37 CFR 1.84(k)
___ Scale not large enough to show mechanism without crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s)___

10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
___ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (poor line quality). Fig(s)___

11. SHADING. 37 CFR 1.84(m)
___ Solid black areas pale. Fig(s)___
___ Solid black shading not permitted. Fig(s)___

12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
___ Numbers and reference characters not plain and legible. Fig(s)___
___ Figure legends are poor. Fig(s)___
___ Numbers and reference characters not oriented in the same direction as the view. 37 CFR 1.84(p)(1) Fig(s)___
___ English alphabet not used. 37 CFR 1 84(p)(2) Fig(s)___
___ Numbers, letters and reference characters must be at least 32 cm (1/8 inch) in height. 37 CFR 1.84(p)(3). Fig(s)___

13. LEAD LINES. 37 CFR 1.84(q)
___ Lead lines missing. Fig(s)___

14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
___ Sheets not numbered consecutively, and in Arabic numbers beginning with number 1. Sheet(s)___

15. NUMBERING OF VIEWS. 37 CFR 1.84(u)
___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)___

16. DESIGN DRAWINGS. 37 CFR 1.152
___ Surface shading shown not appropriate. Fig(s)___
___ Solid black surface shading is not permitted except when used to represent the color black as well as color contrast Fig(s)___

COMMENTS:

Reviewer A.D

Date 8/29/03

If you have questions, call (703) 305-8404.

Attachment to Paper No. 6

☆ U.S. GOVERNMENT PRINTING OFFICE: 2003-300-153

7/A
1/Alvins

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

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.*	Group Art Unit: 2662
Application No.: 09/608266	Examiner: Alan Nguyen
Filed: June 30, 2000	
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	

RESPONSE TO OFFICE ACTION UNDER 37 CFR 1.111

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is a response to the Office Action of September 10, 2003.

Any ***amendments to the specification*** begin on a new page immediately after these introductory remarks.

Any ***amendments to the claims*** begin on a new page immediately after such ***amendments to the specification***, if any.

Any ***amendments to the drawings*** begin on a new page immediately after such ***amendments to the claims***, if any.

The ***Remarks/arguments*** begin on a new page immediately after such ***amendments to the drawings***, if any.

If there are drawing amendments, an ***Appendix*** including amended drawings is attached following the ***Remarks/arguments***.

Certificate of Facsimile Transmission under 37 CFR 1.8

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, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 10, 2004

Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

AMENDMENT(S) TO THE CLAIMS:

The following listing of claims will replace all prior versions, and listings, of claims on the application. All claims are set forth below with one of the following annotations.

- (Original): Claim filed with the application.
- (Currently amended): Claim being amended in the current amendment paper.
- (Canceled): Claim cancelled or deleted from the application. No claim text is shown.
- (Withdrawn): Claim still in the application, but in a non-elected status.
- (Previously presented): Claim being added in the current amendment paper.
- (Previously presented): Claim added or amended in an earlier amendment paper.
- (Not entered): Claim presented in a previous amendment, but not entered or whose entry status unknown. No claim text is shown.

The following listing of claims assumes the amendment submitted on 10 February 2004 has been entered.

Sub B1

1. (Previously presented) 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 flow-entries for previously encountered conversational flows to which a received packet may belong, a conversational flow being an exchange of one or more packets in any direction as a result of an activity corresponding to the flow;

   (c) a cache subsystem coupled to the flow-entry database memory providing for fast access of flow-entries from the flow-entry database;

   (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 the cache subsystem; and

   (e) a state processor coupled to the lookup engine and to the flow-entry-database memory, the state processor being to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

A1

2. (Original) 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. (Original) 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. (Currently amended) 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~~ a 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. (Original) 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. (Original) 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 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 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.–20. (Cancelled).

21. (Currently amended) 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 packet acquisition device coupled to the connection point and configured to receive packets passing through the connection point;

an input buffer memory coupled to and configured to accept a packet from the packet acquisition device;

a parser subsystem coupled to the input buffer memory, the parsing subsystem configured to extract selected portions of the accepted packet and to output a parser record containing the selected portions;

a memory to storing a database of one or more flow-entries for any previously encountered conversational flows, each flow-entry identified by identifying information stored in the flow-entry;

a lookup engine coupled to the output of the parser subsystem and to the flow-entry memory and configured to lookup whether the particular packet whose parser record is output by the parser subsystem has a matching flow-entry, the looking up using at least some of the selected packet portions and determining if the packet is of an existing ~~flow~~ flow;

a cache subsystem coupled to and between the lookup engine and the flow-entry database memory providing for fast access of a set of likely-to-be-accessed flow-entries from the flow-entry database; and

a flow insertion engine coupled to the flow-entry memory and to the lookup engine and configured to create a flow-entry in the flow-entry database, the flow-entry including identifying information for future packets to be identified with the new flow-entry,

the lookup engine configured such that if the packet is of an existing flow, the monitor classifies the packet as belonging to the found existing flow; and if the packet is of a new flow, the flow insertion engine stores a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the operation of the parser subsystem depends on one or more of the protocols to which the packet conforms.

Cont A1

~~21~~22. (Currently amended) A monitor according to ~~claim 20~~ claim 21, wherein the lookup engine updates the flow-entry of an existing flow in the case that the lookup is successful.

~~22~~23. (Currently amended) A monitor according to ~~claim 20~~ claim 21, further including a mechanism for building a hash from the selected portions, wherein the hash is included in the input for a particular packet to the lookup engine, and wherein the hash is used by the lookup engine to search the flow-entry database.

~~23~~24. (Currently amended) A monitor according to ~~claim 20~~ claim 21, further including a memory containing a database of parsing/extraction operations, the parsing/extraction database memory coupled to the parser subsystem, wherein the parsing/extraction operations are according to one or more parsing/extraction operations looked up from the parsing/extraction database.

~~24~~25. (Currently amended) A monitor according to ~~claim 33~~ claim 24, wherein the database 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.

~~25~~26. (Currently amended) A method according to ~~claim 20~~ claim 21, further including a state processor coupled to the lookup engine and to the flow-entry-database memory, and configured to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

~~26~~27. (Currently amended) A method according to ~~claim 25~~ claim 26, wherein the set of possible state operations that the state processor is configured to perform includes searching for one or more patterns in the packet portions.

~~27~~28. (Currently amended) A monitor according to ~~claim 25~~ claim 26, wherein the state processor is programmable, the monitor further including a state patterns/operations memory coupled to the state processor, the state operations memory configured to store a database of protocol dependent state patterns/operations.

~~28~~29. (Currently amended) A monitor according to ~~claim 25~~ claim 26, wherein the state operations include updating the flow-entry, including identifying information for future packets to be identified with the flow-entry.

~~29~~30. (Currently amended) A method of examining packets passing through a connection point on a computer network, each packets conforming to one or more protocols, the method comprising:

(a) receiving a packet from a packet acquisition device;

(b) performing one or more parsing/extraction operations on the packet to create a parser record comprising a function of selected portions of the packet;

(c) looking up a flow-entry database comprising none or more flow-entries for previously encountered conversational flows, the looking up using at least some of the selected packet portions and determining if the packet is of an existing flow, the lookup being via a cache;

(d) if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

(e) if the packet is of a new flow, storing a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the parsing/extraction operations depend on one or more of the protocols to which the packet conforms.

~~30~~31. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein classifying the packet as belonging to the found existing flow includes updating the flow-entry of the existing flow.

Cont A1

~~31~~32. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein the function of the selected portions of the packet forms a signature that includes the selected packet portions and that can identify future ~~packers~~ packets, wherein the lookup operation uses the signature and wherein the identifying information stored in the new or updated flow-entry is a signature for identifying future packets.

~~32~~33. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein the looking up of the flow-entry database uses a hash of the selected packet portions.

~~33~~34. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein step (d) includes if the packet is of an existing flow, obtaining the last encountered state of the flow and performing any state operations specified for the state of the flow starting from the last encountered state of the flow; and wherein step (e) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow.

REMARKS

Claims 1–6 and 21–33 (including two claims numbered 21 prior to this amendment) are the claims of record of the application. A response to an office action was filed 10 February 2004.

The examiner has indicated to the undersigned that there were two claim 21s in the listing of claims in the response filed 10 February 2004.

The present amendment corrects several typographical errors found in both the original application and the previous amendment filed on 10 February 2004. The present amendment assumes that the previous amendment has been entered.

The undersigned discovered the previous amendment incorrectly annotated claims 2–6 as "previously presented" instead of being annotated as "original." The present amendment correctly annotates the claims.

The present amendment corrects the typographical error in the previous amendment of there being two claim 21s. The second instance of claim 21 has been renumbered claim 22, and previous claims 22–33 have been renumbered to claims 23–34, respectively. In addition, newly numbered claims 22–24, 26–29, 31–34 have been amended to depend on the appropriate newly numbered claims.

Claim 24 of the previous amendment was erroneously dependent on claim 33. The present amendment corrects this typographical error-newly numbered claim 25 depends on newly numbered claim 24.

Minor typographical errors were found claims 4, 21, and newly-numbered 32. The present amendment corrects these typographical errors.

No new matter has been added by this amendment.

The Applicants believe that the remaining claims are allowable. Action to that end is respectfully requested.

If the Examiner has any questions or comments that would advance the prosecution and allowance of this application, an email message to the undersigned at dov@inventek.com, or a telephone call to the undersigned at +1-510-547-3378 is requested.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985
Email: dov@inventek.com

INVENTEK

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510) 291-2985
dov@inventek.com

Fax

Patent Application Ser. No.: 09/608266	***Ref./Docket No***: APPT-001-4
Applicant(s): Sarkissian, *et al.*	***Examiner.***: Alan Nguyen
Filing Date: June 30, 2000	***Art Unit***: 2662

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Alan Nguyen, Art Unit 2662)

Fax No.: 703-872-9306

DATE: February 20, 2004

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action (13)

Number of pages including cover: 13

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO
EXAMINER ALAN NGUYEN, ART UNIT 2662

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004

Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/608266
Filing Date	30 Jun 2000
First Named Inventor	Sarkissian, Haig A.
Group Art Unit	2862
Examiner Name	Alan Nguyen
Attorney Docket Number	APPT-001-4

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form
- ☐ Fee Attached (credit card form)
- ☒ Amendment (Supplementary)
- ☐ ☐ After Final
- ☐ ☐ Affidavits/declaration(s)
- ☐ Extension of Time Request
- ☐ Express Abandonment Request
- ☐ Information Disclosure Statement
- ☐ Certified Copy of Priority Document(s)
- ☐ Response to Missing Parts/ Incomplete Application
- ☐
- ☐ ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53
- ☐ Assignment Papers *(for an Application)*
- ☐ Drawing(s)
- ☐ Licensing-related Papers
- ☐ Petition Routing Slip (PTO/SB/69) and Accompanying Petition
- ☐ To Convert a Provisional Application
- ☐ Power of Attorney, Revocation Change of Correspondence Address
- ☐ Terminal Disclaimer
- ☐ Small Entity Statement
- ☐ Request of Refund
- ☐ After Allowance Communication to Group
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Proprietary Information
- ☐ Status Letter
- ☐ Additional Enclosure(s) *(please identify below)*:
- ☐
- ☐
- ☐

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm *or* Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	February 20, 2004

ADDRESS FOR CORRESPONDENCE

Firm *or* Individual name	Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618, Tel: +1-510-547-3378

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence is being facsimile transmitted with the United States Patent and Trademark Office at Telephone number 703-872-9306 addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: February 20, 2004

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Date	February 20, 2004

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

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: 2662 Examiner: Alan Nguyen

TRANSMITTAL: SUPPLEMENTARY AMENDMENT

P.O. Box No Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a supplementary amendment for the above referenced application.

This application has:

_____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

_X_ No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004 Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

S/N: 09/608266 Page 2

	TOTAL CLAIMS PREVIOUSLY PAID FOR	NEW TOTAL	NO. OF EXTRA CLAIMS	EXTRA CLAIM FEE
TOTAL CLAIMS	20	19	0	$ 0.00
INDEP. CLAIMS	3	3	0	$ 0.00
			TOTAL CLAIM FEES PAYABLE:	0.00

_____ 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.

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

_____ one months ($110) _X_ two months ($420)

_____ two months ($930) _____ four months ($1450)

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

_X_ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

_X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.

_X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

02/11/2004 AWABI1 00000061 09608266
01 FC:1252 420.00 OP

Address for correspondence:
Dov Rosenfeld
5507 College Avenue,Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

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

Patent

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: 2662

Examiner: Alan Nguyen

TRANSMITTAL: SUPPLEMENTARY AMENDMENT

P.O. Box No Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a supplementary amendment for the above referenced application.

This application has:

____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

X No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004

Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

****
*** TX REPORT ***
****

TRANSMISSION OK

TX/RX NO	0111
CONNECTION TEL	17038729306
SUBADDRESS	
CONNECTION ID	
ST. TIME	02/20 12:23
USAGE T	05'18
PGS. SENT	13
RESULT	OK

INVENTEK

Fax

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510) 291-2985
dov@inventek.com

Patent Application Ser. No.: 09/608266	***Ref./Docket No***: APPT-001-4
Applicant(s): Sarkissian, *et al.*	***Examiner.***: Alan Nguyen
Filing Date: June 30, 2000	***Art Unit***: 2662

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Alan Nguyen, Art Unit 2662)

Fax No.: 703-872-9306

DATE: February 20, 2004

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action (13)

Number of pages including cover: 13

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF

INVENTEK

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510) 291-2985
dov@inventek.com

Fax

Patent Application Ser. No.: 09/608266	***Ref./Docket No***: APPT-001-4
Applicant(s): Sarkissian, *et al.*	***Examiner.***: Alan Nguyen
Filing Date: June 30, 2000	***Art Unit***: 2662

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Alan Nguyen, Art Unit 2662)

Fax No.: 703-872-9306

DATE: February 26, 2004

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Copy of Response to Office Action faxed February 20, 2004.

Number of pages including cover: 15

OFFICIAL COMMUNICATION

Attached are 1) a response that was faxed with a proper certificate under 37 CFR 1.8 on February 20, 2004 and 2) A confirmation from our fax machine that the 13 pages were properly received on February 20, 2004.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: ______________________ Signed: ______________________
Name: Amy Drury

INVENTEK

Fax

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510) 291-2985
dov@inventek.com

Patent Application Ser. No.: 09/608266	***Ref./Docket No***: APPT-001-4
Applicant(s): Sarkissian, *et al.*	***Examiner.***: Alan Nguyen
Filing Date: June 30, 2000	***Art Unit***: 2662

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Alan Nguyen, Art Unit 2662)

Fax No.: 703-872-9306

DATE: February 20, 2004

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action

Number of pages including cover: 13

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER ALAN NGUYEN, ART UNIT 2662

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004

Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

TRANSMITTAL FORM (to be used for all correspondence after initial filing)	**Application Number**	08266
	Filing Date	30 Jun 2000
	First Named Inventor	Sarkissian, Haig A.
	Group Art Unit	2662
	Examiner Name	Alan Nguyen
	Attorney Docket Number	APPT-001-4

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form
- ☐ Fee Attached (credit card form)
- ☒ Amendment (Supplementary)
- ☐ ☐ After Final
- ☐ ☐ Affidavits/declaration(s)
- ☐ Extension of Time Request
- ☐ Express Abandonment Request
- ☐ Information Disclosure Statement
- ☐ Certified Copy of Priority Document(s)
- ☐ Response to Missing Parts/ Incomplete Application
- ☐
- ☐ ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53
- ☐ Assignment Papers *(for an Application)*
- ☐ Drawing(s)
- ☐ Licensing-related Papers
- ☐ Petition Routing Slip (PTO/SB/69) and Accompanying Petition
- ☐ To Convert a Provisional Application
- ☐ Power of Attorney, Revocation Change of Correspondence Address
- ☐ Terminal Disclaimer
- ☐ Small Entity Statement
- ☐ Request of Refund
- ☐ After Allowance Communication to Group
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Proprietary Information
- ☐ Status Letter
- ☐ Additional Enclosure(s) *(please identify below):*
- ☐
- ☐
- ☐

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm *or* Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	February 20, 2004

ADDRESS FOR CORRESPONDENCE

Firm *or* Individual name	Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618, Tel: +1-510-547-3378

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence is being facsimile transmitted with the United States Patent and Trademark Office at Telephone number 703-872-9306 addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: February 20, 2004

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Date	February 20, 2004

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

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: 2662 Examiner: Alan Nguyen

TRANSMITTAL: SUPPLEMENTARY AMENDMENT

P.O. Box No Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a supplementary amendment for the above referenced application.

This application has:

_____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

__X__ No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004 Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

S/N: 09/608266 Page 2

	TOTAL CLAIMS PREVIOUSLY PAID FOR	NEW TOTAL	NO. OF EXTRA CLAIMS	EXTRA CLAIM FEE
TOTAL CLAIMS	20	20	0	$ 0.00
INDEP. CLAIMS	3	3	0	$ 0.00
			TOTAL CLAIM FEES PAYABLE:	0.00

_X_ 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 ($420)

_____ two months ($930) _____ four months ($1450)

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

_____ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

_X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.

_X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue,Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

S/N: 09/608266 Page 2

	TOTAL CLAIMS PREVIOUSLY PAID FOR	NEW TOTAL	NO. OF EXTRA CLAIMS	EXTRA CLAIM FEE
TOTAL CLAIMS	20	20	0	$ 0.00
INDEP. CLAIMS	3	3	0	$ 0.00
TOTAL CLAIM FEES PAYABLE:				0.00

_X_ 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 ($420)

_____ two months ($930) _____ four months ($1450)

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

_____ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

- _X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.
- _X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue,Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

8/Supp B

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.*	Group Art Unit: 2662
Application No.: 09/608266	Examiner: Alan Nguyen
Filed: June 30, 2000	
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	

SUPPLEMENTARY AMENDMENT

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is a supplementary amendment for the above reference application.

Any ***amendments to the claims*** begin on a new page immediately after these introductory remarks.

The ***Remarks/arguments*** begin on a new page immediately after such ***amendments to the claims***, if any.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this correspondence is being Facsimile Transmitted to the U.S. Patent and Trademark Office at telephone number 703-872-9306 addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004 Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

AMENDMENT(S) TO THE CLAIMS:

The following listing of claims will replace all prior versions, and listings, of claims on the application. All claims are set forth below with one of the following annotations.

- (Original): Claim filed with the application.
- (Currently amended): Claim being amended in the current amendment paper.
- (Canceled): Claim cancelled or deleted from the application. No claim text is shown.
- (Withdrawn): Claim still in the application, but in a non-elected status.
- (Previously presented): Claim being added in the current amendment paper.
- (Previously presented): Claim added or amended in an earlier amendment paper.
- (Not entered): Claim presented in a previous amendment, but not entered or whose entry status unknown. No claim text is shown.

The following listing of claims assumes the amendment submitted on 10 February 2004 has been entered.

1. (Previously presented) 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 flow-entries for previously encountered conversational flows to which a received packet may belong, a conversational flow being an exchange of one or more packets in any direction as a result of an activity corresponding to the flow;

   (c) a cache subsystem coupled to the flow-entry database memory providing for fast access of flow-entries from the flow-entry database;

   (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 the cache subsystem; and

   (e) a state processor coupled to the lookup engine and to the flow-entry-database memory, the state processor being to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

2. (Original) 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. (Original) 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. (Currently amended) 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~~ a 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. (Original) 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. (Original) 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

Cont B1

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 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.–20. (Cancelled).

21. (Currently amended) 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 packet acquisition device coupled to the connection point and configured to receive packets passing through the connection point;

an input buffer memory coupled to and configured to accept a packet from the packet acquisition device;

a parser subsystem coupled to the input buffer memory, the parsing subsystem configured to extract selected portions of the accepted packet and to output a parser record containing the selected portions;

a memory to storing a database of one or more flow-entries for any previously encountered conversational flows, each flow-entry identified by identifying information stored in the flow-entry;

a lookup engine coupled to the output of the parser subsystem and to the flow-entry memory and configured to lookup whether the particular packet whose parser record is output by the parser subsystem has a matching flow-entry, the looking up using at least some of the selected packet portions and determining if the packet is of an existing ~~flow~~ flow;

a cache subsystem coupled to and between the lookup engine and the flow-entry database memory providing for fast access of a set of likely-to-be-accessed flow-entries from the flow-entry database; and

a flow insertion engine coupled to the flow-entry memory and to the lookup engine and configured to create a flow-entry in the flow-entry database, the flow-entry including identifying information for future packets to be identified with the new flow-entry,

the lookup engine configured such that if the packet is of an existing flow, the monitor classifies the packet as belonging to the found existing flow; and if the packet is of a new flow, the flow insertion engine stores a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the operation of the parser subsystem depends on one or more of the protocols to which the packet conforms.

~~21~~22. (Currently amended) A monitor according to ~~claim 20~~ claim 21, wherein the lookup engine updates the flow-entry of an existing flow in the case that the lookup is successful.

~~22~~23. (Currently amended) A monitor according to ~~claim 20~~ claim 21, further including a mechanism for building a hash from the selected portions, wherein the hash is included in the input for a particular packet to the lookup engine, and wherein the hash is used by the lookup engine to search the flow-entry database.

~~23~~24. (Currently amended) A monitor according to ~~claim 20~~ claim 21, further including a memory containing a database of parsing/extraction operations, the parsing/extraction database memory coupled to the parser subsystem, wherein the parsing/extraction operations are according to one or more parsing/extraction operations looked up from the parsing/extraction database.

~~24~~25. (Currently amended) A monitor according to ~~claim 33~~ claim 24, wherein the database 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.

~~25~~26. (Currently amended) A method according to ~~claim 20~~ claim 21, further including a state processor coupled to the lookup engine and to the flow-entry-database memory, and configured to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

~~26~~27. (Currently amended) A method according to ~~claim 25~~ claim 26, wherein the set of possible state operations that the state processor is configured to perform includes searching for one or more patterns in the packet portions.

~~27~~28. (Currently amended) A monitor according to ~~claim 25~~ claim 26, wherein the state processor is programmable, the monitor further including a state patterns/operations memory coupled to the state processor, the state operations memory configured to store a database of protocol dependent state patterns/operations.

~~28~~29. (Currently amended) A monitor according to ~~claim 25~~ claim 26, wherein the state operations include updating the flow-entry, including identifying information for future packets to be identified with the flow-entry.

~~29~~30. (Currently amended) A method of examining packets passing through a connection point on a computer network, each packets conforming to one or more protocols, the method comprising:

(a) receiving a packet from a packet acquisition device;

(b) performing one or more parsing/extraction operations on the packet to create a parser record comprising a function of selected portions of the packet;

(c) looking up a flow-entry database comprising none or more flow-entries for previously encountered conversational flows, the looking up using at least some of the selected packet portions and determining if the packet is of an existing flow, the lookup being via a cache;

(d) if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

(e) if the packet is of a new flow, storing a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the parsing/extraction operations depend on one or more of the protocols to which the packet conforms.

~~30~~31. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein classifying the packet as belonging to the found existing flow includes updating the flow-entry of the existing flow.

~~31~~32. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein the function of the selected portions of the packet forms a signature that includes the selected packet portions and that can identify future ~~packers~~ packets, wherein the lookup operation uses the signature and wherein the identifying information stored in the new or updated flow-entry is a signature for identifying future packets.

~~32~~33. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein the looking up of the flow-entry database uses a hash of the selected packet portions.

~~33~~34. (Currently amended) A method according to ~~claim 29~~ claim 30, wherein step (d) includes if the packet is of an existing flow, obtaining the last encountered state of the flow and performing any state operations specified for the state of the flow starting from the last encountered state of the flow; and wherein step (e) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow.

REMARKS

Claims 1–6 and 21–33 (including two claims numbered 21 prior to this amendment) are the claims of record of the application. A response to an office action was filed 10 February 2004.

The examiner has indicated to the undersigned that there were two claim 21s in the listing of claims in the response filed 10 February 2004.

The present amendment corrects several typographical errors found in both the original application and the previous amendment filed on 10 February 2004. The present amendment assumes that the previous amendment has been entered.

The undersigned discovered the previous amendment incorrectly annotated claims 2–6 as "previously presented" instead of being annotated as "original." The present amendment correctly annotates the claims.

The present amendment corrects the typographical error in the previous amendment of there being two claim 21s. The second instance of claim 21 has been renumbered claim 22, and previous claims 22–33 have been renumbered to claims 23–34, respectively. In addition, newly numbered claims 22–24, 26–29, 31–34 have been amended to depend on the appropriate newly numbered claims.

Claim 24 of the previous amendment was erroneously dependent on claim 33. The present amendment corrects this typographical error-newly numbered claim 25 depends on newly numbered claim 24.

Minor typographical errors were found claims 4, 21, and newly-numbered 32. The present amendment corrects these typographical errors.

No new matter has been added by this amendment.

The Applicants believe that the remaining claims are allowable. Action to that end is respectfully requested.

If the Examiner has any questions or comments that would advance the prosecution and allowance of this application, an email message to the undersigned at dov@inventek.com, or a telephone call to the undersigned at +1-510-547-3378 is requested.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985
Email: dov@inventek.com

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

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: 2662 Examiner: Alan Nguyen

TRANSMITTAL: SUPPLEMENTARY AMENDMENT

P.O. Box No Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a supplementary amendment for the above referenced application.

This application has:

_____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

__X__ No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Feb 20, 2004 Signed: [signature]

Name: Dov Rosenfeld, Reg. No. 38687

S/N: 09/608266 Page 2

	TOTAL CLAIMS PREVIOUSLY PAID FOR	NEW TOTAL	NO. OF EXTRA CLAIMS	EXTRA CLAIM FEE
TOTAL CLAIMS	20	20	0	$ 0.00
INDEP. CLAIMS	3	3	0	$ 0.00
TOTAL CLAIM FEES PAYABLE:				0.00

_X_ 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 ($420)

____ two months ($930) ____ four months ($1450)

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

____ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

_X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.

_X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

Feb 20, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue,Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

Our Docket/Ref. No.: APL-001-4 Patent 2662

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: 2662 Examiner: Alan Nguyen RECEIVED MAR 1 2 2004 Technology Center 2600

OIPE JC136 MAR 0 8 2004 PATENT & TRADEMARK OFFICE

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

X 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.

X A payment for petition fees.

X Return postcard.

X 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: March 4, 2004

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, P.O. Box 1450, Alexandria, VA 22313-1450.

Date of Deposit: March 4, 2004 Signature: Amy Drury
Amy Drury

Our Docket/Ref. No.: APPT-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: 2662

Examiner: Alan Nguyen

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

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)

_X_ under 37 CFR 1.97(c) together with either a:
- ___ Certification under 37 CFR 1.97(e), or
- _X_ 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)

_X_ 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, P.O. Box 1450, Alexandria, VA 22313-1450.

Date of Deposit: March 4, 2004 Signature: [signature]
Amy Drury

X (Cited in a related case) Each item of information contained in this information disclosure statement was first cited in a communication from the U.S. Patent and Trademark Office in a related application. The present application is related to such other applications by claiming priority of the same U.S. Provisional patent 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: March 4, 2004

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

Et al. FORM - 1449 SHEET 1 OF 1.

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO.	SERIAL NO.
APPT-001-4	09/608266
APPLICANT	
Sarkissian et al.	
FILING DATE	GROUP
30 Jun 2000	2662

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
[initials]	AA	6,625,657 B1	Sep. 23, 2003	Bullard	709	237	Mar. 25, 1999
[initials]	AB	6,330,226 B1	Dec. 11, 2001	Chapman et al.	370	232	Jan. 27, 1998
[initials]	AC	6,651,099 B1	Nov. 18, 2003	Dietz et al.	709	224	Jun. 30, 2000
[initials]	AD	6,424,624 B1	Jul. 23, 2002	Galand et al.	370	231	Oct. 7, 1998
[initials]	AE	6,279,113 B1	Aug. 21, 2001	Vaidya	713	201	Jun. 4, 1998
[initials]	AF	6,363,056 B1	Mar. 26, 2002	Beigi et al.	370	252	Jul. 15, 1998
[initials]	AG	6,115,393	Sep. 5, 2000	Engel et al.	370	469	Jul. 21, 1995
[initials]	AH	4,972,453	Nov. 20, 1990	Daniel, III et al.	379	10	Feb. 28, 1989
[initials]	AI	5,535,338	Jul. 9, 1996	Krause et al.	395	200.20	May 30, 1995
[initials]	AJ	5,802,054	Sep. 1, 1998	Bellenger	370	401	Aug. 16, 1996
[initials]	AK	5,720,032	Feb. 17, 1998	Picazo, Jr. et al.	395	200.2	Jan. 28, 1997

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANSLATION YES	TRANSLATION NO
	AL							
	AM							

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

[initials]	AN	R. Periakaruppam and E. Nemeth. "GTrace-A Graphical Traceroute Tool." 1999 Usenix LISA. Available on www.caida.org, URL: http://www.caida.org/outreach/papers/1999/GTrace/GTrace.pdf
[initials]	AO	W. Stallings. "Packet Filtering in the SNMP Remote Monitor." November 1994. Available on www.ddj.com, URL: http://www.ddj.com/documents/s=1013/ddj9411h/9411h.htm

EXAMINER	DATE CONSIDERED
[signature]	4/14/04

*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.

Our Docket/Ref. No.: APK_001-4

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian et al.	
Serial No.: 09/608266	Group Art Unit: 2662
Filed: June 30, 2000	Examiner: Alan Nguyen
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

X 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.

X A payment for petition fees.

X Return postcard.

X 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: March 4, 2004

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, P.O. Box 1450, Alexandria, VA 22313-1450.

Date of Deposit: March 4, 2004 Signature: Amy Drury

Notice of Allowability	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--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to 3/11/04.
2. ☒ The allowed claim(s) is/are 1-6 and 21-34, now renumbered 1-20, respectively.
3. ☐ The drawings filed on _____ are accepted by the Examiner.
4. ☐ 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 the:
   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)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

5. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.
6. ☒ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☒ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☒ to Paper No./Mail Date 6.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

7. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☒ Information Disclosure Statements (PTO-1449 or PTO/SB/08), Paper No./Mail Date 9
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application (PTO-152)
6. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
7. ☒ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DETAILED ACTION

EXAMINER'S AMENDMENT

1. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Dov Rosenfeld on April 15, 2004.

The application has been amended as follows:

In the specification:

On **page 1** line 8, -- U.S. patents and -- has been inserted between the words "following" and "U.S."

On line 11, "Application Serial No. ___/____" has been replaced with -- No. 6,651,099 --.

On lines 12 and 13, "filed June 30, 2000, Attorney/Agent Reference Number APPT-001-1," has been deleted.

On line 15, "Application Serial No. ___/____" has been replaced with -- No. 6,665,725 --.

On lines 17 and 18, "filed June 30, 2000, Attorney/Agent Reference Number APPT-001-2," has been deleted.

On line 20, "___/____" has been replaced with -- 09/608,126 --.

On lines 22 and 23, “filed June 30, 2000, Attorney/Agent Reference Number APPT-001-3,” has been deleted.

On line 24, “___/____” has been replaced with -- 09/608,267 --.

On lines 26 and 27, “filed June 30, 2000, Attomey/Agent Reference Number APPT-001-5,” has been deleted.

On **page 2** line 21, “application ___/____” has been replaced with -- No. 6,651,099 --.

On line 23, “Attorney/Agent Reference Number APPT-001-1,” has been deleted.

On **page 4** line 12, “Which” has been replaced with -- The --.

On line 13, “element a ” has been replaced with -- element which a --.

In the abstract:

On **page 64** line 2, “comprising” has been replaced with -- includes --.

On line 5, “including” has been deleted.

On line 7, “Which cache memory element” has been replaced with -- The cache memory element which --.

In the claims:

In **claim 1** line 2, “packets” has been replaced with -- packet --.

In **claim 21** line 2, “packets” has been replaced with -- packet --.

Allowable Subject Matter

2. Claims 1-20 are allowed.

3. The following is an examiner's statement of reasons for allowance:

Claims 1, 7, and 16 are allowable over the prior art of record since the cited references taken individually or in combination fails to particularly disclose **a packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to one or more protocols, the monitor having a packet acquisition device to receive packets, an input buffer to accept packets, a parser subsystem to extract selected portions of the accepted packet, a memory for storing the flow-entries, a lookup engine configured to lookup whether the particular packet whose parser record is output by the parser subsystem has a matching flow-entry, a cache subsystem for access of a set of flow-entries, a flow insertion engine to create a flow-entry, the lookup engine configured such that if the packet is of an existing flow, it is monitored as so, and if it is a new flow, the insertion engine stores a new flow entry.** It is noted that the closest prior art, Chang (US Patent 4,458,310) discloses a cache memory subsystem that utilizes the use of flow entries, but fails to show that the ability to distinguish conversational data flow.

Conclusion

4. 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 9am-6pm ET

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ricky Ngo can be reached on 703-305-4798. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9314.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

AVN
April 14, 2004

RICKY NGO
PRIMARY EXAMINER

UNITED STATES PATENT AND TRADEMARK OFFICE

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

NOTICE OF ALLOWANCE AND FEE(S) DUE

7590 04/20/2004

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
NGUYEN, ALAN V	
ART UNIT	**PAPER NUMBER**
2662	10

DATE MAILED: 04/20/2004

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

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

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	07/20/2004

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status is changed, pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above and notify the United States Patent and Trademark Office of the change in status, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check the box below and enclose the PUBLICATION FEE and 1/2 the ISSUE FEE shown above.

☐ Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PTOL-85 (Rev. 11/03) Approved for use through 04/30/2004.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

7590 04/20/2004

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

(Depositor's name)

(Signature)

(Date)

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

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

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	07/20/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
NGUYEN, ALAN V	2662	370-392000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

- ❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
- ❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1____________________
2____________________
3____________________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent); ❑ individual ❑ corporation or other private group entity ❑ government

4a. The following fee(s) are enclosed:

- ❑ Issue Fee
- ❑ Publication Fee
- ❑ Advance Order - # of Copies ____________

4b. Payment of Fee(s):

- ❑ A check in the amount of the fee(s) is enclosed.
- ❑ Payment by credit card. Form PTO-2038 is attached.
- ❑ The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number ____________ (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) (Date)

NOTE; The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

TRANSMIT THIS FORM WITH FEE(S)

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O 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 04/20/2004

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
NGUYEN, ALAN V	
ART UNIT	**PAPER NUMBER**
2662	

DATE MAILED: 04/20/2004

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 652 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 652 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) system (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (703) 305-1383. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.

PTOL-85 (Rev. 11/03) Approved for use through 04/30/2004.

OIPE JUN 04 2004 PATENT & TRADEMARK OFFICE

Our Ref./Docket No: APPT-00[illegible] Patent 98

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.* Application No.: 09/608,266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Group Art Unit: 2662 Examiner: Alan V. Nguyen #11/ML

SUBMISSION OF (CORRECTED) FORMAL DRAWINGS

Mail Stop Issue Fee
The Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir or Madam:

Attached please find new formal drawings corrected as necessary, to be made of record for the above-identified application. These drawings have been corrected per the objections raised by the Official Draftsperson. In particular, the left margin has been adjusted on FIGS. 3, 14 and 21. No new matter is being added. A copy of the Notice of Draftsperson's Patent Drawing Review has been attached. Also attached are sheets of annotated figure(s), each titled "Annotated Marked-up Drawings". Replacement figures that show the desired changes are attached, and each attached sheet of such replacement figure(s) is titled "Replacement Sheet".

If there are any issues remaining for the drawings, a telephone call or email to the undersigned at +1-510-547-3378 or dov@inventek.com is requested.

Respectfully Submitted,

June 1, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. +1-510-547-3378. Fax: +1-510-291-2985. Email: dov@inventek.com

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being mailed as U.S. First Class Mail addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: June 1, 2004 Signed: ________

Name: Amy Drury

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

OIPE JC151
JUN 04 2004
PATENT & TRADEMARK OFFICE

Application Number	09/608
Filing Date	30 Jun 2000
First Named Inventor	Sarkissian, Haig A.
Group Art Unit	2662
Examiner Name	Alan V. Nguyen
Attorney Docket Number	APPT-001-4

ENCLOSURES *(check all that apply)*

☐ Fee Transmittal Form	☐ Assignment Papers *(for an Application)*	☐ After Allowance Communication to Group
☒ Fee Attached	☒ Drawing(s)	☐ Appeal Communication to Board of Appeals and Interferences
☐ Preliminary Amendment	☐ Licensing-related Papers	☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
☐ ☐ After Final	☐ Petition Routing Slip (PTO/SB/69) and Accompanying Petition	☐ Proprietary Information
☐ ☐ Affidavits/declaration(s)	☐ To Convert a Provisional Application	☐ Status Letter
☐ Extension of Time Request	☐ Power of Attorney, Revocation Change of Correspondence Address	☒ Additional Enclosure(s) *(please identify below)*:
☐ Express Abandonment Request	☐ Terminal Disclaimer	☒ Issue Fee(s) Transmittal (form PTOL-85)
☐ Information Disclosure Statement	☐ Small Entity Statement	☒ A Transmittal Letter and Copy
☐ Certified Copy of Priority Document(s)	☐ Request of Refund	☒ Return Postcard
☐ Response to Missing Parts/ Incomplete Application	Remarks	
☐ Response to Missing Parts under 37 CFR 1.52 or 1.53		

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm or Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	X [signature]
Date	June 4, 2004

ADDRESS FOR CORRESPONDENCE

Firm or Individual name	Dov Rosenfeld 5507 College Avenue, Suite 2, Oakland, CA 94618, Tel: +1-510-547-3378

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence is being facsimile transmitted with the United States Patent and Trademark Office at Telephone number 703-872-9306 addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: June 1, 2004

Type or printed name	Amy Drury		
Signature	[signature]	Date	June 1, 2004

Annotated Marked-up Drawings

09/608266

3/21

2.5cm Margin

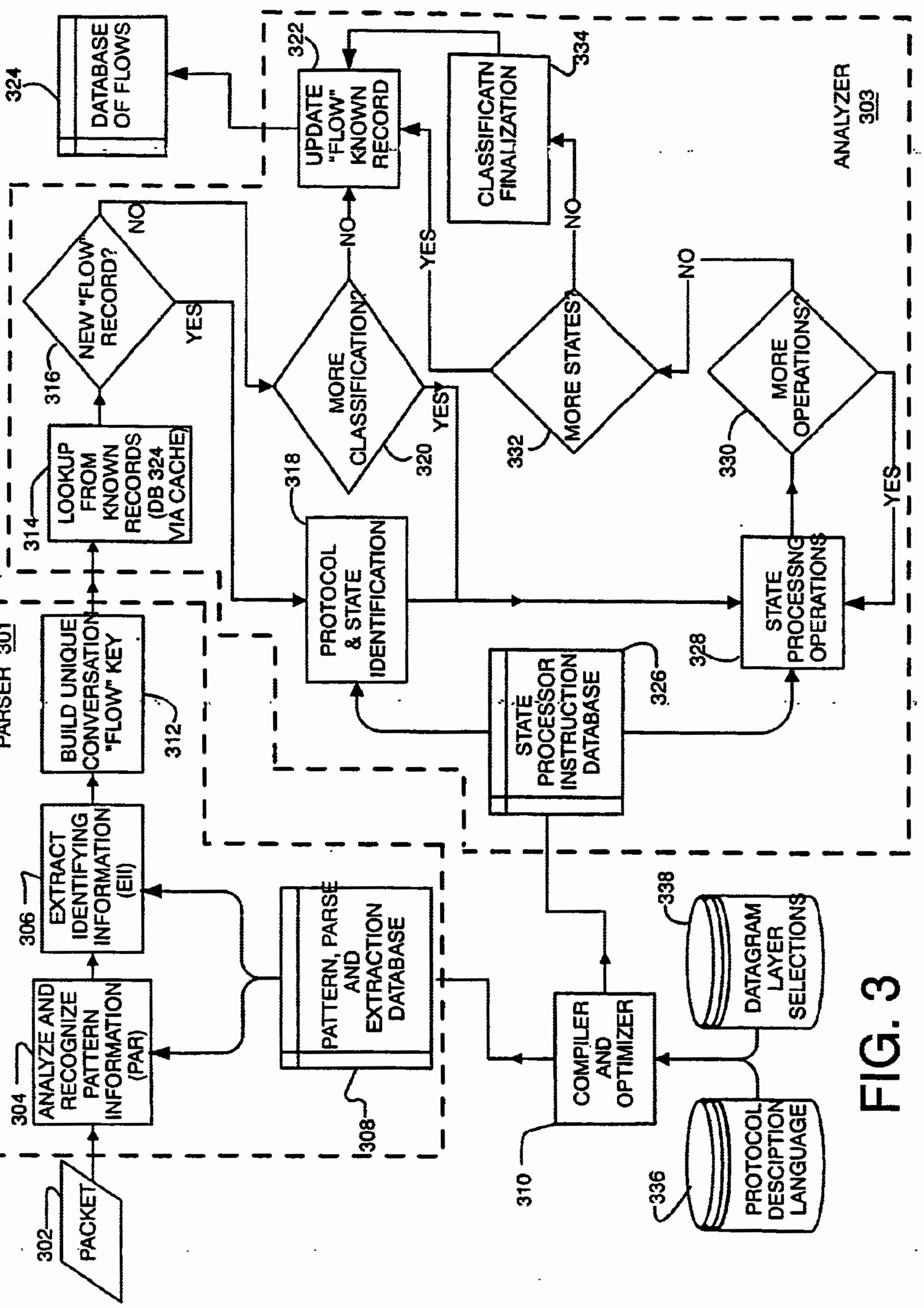


FIG. 3

Replacement Sheet

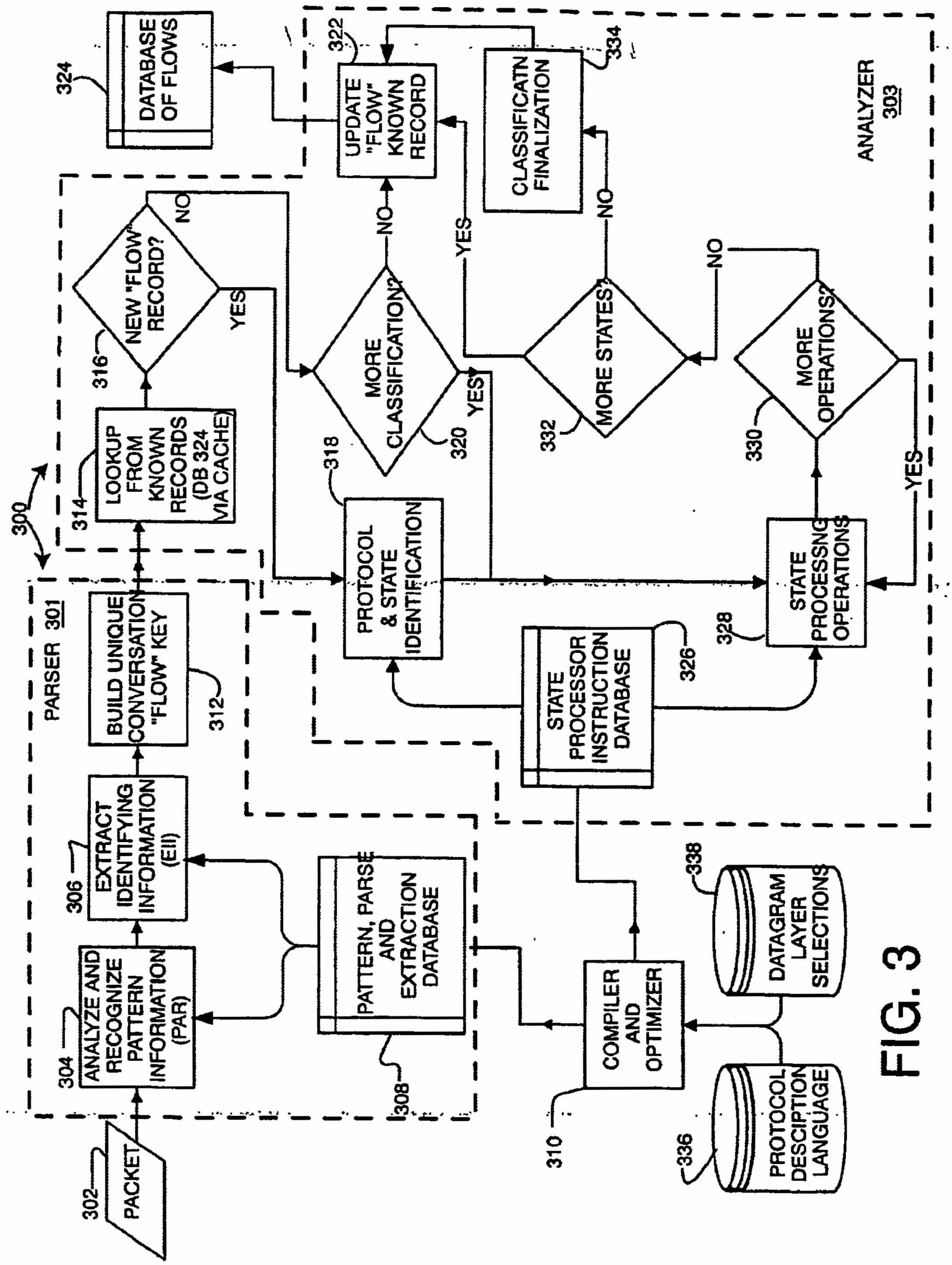


FIG. 3

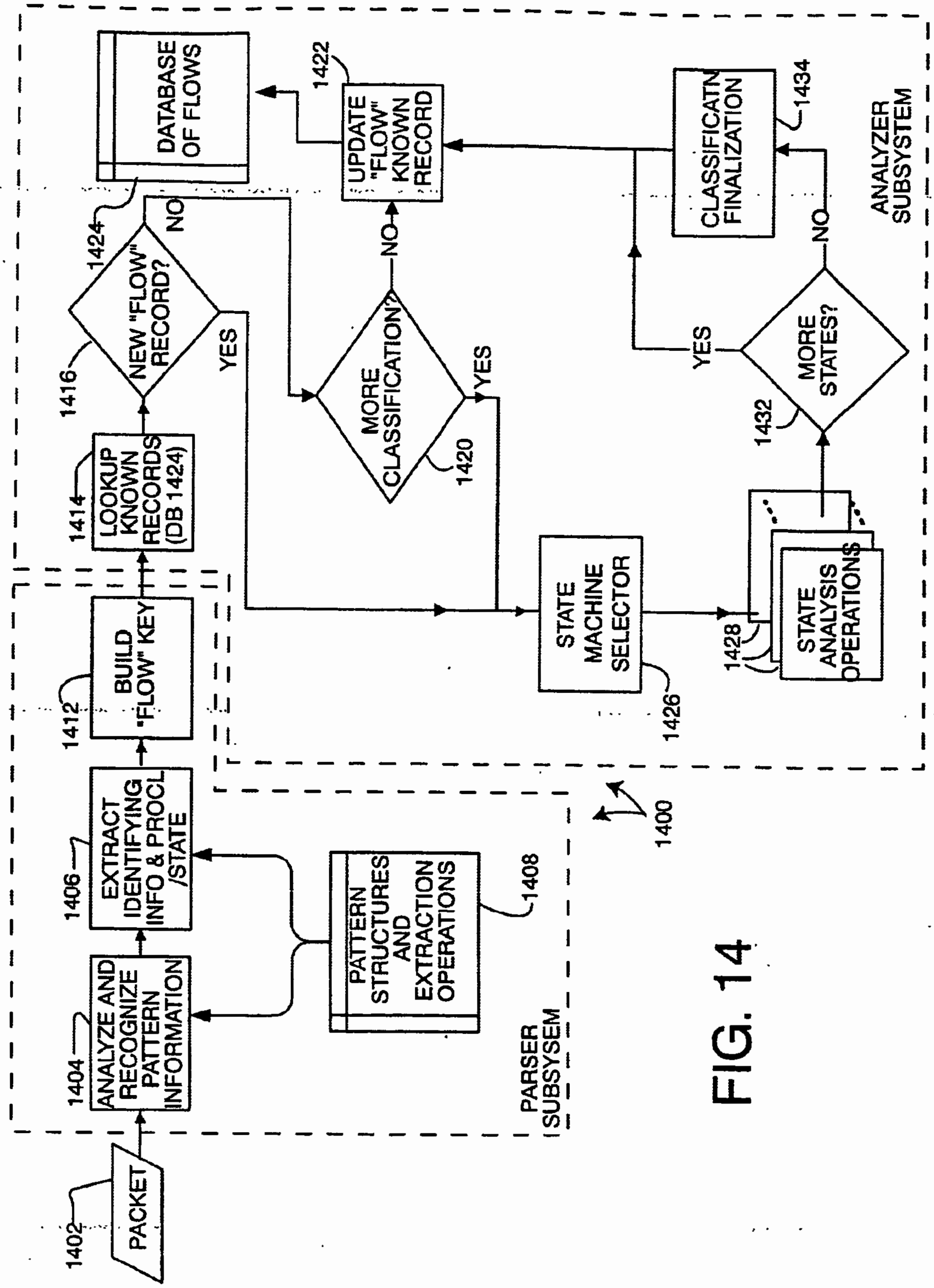

PACKET
1402
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1404
EXTRACT IDENTIFYING INFO & PROCL /STATE
1406
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
BUILD "FLOW" KEY
1412
LOOKUP KNOWN RECORDS (DB 1424)
1414
NEW "FLOW" RECORD?
1416
YES
NO
DATABASE OF FLOWS
1424
MORE CLASSIFICATION?
1420
YES
NO
UPDATE "FLOW" KNOWN RECORD
1422
STATE MACHINE SELECTOR
1426
STATE ANALYSIS OPERATIONS
1428
MORE STATES?
1432
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM
1400

FIG. 14

Annotated Marked-up Drawings

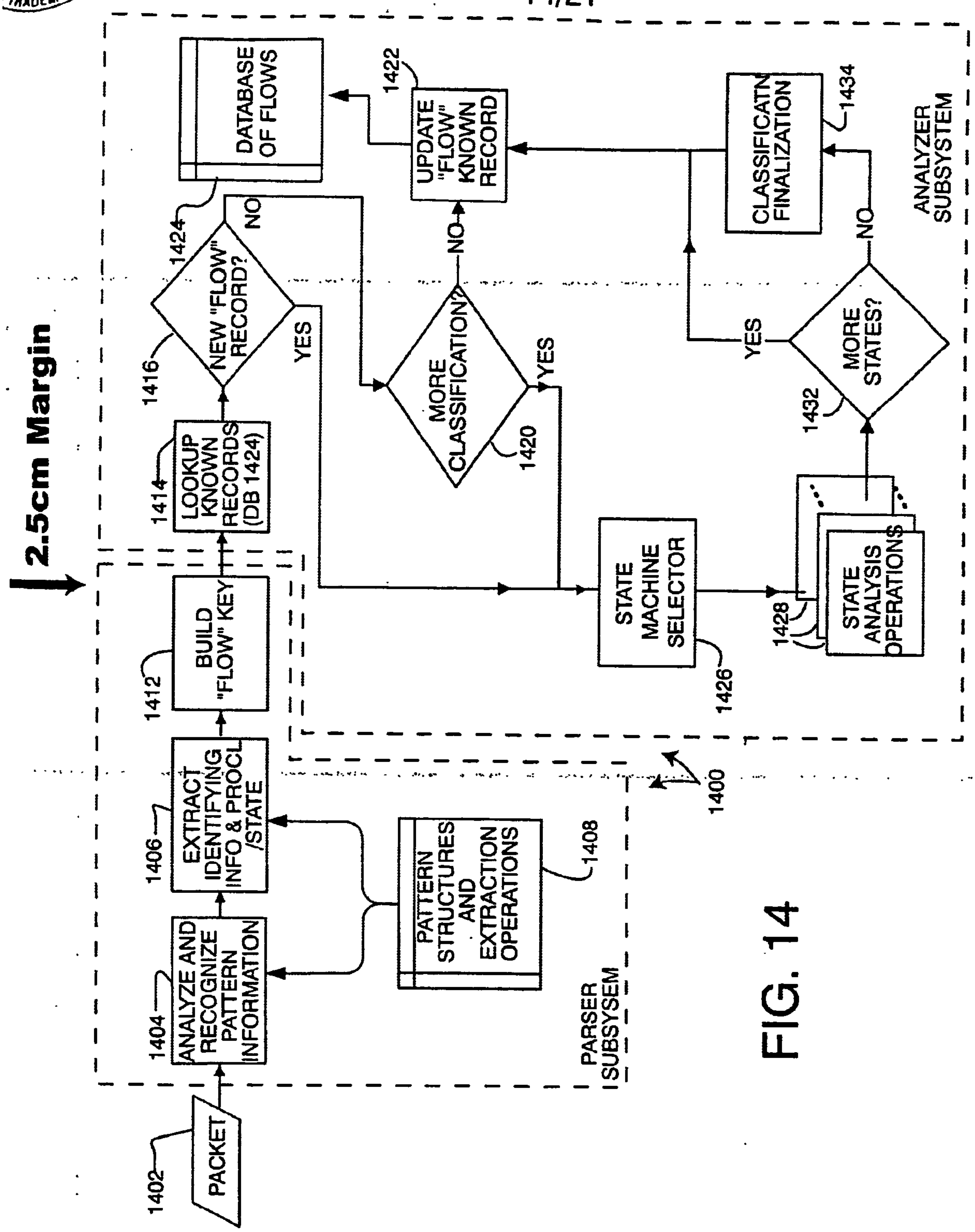


FIG. 14

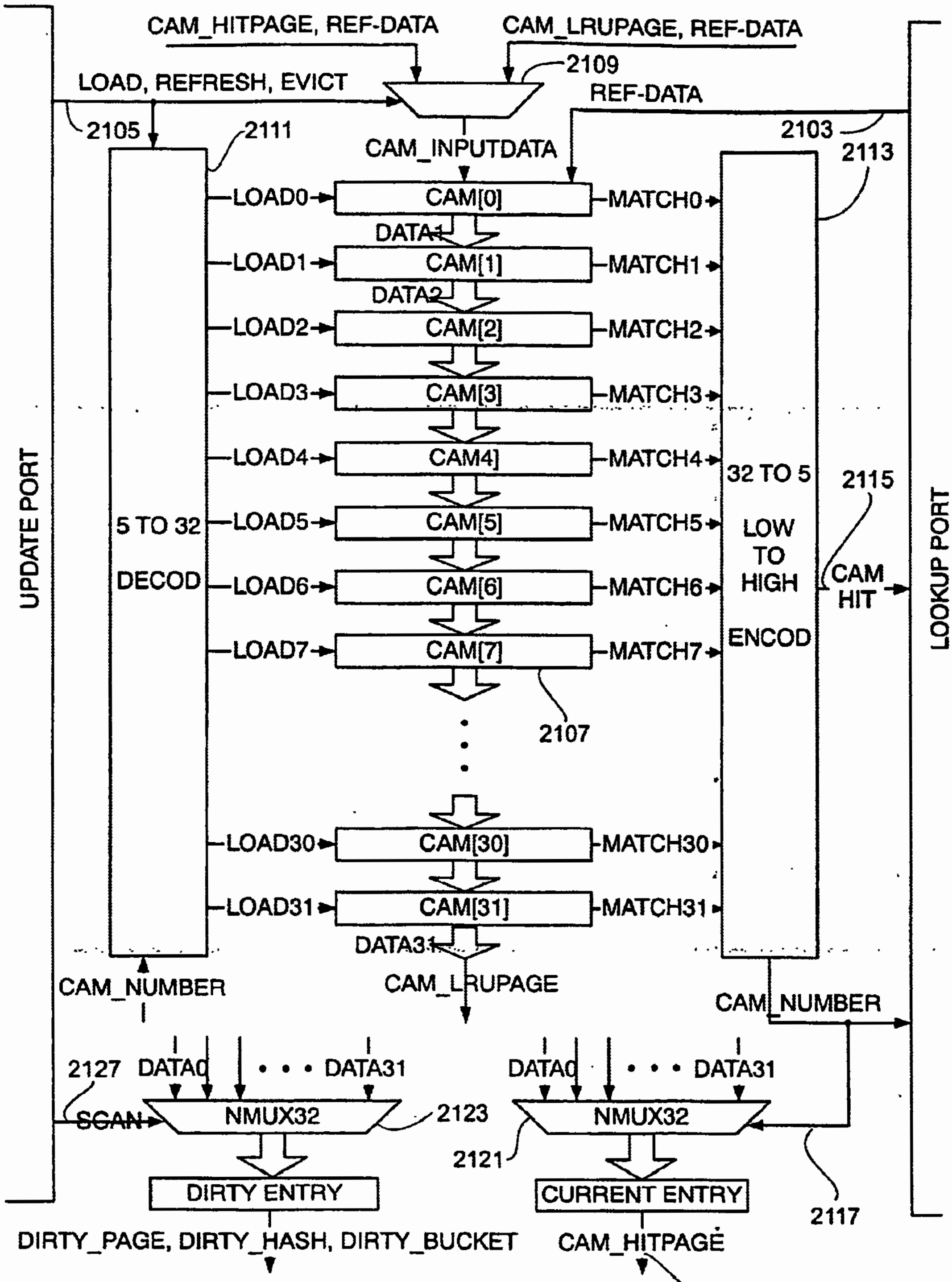

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
2109
LOAD, REFRESH, EVICT
REF-DATA
2105
2111
CAM_INPUTDATA
2103
2113
LOAD0
CAM[0]
MATCH0
DATA1
LOAD1
CAM[1]
MATCH1
DATA2
LOAD2
CAM[2]
MATCH2
LOAD3
CAM[3]
MATCH3
LOAD4
CAM4]
MATCH4
LOAD5
CAM[5]
MATCH5
LOAD6
CAM[6]
MATCH6
LOAD7
CAM[7]
MATCH7
UPDATE PORT
5 TO 32 DECOD
32 TO 5 LOW TO HIGH ENCOD
2115
CAM HIT
LOOKUP PORT
2107
LOAD30
CAM[30]
MATCH30
LOAD31
CAM[31]
MATCH31
DATA31
CAM_NUMBER
CAM_LRUPAGE
CAM_NUMBER
2127
DATA0
DATA31
SCAN
NMUX32
2123
DATA0
DATA31
NMUX32
2121
DIRTY ENTRY
CURRENT ENTRY
2117
DIRTY_PAGE, DIRTY_HASH, DIRTY_BUCKET
CAM_HITPAGE
2119

FIG. 21

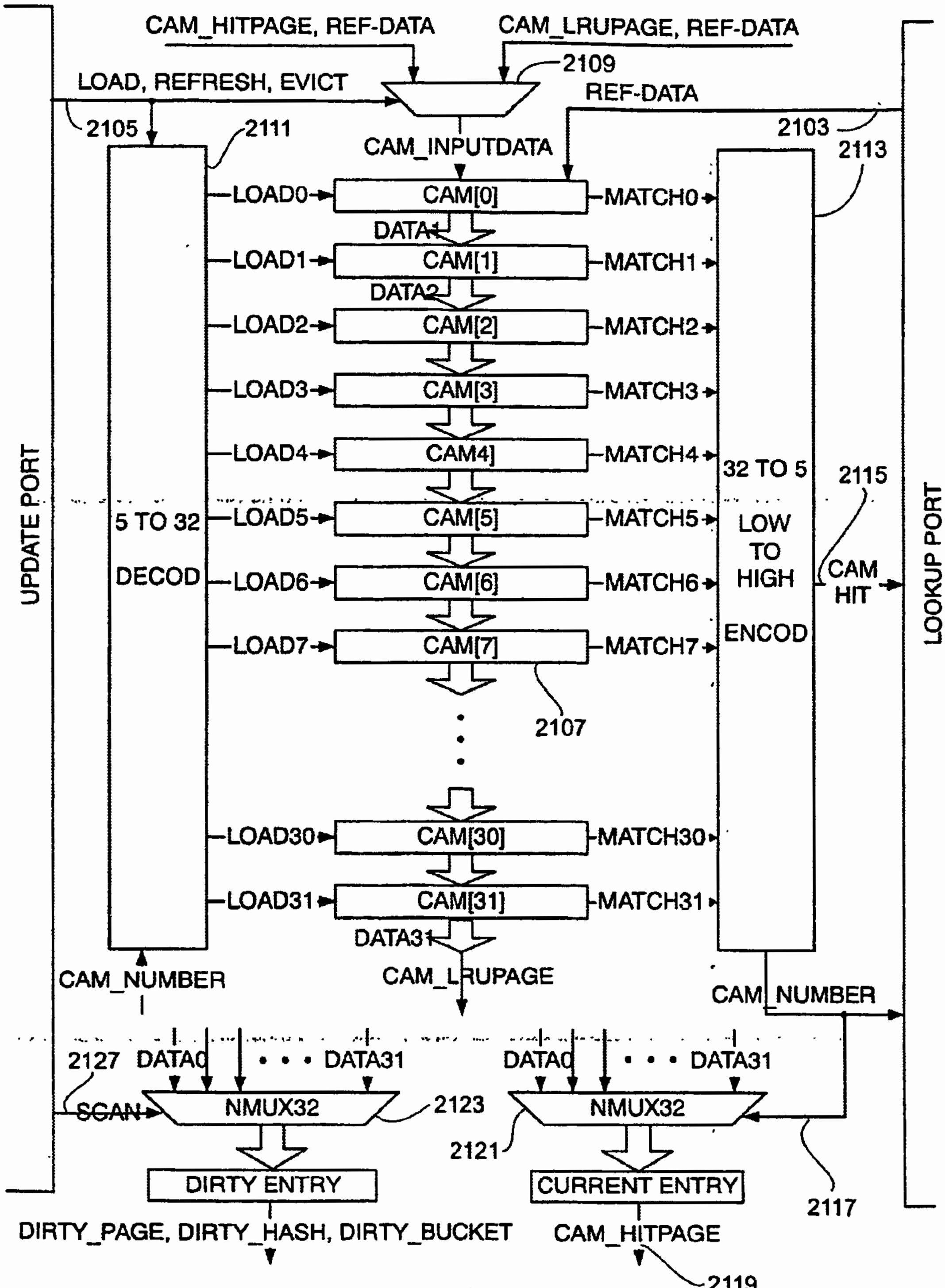

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
LOAD, REFRESH, EVICT
2109
REF-DATA
2105
2111
CAM_INPUTDATA
2103
2113
UPDATE PORT
5 TO 32
DECOD
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
LOAD30
LOAD31
CAM[0]
CAM[1]
CAM[2]
CAM[3]
CAM4]
CAM[5]
CAM[6]
CAM[7]
CAM[30]
CAM[31]
DATA1
DATA2
DATA31
MATCH0
MATCH1
MATCH2
MATCH3
MATCH4
MATCH5
MATCH6
MATCH7
MATCH30
MATCH31
32 TO 5
LOW
TO
HIGH
ENCOD
2115
CAM
HIT
LOOKUP PORT
2107
CAM_NUMBER
CAM_LRUPAGE
CAM_NUMBER
2127
DATA0
DATA31
SCAN
NMUX32
2123
DATA0
DATA31
NMUX32
2121
DIRTY ENTRY
CURRENT ENTRY
2117
DIRTY_PAGE, DIRTY_HASH, DIRTY_BUCKET
CAM_HITPAGE
2119

FIG. 21

Form PTO-948 (Rev. 06/03)
Application No. 09/608266

U.S. DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

The drawing(s) filed (insert date) 6/30/00 are:

A. _____ approved by the Draftsperson under 37 CFR 1.84 or 1.152.

B. X objected to by the Draftsperson under 37 CFR 1.84 or 1.152 for the reasons indicated below. Corrected drawings are required.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink or Color (3 sets required).
___ Color drawings are not acceptable until petition is granted. Fig(s)____
___ Pencil and non black ink not permitted. Fig(s)____

2. PHOTOGRAPHS. 37 CFR 1.84(b)
___ One (1) full-tone set is required. Fig(s)____
___ Photographs may not be mounted. 37 CFR 1.84(e)
___ Photographs must meet paper size requirements of 37 CFR 1.84(f). Fig(s)____
___ Poor quality (half-tone). Fig(s)____

3. TYPE OF PAPER. 37 CFR 1.84(e)
___ Paper not flexible, strong, white, and durable. Fig(s)____
___ Erasures, alterations, overwritings, interlineations, folds, copy machine marks not accepted. Fig(s)____

4. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
21.0 cm by 29.7 cm (DIN size A4) or
21.6 cm by 27.9 cm (8 1/2x 11 inches)
___ All drawing sheets not the same size. Sheet(s)____
___ Drawings sheets not an acceptable size. Fig(s)____

5. MARGINS. 37 CFR 1.84(g): Acceptable margins:
Top 2.5 cm Left 2.5 cm Right 1.5 cm Bottom 1.0 cm
___ Margins not acceptable. Fig(s) 3, 14, 21
___ Top (T) X Left (L)
___ Right (R) ___ Bottom (B)

6. VIEWS. 37 CFR 1.84(h)
REMINDER: Specification may require revision to correspond to drawing changes, e.g., if Fig. 1 is changed to Fig. 1A, Fig 1B and Fig. 1C, etc., the specification, at the Brief Description of the Drawings, must likewise be changed.
___ Views not labeled separately or properly. Fig(s)____

7. SECTIONAL VIEWS. 37 CFR 1.84(h)(3)
___ Sectional designation should be noted with Arabic or Roman numbers. Fig(s)____

8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)____

9. SCALE. 37 CFR 1.84(k)
___ Scale not large enough to show mechanism without crowding when drawing is reduced in size to two-thirds in reproduction Fig(s)____

10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
___ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (poor line quality). Fig(s)____

11. SHADING. 37 CFR 1.84(m)
___ Solid black areas pale. Fig(s)____
___ Solid black shading not permitted. Fig(s)____

12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
___ Numbers and reference characters not plain and legible. Fig(s)____
___ Figure legends are poor. Fig(s)____
___ Numbers and reference characters not oriented in the same direction as the view. 37 CFR 1.84(p)(1) Fig(s)____
___ English alphabet not used 37 CFR 1.84(p)(2) Fig(s)____
___ Numbers, letters and reference characters must be at least 32 cm (1/8 inch) in height. 37 CFR 1.84(p)(3). Fig(s)____

13. LEAD LINES. 37 CFR 1.84(q)
___ Lead lines missing. Fig(s)____

14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
___ Sheets not numbered consecutively, and in Arabic numbers beginning with number 1. Sheet(s)____

15. NUMBERING OF VIEWS. 37 CFR 1.84(u)
___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)____

16. DESIGN DRAWINGS. 37 CFR 1.152
___ Surface shading shown not appropriate. Fig(s)____
___ Solid black surface shading is not permitted except when used to represent the color black as well as color contrast. Fig(s)____

COMMENTS:

Reviewer A D

Date 8/29/03

If you have questions, call (703) 305-8404.

Attachment to Paper No. 6

☆ U.S. GOVERNMENT PRINTING OFFICE: 2003-300-153

OIPE JUN 04 2004 PATENT & TRADEMARK OFFICE

Our Ref./Docket No: APPT-001 — Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.*	Group Art Unit: 2662
Application No.: 09/608,266	Examiner: Alan V. Nguyen
Filed: June 30, 2000	Notice of Allowance Mailed: April 20, 2004
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Confirmation No: 9867

SUBMISSION OF ISSUE FEE AND FORMAL DRAWINGS

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a completed "Issue Fee Transmittal" Form. Included with the form are:

_X_ A credit card payment form for the issue fee and any advance order of copies.
_X_ Corrected formal drawings (with separate letter).
_X_ Return postcard

_X_ The Commissioner is hereby authorized to charge payment of the any missing fee or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

June 1, 2004
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being mailed as U.S. First Class Mail addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on

Date: June 1, 2004

Signed:
Name: Amy Drury

OIPE JUN 0 4 2004 PATENT & TRADEMARK OFFICE

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

7590 04/20/2004

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

Amy Drury (Depositor's name)
[signature] (Signature)
June 1, 2004 (Date)

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

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

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	07/20/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
NGUYEN, ALAN V	2662	370-392000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. Use of a Customer Number is required.

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Dov Rosenfeld
2 Inventek
3

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Hi/fn, Inc.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): Los Gatos, CA

Please check the appropriate assignee category or categories (will not be printed on the patent): ☐ individual ☒ corporation or other private group entity ☐ government

4a. The following fee(s) are enclosed:

☒ Issue Fee
☐ Publication Fee
☒ Advance Order - # of Copies 10

4b. Payment of Fee(s):

☐ A check in the amount of the fee(s) is enclosed.
☒ Payment by credit card. Form PTO-2038 is attached.
☒ The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number 50-0292 (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) X [signature] (Date) June 1, 2004

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

06/08/2004 MBERHE1 00000070 09608266

01 FC:1501 1330.00 OP
02 FC:8001 30.00 OP

TRANSMIT THIS FORM WITH FEE(S)

PTOL-85 (Rev. 11/03) Approved for use through 04/30/2004. OMB 0651-0033 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

OIPE JCS1 JUN 04 2004 PATENT & TRADEMARK OFFICE

Our Ref./Docket No: APPT-001

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Sarkissian, *et al.*	Group Art Unit: 2662
Application No.: 09/608,266	Examiner: Alan V. Nguyen
Filed: June 30, 2000	Notice of Allowance Mailed: April, 20, 2004
Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Confirmation No: 9867

SUBMISSION OF ISSUE FEE

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a completed "Issue Fee Transmittal" Form. Included with the form are:

_X_ A credit card payment form for the issue fee and any advance order of copies.
_X_ Corrected formal drawings (with separate letter).
_X_ Return postcard

_X_ The Commissioner is hereby authorized to charge payment of the any missing fee or credit any overpayment to Deposit Account No. 50-0292
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

June 1, 2004
Date

[signature]
Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being mailed as U.S. First Class Mail addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: June 1, 2004

Signed: [signature]
Name: Amy Drury

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

AUG 13 2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Hi/fn, Inc. Patent No: 6,771,646 B1 Issue Date: August, 3, 2004 Application No.: 09/608,266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	Certificate AUG 17 2004 of Correction

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

_X_ Such error(s) arose through the fault of the Patent and Trademark Office. It is requested that the certificate be issued at no cost to the applicant.

However, if it is determined that the error(s) arose through the fault of applicant(s), please note that such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The Commissioner is authorized to charge Deposit Account No. 50-0292 any required fee. A duplicate of this request is attached.

_____ Such error(s) arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Such error(s) specifically:

In column 4, line 38, please change "part of the cache subsystem of the analyzer subsystem" to --part of the cache subsystem 1115 of the analyzer subsystem--.

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 Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Aug. 10, 2004 Signed: [signature]
Name: Amy Drury

20 AUG 2004

In column 5, line 28, please change "tha provides a framework" to --that provides a framework--.

In column 5, line 30, please change "for understanding the functionaly" to --understanding the functionality--.

In column 5, line 47, please change "may use a layerd model" to --may use a layered model--.

In column 6, line 58, please change "buut that" to --but that--.

In column 14, line 3, please change "or the or all the lookup tables for the is PRD" to --or all the lookup tables for the PRD--.

In column 15, line 10, please change "described in FIG. 6 FIG. 6 is a flow chart" to --described in FIG. 6. FIG. 6 is a flow chart--.

In column 28, line 34, please change "denoted "i_1" 219" to --denoted "i^1" 219--.

In column 29, line 16, please change "UDS for p_1 that" to --UDS for p^1 that--.

In column 29, line 61, please change "and source address Sand C_1," to --and source address S_1 and C_1,--.

In column 36, lines 39-41, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

In column 37, lines 61-63, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Aug. 10, 2004
Date

Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 291-2985

08/31/2004 VTOLBERT 00000004 500292 09608266
Sale Ref: 00000004 DA#: 500292 09608266
01 FC:1811 100.00 DA

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 B1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 4, line 38, please change "part of the cache subsystem of the analyzer subsystem" to -- part of the cache subsystem 1115 of the analyzer subsystem--.

In column 5, line 28, please change "tha provides a framework" to --that provides a framework--.

In column 5, line 30, please change "for understanding the functionaly" to --understanding the functionality--.

In column 5, line 47, please change "may use a layerd model" to --may use a layered model--.

In column 6, line 58, please change "buut that" to --but that--.

In column 14, line 3, please change "or the or all the lookup tables for the is PRD" to --or all the lookup tables for the PRD--.

In column 15, line 10, please change "described in FIG. 6 FIG. 6 is a flow chart" to --described in FIG. 6.

FIG. 6 is a flow chart--.

In column 28, line 34, please change "denoted "i_1" 219" to --denoted "i^1" 219--.

In column 29, line 16, please change "UDS for p_1 that" to --UDS for p^1 that--.

In column 29, line 61, please change "and source address Sand C_1," to --and source address S_1 and C_1,--.

In column 36, lines 39-41, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

In column 37, lines 61-63, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646 B1
No. of additional copies

20 AUG 2004

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 B1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 4, line 38, please change "part of the cache subsystem of the analyzer subsystem" to -- part of the cache subsystem 1115 of the analyzer subsystem--.

In column 5, line 28, please change "tha provides a framework" to --that provides a framework--.

In column 5, line 30, please change "for understanding the functionaly" to --understanding the functionality--.

In column 5, line 47, please change "may use a layerd model" to --may use a layered model--.

In column 6, line 58, please change "buut that" to --but that--.

In column 14, line 3, please change "or the or all the lookup tables for the is PRD" to --or all the lookup tables for the PRD--.

In column 15, line 10, please change "described in FIG. 6 FIG. 6 is a flow chart" to --described in FIG. 6.

FIG. 6 is a flow chart--.

In column 28, line 34, please change "denoted "i_1" 219" to --denoted "i^1" 219--.

In column 29, line 16, please change "UDS for p_1 that" to --UDS for p^1 that--.

In column 29, line 61, please change "and source address Sand C_1," to --and source address S_1 and C_1,--.

In column 36, lines 39-41, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

In column 37, lines 61-63, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646 B1
No. of additional copies

20 AUG 2004

OIPE AUG 30 2004 PATENT & TRADEMARK OFFICE

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

(0/4

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

Assignee: Hi/fn, Inc.

Patent No: 6,771,646 B1

Issue Date: August, 3, 2004

Application No.: 09/608,266

Filed: June 30, 2000

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

Certificate SEP 01 2004 of Correction

#13

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

_X_ Such error(s) arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Such error(s) specifically:

In column 37, line 54 (the 8th line of claim 6), kindly change "stack," to --set,--.

In column 38, line 56 (the 1st line of claim 12), kindly change "A method" to --A monitor--.

In column 38, line 64 (the 1st line of claim 13), kindly change "A method" to --A monitor--.

4 SHAKETAE 00000020 6771646

100.00 DA

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 Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Aug. 27, 2004

Signed: [signature]

Name: Amy Drury

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Aug. 27, 2004
Date

Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. (510)547-3378; Fax: (510)291-2985

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 B1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 37, line 54 (the 8th line of claim 6), kindly change "stack," to --set,--.

In column 38, line 56 (the 1st line of claim 12), kindly change "A method" to --A monitor--.

In column 38, line 64 (the 1st line of claim 13), kindly change "A method" to --A monitor--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646 B1
No. of additional copies

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 B1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 37, line 54 (the 8th line of claim 6), kindly change "stack," to --set,--. A

In column 38, line 56 (the 1st line of claim 12), kindly change "A method" to --A monitor--. A

In column 38, line 64 (the 1st line of claim 13), kindly change "A method" to --A monitor--. A

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646 B1
No. of additional copies

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,771,646 B1
DATED : August 3, 2004
INVENTOR(S) : Sarkissian et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 4,
Line 38, please change "part of the cache subsystem of the analyzer subsystem" to -- part of the cache subsystem 1115 of the analyzer subsystem --.

Column 5,
Line 28, please change "tha provides a framework" to -- that provides a framework --.
Line 30, please change "for understanding the functionaly" to -- understanding the functionality --.
Line 47, please change "may use a layerd model" to -- may use a layered model --.

Column 6,
Line 58, please change "buut that" to -- but that --.

Column 14,
Line 3, please change "or the or all the lookup tables for the is PRD" to -- "or all the lookup tables for the PRD --.

Column 15,
Line 10, please change "described in FIG. 6 FIG. 6 is a flow chart" to -- described in FIG. 6.
FIG. 6 is a flow chart --.

Column 28,
Line 34, please change "denoted "i_1" 219" to -- denoted "i^1" 219 --.

Column 29,
Line 16, please change "UDS for p_1 that" to -- UDS for p^1 that --.
Line 61, please change "and source address Sand C_1," to -- and source address S_1 and C_1, --.

Column 36,
Lines 39-41, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to --.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,771,646 B1
DATED : August 3, 2004
INVENTOR(S) : Sarkissian et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 37,
Lines 61-63, please change "A packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to" to -- A packet monitor for examining packets passing through a connection point on a computer network, each packet conforming to --.

Signed and Sealed this

Twenty-first Day of September, 2004

JON W. DUDAS
Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,771,646 B1
DATED : August 3, 2004
INVENTOR(S) : Sarkissian et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 37,
Line 54, kindly change "stack," to -- set, --.

Column 38,
Lines 56 and 64, kindly change "A method" to -- A monitor --.

Signed and Sealed this

Sixteenth Day of November, 2004

JON W. DUDAS
Director of the United States Patent and Trademark Office

TO: MAS **On (note date):** 9/1/04 **Pat. No.:** 6 111 646 B1

Team Leaders Initials

INFO SUPPLIED BY: ____

OAC/LDRC Initials

SECOND REQUEST (DIFFERENT CORRECITONS), SUPERSEDE OR RECONSIDERATION
(OAC OR LDRC, USE A RED PEN FOR COMPLETING INFO, ON THIS COVER SHEET) (11/2002 cbn)

Team Leader, an Office Automation Clerk may assist you by supplying data from CofC Database (Current & History), PALM, and copies from Intranet, to determine type of request (second request, supersede, and/or reconsideration) and to determine if there were any errors made in decisions and/or publishing are attributable. Team Leader, check appropriate boxes below, key record (if necessary) and forward to JCWS, to order file and assign file to an LIE, to EXPEDITE.

Team Leader, DO NOT ORDER FILE.

MRD (for request attached to this cover sheet): 8 /30 /2004 (Team Leader have LDRC, stamp same MRD on 1050s.)

File Charged to (in PALM): ________ Date Charged to Loc.: __/__/__

Information re most recent record in CofC database (Check Current & History)

MRD: 8 / 20 / 2004 Examiner (LIE's initials): SP

Date Assigned: 8 / 26 / 2004 Turned In: 8 / 27 / 2004

CofC Issued: 9 / 2 / 04 CofC Denied: __/__/__ Updated: Y / N Date: __/__/__

Patent number listed on C of C listing in OG (circle one) Y / N

CofC Issued for this record is attached to patent on Internet (circle one) Y / N

New/different correction(s) requested. Check Intranet or with RTIS. (circle one) Y / N

☐ Duplicate (same heading and corrections published/issued CofC on Intranet. (circle one) Y / N

☐ Substitute or corrected request. Locate the original request(check with JCWS and RTIS).

☒ Second Request (another) requesting new/different corrections or additional corrections. TEAM LEADER, DO NOT ORDER FILE. If necessary, call attorney/applicant for assistance in determining if new/different corrections. Team Leader, key new a record on: 10/12/04. Place and count with CofCs keyed, same week, determine and note in to upper right hand corner if "P", "R", or "RTC". Mark through any corrections on 1050, that were appropriately published; or JCWS assign to: ________

☐ Reconsideration ☐ Supersede ☐ Special CofC ☐ Erratum ☐ Expedite CofC

Team Leader, determine if a Request for a Corrected CofC (Supersede) or Reconsideration, due to error in decisions or keying, attributable to (check the appropriate box, below):

☐ RTIS Keying Error ☐ LIE: ______ LIE Processing or Decision Error ☐ OFFICE Error in Entry of Document or Ex. Decision ☐ ATTY. 1.323 Consideration or Petition Required

If errors are attributable to LIE, use guidelines for appropriately notifying the LIE and recording errors (make copies supporting that the LIE made error, attach copies to this cover sheet, keeping copies for your records, and forward copies to CBN, at the end of each month).

☐ JW or OL, locate request for CofC published on: __/__/__ and return to: ______
(Circle OAC Initials)

☐ Team Leader keyed record on on: __________ ☐ Post card Printed by Tasneem
(Team Leaders, give all second requests to Tasneem, to print a post card.)

☐ JCWS, order file and assign or reassign to LIE/to: _____/LIE, see your Team Leader for assistance.

Comments/ Instructions: ______________________

☐ SEE REVERSE SIDE, FOR ADDITIONAL COMMENTS/INSTRUCTIONS) (Revised 12/15/2003 cbn) #4

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

Assignee: Hi/fn, Inc.

Patent No: 6,771,646 B1

Issue Date: August, 3, 2004

Application No.: 09/608,266

Filed: June 30, 2000

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

OIPE JC33 AUG 30 2004 PATENT & TRADEMARK OFFICE

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

_X_ Such error(s) arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Such error(s) specifically:

In column 37, line 54 (the 8th line of claim 6), kindly change "stack," to --set,--.

In column 38, line 56 (the 1st line of claim 12), kindly change "A method" to --A monitor--.

In column 38, line 64 (the 1st line of claim 13), kindly change "A method" to --A monitor--.

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 Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Aug. 27, 2004 Signed: [signature]

Name: Amy Drury

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Aug. 27, 2004
Date

Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. (510)547-3378; Fax: (510)291-2985

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Hi/fn, Inc. Patent No: 6,771,646 Issue Date: August 3, 2004 Application No.: 09/608,266 Filed: June 30, 2000 Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR	#14

REQUEST FOR CERTIFICATE OF CORRECTIONS
Change of Inventorship

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

Such error(s) arose through the fault of applicant(s). Payment for the fee is being submitted by EFS Web by credit card payment. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The correction does not involve changes which would constitute new matter or require re-examination.

Such error(s) specifically:

Kindly add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor.

The change includes a change of inventorship of an issued patent. As stated in MPEP 1412.04–I, correction of inventorship should be effected under the provisions of 35 USC 256 and 37 CFR 1.324 by filing a request for a Certificate of Correction if (A) the only change being made in the patent is to correct the inventorship; and (B) all parties are in agreement and the inventorship issue is not contested. The only change being requested is to correct the inventorship. All parties are in agreement and the inventorship issue is not contested.

Certificate of Electronic filing by EFS Web

I hereby certify that this response is being submitted via EFS Web on this day.

Date: 26 November 2007

Signed: /Dov Rosenfeld/ Reg No. 38687

Name: Dov Rosenfeld, Reg. No. 38687

Payment for the fees believed required is being submitted by EFS Web by credit card payment.

The Office is hereby authorized to charge payment of any missing fees associated with this communication or credit any overpayment to Deposit Account 50-0292.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

26 November 2007
Date

/Dov Rosenfeld/ Reg. No. 38687
Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel.(510) 547-3378; Fax: (510) 291-2985

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office. U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 PAGE 1 of 1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In the list of Inventors, add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646
No. of additional copies

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U S DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,771,646 PAGE 1 of 1

DATED : August 3, 2004

INVENTOR(S) : Sarkissian, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In the list of Inventors, add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,771,646
No. of additional copies

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Exar Corporation Patent No: 6,771,646 Issue Date: August 3, 2004 Application No.: 09/608,266 Filed: June 30, 2000	Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

REQUEST TO CORRECT INVENTORSHIP

Commissioner for Patent
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Applicant hereby petitions correcting inventorship for the above referenced issued patent. A request for a Certificate of Corrections is included herewith.

Kindly add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor.

Such error(s) in inventorship arose through the fault of applicant(s). Each such error is of clerical nature or minor nature and occurred in good faith and without any deceptive intention on the part of any one of the applicant(s), assignee(s), or the undersigned. The correction does not involve changes which would constitute new matter or require re-examination. Granting of this petition to correct inventorship is respectfully requested.

Included with this request are:

X A request for a Certificate of Corrections.

X A signed statement from William H. Bares, the inventor being added that the error of failing to include William H. Bares as an inventor of the patent occurred without any deceptive intent on the part of William H. Bares;

X Signed statement(s) from current named inventors that each either agrees to adding William H. Bares as inventor or has no disagreement with adding William H. Bares as inventor;

X A signed statement from Exar Corporation, the assignee of record agreeing to adding William H. Bares as inventor;

X A statement that the person signing on behalf of Exar, Inc., the assignee of record that such person is authorized to sign, per 37 CFR 3.73;

X Declaration and Power of Attorney signed by William H. Bares; and

X Payment for the fees required (submitted by credit card payment via EFS Web).

Thus, as stated in MPEP 1481.02, correction of inventorship should be effected under the provisions of 35 USC 256 and 37 CFR 1.324 as this petition is accompanied by:

(a) A statement from the person who is being added as an inventor that the inventorship error occurred without any deceptive intention on his part;

(b) A statement from the current named inventors agreeing to the change of inventorship.

(c) A statement from the assignee of the parties submitting this petition, agreeing to the change of inventorship in the patent.

X Throughout pendency of this application, the Commissioner is hereby authorized to charge payment of any missing fee(s) or credit any overpayment to Deposit Account No. 50-0292.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with the petition to correct inventorship.

Respectfully Submitted,

August 18, 2012
Date

/Dov Rosenfeld/ Reg. No. 38687
Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel.(510) 547-3378; Fax: (510) 291-2985

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Exar Corporation Patent No: 6,771,646 Issue Date: August 3, 2004 Application No.: 09/608,266 Filed: June 30, 2000	Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

REQUEST TO CORRECT INVENTORSHIP:
STATEMENT FROM ASSIGNEE OF INVENTOR BEING ADDED

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Exar Corporation, a corporation incorporated in the state of Delaware, the Assignee of record of the above-referenced patent, agrees to adding William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as the third inventor of the above referenced patent.

The undersigned is authorized to sign on behalf of Exar Corporation, the assignee.

Attached is a statement under 37 CFR 3.73(b) that the subject application is indeed assigned to Exar Corporation.

Respectfully Submitted,

Thomas R Melendrez 8/10/12

Signature DATE

Printed name: Thomas R. Melendrez
Title: General Counsel, Secretary, and EVP Business Development

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Exar Corporation Patent No: 6,771,646 Issue Date: August 3. 2004 Application No.: 09/608,266 Filed: June 30, 2000	Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

PETITION TO CORRECT INVENTORSHIP:
STATEMENT FROM INVENTOR BEING ADDED

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

With respect to the petition to correct inventorship in the above referenced patent by adding me:

William H. Bares, of 1655 Parkview Green Circle, San Jose, CA 95131, a Citizen of US

as the **third** inventor,

please note that the error of failing to include me as an inventor of the above referenced application and patent occurred without any deceptive intent on the part of me, William H. Bares. Furthermore note that the other inventors are indicating that they agree to this change of inventorship. This patent and its application has been assigned by me to Exar Corporation, the present assignee of record. Also included is a declaration signed by me and my Power of Attorney to Dov Rosenfeld to prosecute the application and Patent. Furthermore note that Exar Corporation, the assignee of record is indicating that it agrees to this change of inventorship.

Respectfully Submitted,

THIRD INVENTOR:

William H. Bares 7/30/2012

Inventor's Signature DATE

Inventor's Printed Name: William H. Bares

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel.(510) 547-3378; Fax: (510) 291-2985

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

Assignee: Hi/fn, Inc.

Patent No: 6,771,646

Issue Date: August 3, 2004

Application No.: 09/608,266

Filed: June 30, 2000

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

REQUEST TO CORRECT INVENTORSHIP: STATEMENT FROM CURRENT NAMED INVENTORS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

We, the current named inventors of the above referenced patent agree to adding William H. Bares of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as the third inventor of the above referenced patent and application.

Respectfully Submitted,

FIRST INVENTOR:

Haig A. Sarkissian (signature) 5-13-2009

Inventor's Signature DATE

Inventor's Printed Name: Haig A. Sarkissian

SECOND INVENTOR:

Inventor's Signature DATE

Inventor's Printed Name: Russell S. Dietz

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel.(510) 547-3378; Fax: (510) 291-2985

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Sarkissian, *et al.* Assignee: Exar Corporation Patent No: 6,771,646 Issue Date: August 3, 2004 Application No.: 09/608,266 Filed: June 30, 2000	Title: ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

REQUEST TO CORRECT INVENTORSHIP:
STATEMENT FROM CURRENT NAMED INVENTORS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

We, the current named inventors of the above referenced patent agree to adding 1655 Parkview Green Circle, San Jose, CA 95131, a Citizen of US as the third inventor of the above referenced patent and application.

Respectfully Submitted,

FIRST INVENTOR:

Inventor's Signature DATE

Inventor's Printed Name: Haig A. Sarkissian

SECOND INVENTOR:

Digitally signed by Dietz,Russell
DN: dc=local, dc=sfnt, dc=amer, ou=Locations, ou=US, ou=Redwood City, ou=Users, cn=Dietz,Russell, email=Russell.Dietz@safenet-inc.com
Date: 2012.08.01 10 27:07 -07'00'

Inventor's Signature DATE

Inventor's Printed Name: Russell S. Dietz

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel.(510) 547-3378; Fax: (510) 291-2985

DECLARATION, POWER OF ATTORNEY, AND AUTHORIZATION TO PERMIT ACCESS FOR UTILITY PATENT APPLICATION (37 CFR § 1.63)	*Attorney Docket No.*	APPT-001-4
	First Inventor	Haig A. Sarkissian
	Title	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR

As a below named inventor, I hereby declare that:

My residence/mailing 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 being submitted concurrently unless the following is checked or marked with an X:

X was filed on June 30, 2000 as US Application Serial No. 09/608,266

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 known to me to be material to patentability as defined in 37 CFR 1.56.

Authorization To Permit Access To Application by Participating Offices

The undersigned hereby grant(s) the USPTO authority to provide the European Patent Office (EPO), the Japan Patent Office (JPO), and any other intellectual property offices in which a foreign application claiming priority to the above-identified application is filed access to the above-identified patent application. See 37 CFR 1.14(c) and (h).

In accordance with 37 CFR 1.14(h)(3), access will be provided to a copy of the application-as-filed with respect to: 1) the above-identified application, 2) any foreign application to which the above-identified application claims priority under 35 USC 119(a)-(d) if a copy of the foreign application that satisfies the certified copy requirement of 37 CFR 1.55 has been filed in the above-identified US application, and 3) any U.S. application from which benefit is sought in the above-identified application.

In accordance with 37 CFR 1.14(c), access may be provided to information concerning the date of filing the Authorization to Permit Access to Application by Participating Offices.

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:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED 35 U.S.C. 119
			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:

APPLICATION SERIAL NUMBER	FILING DATE
60/141,903	June 30, 1999

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:

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

Send Correspondence to:	Direct Telephone Calls or Emails To:
Customer number: 21921	Dov Rosenfeld, Reg. No. 38,687 Tel: (510) 547-3378 Email: dov@inventek.com

I authorize the above-referenced attorney(s) and/or agent(s) to insert, on my behalf, the filing date and/or serial number above pertaining to this application, if not known as of the date of execution of this document.

I hereby declare under penalty of perjury under the laws of the United States of America 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.

INVENTOR SIGNATURE(S):

NAME OF FIRST INVENTOR:		___ A petition has been filed for this unsigned inventor					
Given Name (first and MI)	Haig A.			Family Name or Surname	Sarkissian		
Inventor's Signature						Date	
Residence City	Cornwall on Hudson	Residence State	New York	Residence Country	USA	Citizenship	US
Mailing Address	11 Braden Place						
City	Cornwall on Hudson	State	New York	Postcode/ Zip	12520	Country	USA

NAME OF SECOND INVENTOR:		___ A petition has been filed for this unsigned inventor					
Given Name (first and MI)	Russell S.			Family Name or Surname	Dietz		
Inventor's Signature						Date	
Residence City	San Jose	Residence State	CA	Residence Country	USA	Citizenship	US
Mailing Address	6475 Deer Hollow Drive						
City	San Jose	State	CA	Postcode/ Zip	95120-1623	Country	USA

NAME OF THIRD INVENTOR:		___ A petition has been filed for this unsigned inventor					
Given Name (first and MI)	William H.			Family Name or Surname	Bares		
Inventor's Signature	William H. Bares					Date	7/30/2012
Residence City	San Jose	Residence State	CA	Residence Country	USA	Citizenship	US
Mailing Address	1655 Parkview Green Circle						
City	San Jose	State	CA	Postcode/ Zip	95131	Country	USA

PTO/SB/96 (01-09)
Approved for use through 01/31/2009. OMB 0651-0031
U S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

STATEMENT UNDER 37 CFR 3.73(b)

Applicant/Patent Owner: First Inventor: Haig Sarkissian ; Assignee: Hifn, Inc.

Application No./Patent No.: 09/608,266 / 6,771,646 Filed/Issue Date: Filed 06-30-2000 / Issued 08-03-2004

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

Exar Corporation, a corporation
(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc.

states that it is:

1. ☒ the assignee of the entire right, title, and interest in;

2. ☐ an assignee of less than the entire right, title, and interest in
(The extent (by percentage) of its ownership interest is ______ %); or

3. ☐ the assignee of an undivided interest in the entirety of (a complete assignment from one of the joint inventors was made)

the patent application/patent identified above, by virtue of either:

A. ☐ An assignment from the inventor(s) of the patent application/patent identified above. The assignment was recorded in the United States Patent and Trademark Office at Reel ______, Frame ______, or for which a copy therefore is attached.

OR

B. ☒ A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

1. From: Sakissian and Dietz (Inventors) To: Apptitude, Inc
The document was recorded in the United States Patent and Trademark Office at
Reel 011258, Frame 0672, or for which a copy thereof is attached.

2. From: Apptitude, Inc To: Hi/Fn, Inc
The document was recorded in the United States Patent and Trademark Office at
Reel 028800, Frame 0034, or for which a copy thereof is attached.

3. From: Hi/Fn, Inc To: Exar Corporation
The document was recorded in the United States Patent and Trademark Office at
Reel 023180, Frame 0733, or for which a copy thereof is attached.

☒ Additional documents in the chain of title are listed on a supplemental sheet(s).

☐ As required by 37 CFR 3.73(b)(1)(i), the documentary evidence of the chain of title from the original owner to the assignee was, or concurrently is being, submitted for recordation pursuant to 37 CFR 3.11.

[NOTE: A separate copy (*i.e.*, a true copy of the original assignment document(s)) must be submitted to Assignment Division in accordance with 37 CFR Part 3, to record the assignment in the records of the USPTO. See MPEP 302.08]

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

/Dov Rosenfeld/#38687 — Signature

August 18, 2012 — Date

Dov Rosenfeld, Reg. No. 38687 — Printed or Typed Name

Agent to Assignee — Title

STATEMENT UNDER 37 CFR 3.73(b) -- ADDITIONAL SHEET

Applicant/Patent Owner: First Inventor: Haig Sarkissian ; Assignee: Hifn, Inc.

Application No./Patent No.: 09/608,266 / 6,771,646 Filed/Issue Date: Filed 06-30-2000 / Issued 08-03-2004

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

Exar Corporation, a corporation

(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc

states that it is:

1. ☒ the assignee of the entire right, title, and interest in;

☒ Continuing the chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

4. From: Inventor: William H. Bares To: Exar Corporation

The document was recorded in the United States Patent and Trademark Office at

Reel 028799, Frame 0658, or for which a copy thereof is attached.

×

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

/Dov Rosenfeld/#38687	August 18, 2012
Signature	Date
Dov Rosenfeld, Reg. No. 38687	Agent to Assignee
Printed or Typed Name	Title

Electronic Acknowledgement Receipt

EFS ID:	13529872
Application Number:	09608266
International Application Number:	
Confirmation Number:	9867
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Correspondence Address:	Dov Rosenfeld - 5507 College Avenue Suite 2 Oakland CA 94618 US 510-547-3378 -
Filer:	Dov Rosenfeld
Filer Authorized By:	
Attorney Docket Number:	APPT-001-4
Receipt Date:	18-AUG-2012
Filing Date:	30-JUN-2000
Time Stamp:	19:43:23
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	$100

RAM confirmation Number	7463
Deposit Account	500292
Authorized User	ROSENFELD,DOV

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

Charge any Additional Fees required under 37 C.F.R. Section 1.16 (National application filing, search, and examination fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.19 (Document supply fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.20 (Post Issuance fees)

Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Request for Certificate of Correction	APPT-001-4_CertificCorrection__2012-08-18.pdf	38502 78c5588b0d1902bffa0c090464e45f2c6ea30ffb	no	3
Warnings:					
Information:					
2	Petition for review by the Technology Center SPRE.	APPT-001-4_PetitionAdd_2012-08-18.pdf	26599 ab781f6db6a3dce5da6312e1c370583be73081c8	no	2
Warnings:					
Information:					
3	Miscellaneous Incoming Letter	APPT-001-4_Statements_signed.pdf	299711 8712b0c13e4c0b51c72175c11904caeddc50776a	no	4
Warnings:					
Information:					
4	Oath or Declaration filed	APPT-001-4_Bares_Declaration_signed.pdf	127415 08483e4e32300c360d12fd7482b78061f1776a33	no	3
Warnings:					
Information:					
5	Assignee showing of ownership per 37 CFR 3.73(b).	APPT-001-4_37CFR373_signed.pdf	703714 65827ca599424a00fc85e0200b441936bef1cdda	no	2
Warnings:					
Information:					
6	Fee Worksheet (SB06)	fee-info.pdf	30039 f077ba23a89d49e62baf948ab9baeb00dc3d6218	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	1225980		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Electronic Patent Application Fee Transmittal

Application Number:	09608266
Filing Date:	30-Jun-2000
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Filer:	Dov Rosenfeld
Attorney Docket Number:	APPT-001-4

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Certificate of correction	1811	1	100	100
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
		Total in USD ($)		100

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland CA 94618

In re Application of: HAIG SARKISSIAN et al.
Application No. 09608266
Patent No. 6771646
Filed: June 30, 2000
For: **ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR**

DECISION ON REQUEST FOR CERTIFICATE OF CORRECTIONS CHANGE OF INVENTORSHIP

This is a decision on the petition under filed August 18, 2012, to correct inventorship under 37 CFR 1.324.

The petition is **DISMISSED**.

A petition to correct inventorship under 37 C.F.R. 1.324 must be accompanied by:
(1) Where one or more persons are being added, a statement from each person who is being added as an inventor that the inventorship error occurred without any deceptive intention on his or her part;
(2) A statement from the current named inventors who have not submitted a either agreeing to the change of inventorship or stating that they have no disagreement in regard to the requested change;
(3) A statement from all assignees of the parties submitting a statement agreeing to the change of inventorship in the patent; and
(4) The fee set forth in § 1.20(b).

The petition failed to comply with the item (2) above. The statement from current inventor Russell Dietz is defective as it does not identify the name and the correct address of the new inventor to be added.

Telephone inquiries concerning this decision should be directed to Hassan Kizou at 571-272-3088. All other inquiries concerning the status of the application should be directed to Patent Application Information Retrieval (PAIR) system.

/Hassan Kizou/

Hassan Kizou
SPE, Technology Center 2400

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P O Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland CA 94618

In re Application of: HAIG SARKISSIAN et al.
Application No. 09608266
Patent No. 6771646
Filed: June 30, 2000
For: **ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR**

DECISION ON REQUEST FOR CERTIFICATE OF CORRECTIONS CHANGE OF INVENTORSHIP

This is a decision on the petition under filed August 18, 2012, to correct inventorship under 37 CFR 1.324.

The petition is **DISMISSED**.

A petition to correct inventorship under 37 C.F.R. 1.324 must be accompanied by:
(1) Where one or more persons are being added, a statement from each person who is being added as an inventor that the inventorship error occurred without any deceptive intention on his or her part;
(2) A statement from the current named inventors who have not submitted a either agreeing to the change of inventorship or stating that they have no disagreement in regard to the requested change;
(3) A statement from all assignees of the parties submitting a statement agreeing to the change of inventorship in the patent; and
(4) The fee set forth in § 1.20(b).

The petition failed to comply with the item (2) above. The statement from current inventor Russell Dietz is defective as it does not identify the name of the new inventor to be added.

Telephone inquiries concerning this decision should be directed to Hassan Kizou at 571-272-3088. All other inquiries concerning the status of the application should be directed to Patent Application Information Retrieval (PAIR) system.

/Hassan Kizou/

Hassan Kizou
SPE, Technology Center 2400

DOCKET NO.: 10354-001GEN PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of:

Haig A. Sarkissian, Russell S. Dietz	Confirmation No.: 9867
Application No.: 09/608,266	Group Art Unit: 2662
Patent No.: 6,771,646	Issue Date: August 3, 2004
Filing Date: June 30, 2000	Examiner: Alan V. Nguyen

#16

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

Commissioner for Patents
Office of Patent Publications
ATTN: Certificate of Correction Branch
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

REQUEST FOR CERTIFICATE OF CORRECTION PURSUANT TO 37 CFR § 1.322 & 37 CFR § 1.323

It is respectfully requested that a Certificate of Correction be issued for the above-identified patent. The patent has **three (3)** errors that are the fault of the applicant. Applicant's errors occurred in good faith and are of a clerical or typographical nature, or minor character, and are not believed to constitute new matter or require examination.

Enclosed herewith please find a completed Certificate of Correction form.

The fee in the amount of **$100.00** is attached.

Respectfully submitted,

Date: September 4, 2013

/Lawrence A. Aaronson/
Lawrence Aaronson
Reg. No. 38,369

Meunier Carlin & Curfman, LLC
817 W. Peachtree St., NW
Suite 500
Atlanta, GA 30308
phone: (404) 645-7713
fax: (404) 645-7707

PTO/SB/44 (09-07)
Approved for use through 08/31/2013 OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

Page 1 of 2

PATENT NO : 6,771,646

APPLICATION NO.: 09/608,266

ISSUE DATE August 3, 2004

INVENTOR(S) : Haig A. Sarkissian, Russell S. Dietz

It is certified that an error appears or errors appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS:

Column 2, lines 58 and 59, claim 1, change "to looking up being the cache subsystem" to --the looking up being via the cache subsystem--.

Column 2, lines 65, 66 and 67, claim 1, change "perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow" to --perform any state operations required for the initial state of the new flow in the case that the packet is not from an existing flow--.

Column 2, line 7, claim 7, change "to storing" to --for storing--.

MAILING ADDRESS OF SENDER (Please do not use customer number below):

Meunier Carlin & Curfman, LLC
817 W. Peachtree St., NW, Suite 500
Atlanta, GA 30308

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,771,646 B1
APPLICATION NO. : 09/608266
DATED : August 3, 2004
INVENTOR(S) : Haig A. Sarkissian and Russell S. Dietz

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS:

Column 36, lines 58 and 59, claim 1, change “to looking up being the cache subsystem” to --the looking up being via the cache subsystem--.

Column 36, lines 65, 66 and 67, claim 1, change “perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow” to --perform any state operations required for the initial state of the new flow in the case that the packet is not from an existing flow--.

Column 38, line 7, claim 7, change “to storing” to --for storing--.

Signed and Sealed this
Fifteenth Day of October, 2013

Teresa Stanek Rea
Deputy Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE
Certificate

Patent No. 6,771,646 B1 Patented: August 3, 2004

On petition requesting issuance of a certificate for correction of inventorship pursuant to 35 U.S.C. 256, it has been found that the above identified patent, through error and without any deceptive intent, improperly sets forth the inventorship.

Accordingly, it is hereby certified that the correct inventorship of this patent is: Haig A. Sarkissian, Conwall on Hudson, NY (US); Russell S. Dietz, San Jose, CA (US); and William H. Bares, San Jose, CA (US).

Signed and Sealed this Twenty-eighth Day of October 2014.

ROBERTO VELEZ
Supervisory Patent Examiner
Art Unit 2662
Technology Center 2600

Electronic Acknowledgement Receipt

EFS ID:	16761243
Application Number:	09608266
International Application Number:	
Confirmation Number:	9867
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Customer Number:	96039
Filer:	Lawrence Aaronson/Karen Carroll
Filer Authorized By:	Lawrence Aaronson
Attorney Docket Number:	
Receipt Date:	04-SEP-2013
Filing Date:	30-JUN-2000
Time Stamp:	15:25:08
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Electronic Funds Transfer
Payment was successfully received in RAM	$100
RAM confirmation Number	2251
Deposit Account	
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

1	Transmittal Letter	6771646_Request_for_Certificate_of_Correction_Tns.pdf	76243 8207d3961344c2db9ad47e96b0e31b866ee7bb22	no	1
Warnings:					
Information:					
2	Request for Certificate of Correction	6771646_PTO_SB_44_Certificate_of_Correction.pdf	165126 0ca631dfa482cd40325657f155db187f64fc7a1f	no	2
Warnings:					
Information:					
3	Fee Worksheet (SB06)	fee-info.pdf	29952 f27b25c5a435254877d65e033c35b0356e99921f	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	271321		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Electronic Patent Application Fee Transmittal

Application Number:	09608266
Filing Date:	30-Jun-2000
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Filer:	Lawrence Aaronson/Karen Carroll
Attorney Docket Number:	

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Certificate of Correction	1811	1	100	100
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
	Total in USD ($)			100

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

#7

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION TO CORRECT INVENTORSHIP UNDER 37 C.F.R. § 1.324

Dear Commissioner:

Applicant hereby petitions correcting inventorship for the above referenced issued patent. A request for a Certificate of Corrections is included herewith.

Kindly add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor.

Such error(s) in inventorship arose through the fault of applicant(s). Each such error is of clerical nature or minor nature and occurred in good faith and without any deceptive intention on the part of any one of the applicant(s), assignee(s), or the undersigned. The correction does not involve changes which would constitute new matter or require re-examination. Granting of this petition to correct inventorship is respectfully requested.

Included with this request are:

X A request for a Certificate of Corrections;

X A signed statement from William H. Bares, the inventor being added that the error of failing to include William H. Bares as an inventor of the patent occurred without any deceptive intent on the part of William H. Bares;

X Signed statement(s) from current named inventors that each either agrees to adding William H. Bares as inventor or has no disagreement with adding William H. Bares as inventor;

X A signed statement from Packet Intelligence LLC, the assignee of record agreeing to adding William H. Bares as inventor;

X A statement that the person signing on behalf of Packet Intelligence LLC the assignee of record that such person is authorized to sign, per 37 CFR 3.73; and

X Payment for the fees required (submitted by credit card payment via EFS Web).

Thus, as stated in MPEP 1481.02, correction of inventorship should be effected under the provisions of 35 USC 256 and 37 CFR 1.324 as this petition is accompanied by:

(a) A statement from the person who is being added as an inventor that the inventorship error occurred without any deceptive intention on his part;

(b) A statement from the current named inventors agreeing to the change of inventorship.

(c) A statement from the assignee of the parties submitting this petition, agreeing to the change of inventorship in the patent.

The Commissioner is hereby authorized to charge payment of any missing fee(s) or credit any overpayment to Deposit Account No. 50-5226.

The undersigned requests being contacted at (404) 645-7700 if there are any questions or clarifications, or if there are any problems with the petition to correct inventorship.

Respectfully submitted

MEUNIER CARLIN & CURFMAN, LLC

Date: August 9, 2014

/Lawrence A. Aaronson/
Lawrence A. Aaronson
Reg. No. 38,369

Customer No. 96039
docketing@mcciplaw.com
404.645.7700 Phone
404.645.7707 Fax

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE

Patent No. 6,771,646 B1 | Patented: August 3, 2004

On petition requesting issuance of a certificate for correction of inventorship pursuant to 35 U S.C. 256, it has been found that the above identified patent, through error and without any deceptive intent, improperly sets forth the inventorship.

Accordingly, it is hereby certified that the correct inventorship of this patent is: Haig A. Sarkissian, Conwall on Hudson, NY (US); Russell S. Dietz, San Jose, CA (US); William H. Bares, San Jose, CA (US)

Signed and Sealed this Twenty-eighth Day of October 2014.

Roberto Velez
Supervisory Patent Examiner
Art Unit 2662
Technology Center 2600

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE

Patent No. 6,771,646 B2
Patented: August 3, 2004

On petition requesting issuance of a certificate for correction of inventorship pursuant to 35 U.S.C. 256, it has been found that the above-identified patent, through error and without deceptive intent, improperly sets forth the inventorship. Accordingly, it is hereby certified that the correct inventorship of this patent is:

William H. Bares from San Jose, California; Haig A. Sarkissian from San Antonio, Texas; Russell S. Dietz from San Jose, California.

Roberto Velez
Supervisory Patent Examiner
Art Unit 2662
Technology Center 2600

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST FOR CERTIFICATE OF CORRECTIONS
Change of Inventorship

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

Such error(s) arose through the fault of applicant(s). Payment for the fee is being submitted by EFS Web by credit card payment. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The correction does not involve changes which would constitute new matter or require re-examination.

Such error(s) specifically:

> Kindly add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor.

The change includes a change of inventorship of an issued patent. As stated in MPEP 1412.04–I, correction of inventorship should be effected under the provisions of 35 USC 256 and 37 CFR 1.324 by filing a request for a Certificate of Correction if (A) the only change being made in the patent is to correct the inventorship; and (B) all parties are in agreement and the inventorship issue is not contested. The only change being requested is to correct the inventorship. All parties are in agreement and the inventorship issue is not contested.

This request for a Certificate of Corrections is accompanied by:

X A request for change of inventorship;

X A signed statement from William H. Bares, the inventor being added that the error of failing to include William H. Bares as an inventor of the patent occurred without any deceptive intent on the part of William H. Bares;

X Signed statement(s) from current named inventors that each either agrees to adding William H. Bares as inventor or has no disagreement with adding William H. Bares as inventor;

X A signed statement from Packet Intelligence LLC, the assignee of record agreeing to adding William H. Bares as inventor;

X A statement that the person signing on behalf of Packet Intelligence LLC, the assignee of record that such person is authorized to sign, per 37 CFR 3.73; and

X Payment for the fees required (submitted by credit card payment via EFS Web).

The Office is hereby authorized to charge payment of any missing fees associated with this communication or credit any overpayment to Deposit Account 50-5226.

The undersigned requests being contacted at (404) 645-7700 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully submitted

MEUNIER CARLIN & CURFMAN, LLC

Date: August 9, 2014

/Lawrence A. Aaronson/
Lawrence A. Aaronson
Reg. No. 38,369

Customer No. 96039
docketing@mcciplaw.com
404.645.7700 Phone
404.645.7707 Fax

PTO/SB/44 (09-07)
Approved for use through 08/31/2013. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.
(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

Page 1 of 1

PATENT NO · 6,771,646

APPLICATION NO 09/608,266

ISSUE DATE August 3, 2004

INVENTOR(S) · Sarkissian, et al.

It is certified that an error appears or errors appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the list of Inventors, add William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as third inventor

MAILING ADDRESS OF SENDER (Please do not use customer number below):

Meunier Carlin & Curfman
817 W. Peachtree St., NW, Suite 500
Atlanta, GA 30308

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST TO CORRECT INVENTORSHIP:
STATEMENT FROM ASSIGNEE OF INVENTOR BEING ADDED

Dear Commissioner:

Packet Intelligence LLC, a corporation incorporated in the state of Texas, the Assignee of record of the above-referenced patent, agrees to adding William H. Bares, of 5063 Elester Drive, San Jose, CA 95124, a Citizen of US as the third inventor of the above referenced patent.

The undersigned is authorized to sign on behalf of Packet Intelligence LLC, the assignee.

Attached is a statement under 37 CFR 3.73(b) that the subject application is indeed assigned to Packet Intelligence LLC.

Respectfully submitted,

8/1/14
Date

[signature]
Signature

Customer No. 96039
docketing@mcciplaw.com
404.645.7700 Phone
404.645.7707 Fax

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

STATEMENT UNDER 37 CFR § 3.73(c)

Packet Intelligence LLC states that it is:

☒ the assignee of the entire right, title, and interest; or

☐ an assignee of an undivided part interest

in the present patent application/patent by virtue of either:

A. ☐ An assignment from the inventor(s) of the patent application/patent identified below.

☐ 1. The assignment in the parent of the instant applications was recorded in the Patent and Trademark Office at:

☐ 2. The assignment has not yet been recorded. A copy of the assignment is attached.

OR

B. ☒ A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as shown below:

1. From: Sakissian and Dietz (Inventors) To: Apptitude, Inc

The document was recorded in the United States Patent and Trademark Office at

Reel 011258, Frame 0672.

2. From: Apptitude, Inc To: Hi/Fn, Inc

The document was recorded in the United States Patent and Trademark Office at

Reel.028800, Frame 0034.

3. From: Hi/Fn, Inc To: Exar Corporation

   The document was recorded in the United States Patent and Trademark Office at Reel 023180, Frame 0733.

4. From: William H. Bares (Inventor) To: Exar Corporation

   The document was recorded in the United States Patent and Trademark Office at Reel 028799, Frame 0658.

5. From: Exar Corporation To: Packet Intelligence LLC

   The document was recorded in the United States Patent and Trademark Office at Reel 029737, Frame 0613.

☐ Additional documents in the chain of title are listed on a supplemental sheet.

☐ Copies of assignments or other documents in the chain of title are attached.

The undersigned (whose title is supplied below) is empowered to act on behalf of the assignee.

Bradly A Brunell (signature) — Signature

8/1/14 — Date

Bradly A Brunell — Printed or Typed Name

Authorized Member — Title

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION TO CORRECT INVENTORSHIP:
STATEMENT FROM INVENTOR BEING ADDED

Dear Commissioner;

With respect to the petition to correct inventorship in the above referenced patent by adding me:

William H. Bares, of 1655 Parkview Green Circle, San Jose, CA 95131, a Citizen of US as the **third** inventor,

Please note that the error of failing to include me as an inventor of the above referenced application and patent occurred without any deceptive intent on the part of me, William H. Bares. Furthermore note that the other inventors are indicating that they agree to this change of inventorship. This patent has been assigned by me to Packet Intelligence, LLC, who is now the present assignee of record. Furthermore note that Packet Intelligence, LLC, the assignee of record is indicating that it agrees to this change of inventorship.

{Signature Continues on Next Page}

Respectfully Submitted,

THIRD INVENTOR:

William H. Bares   8/1/14

Inventor's Signature   DATE

Inventor's Printed Name: William H. Bares

Address for correspondence:
Customer No. 96039
docketing@mcciplaw.com
404.645.7700 Phone

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST TO CORRECT INVENTORSHIP:
STATEMENT FROM CURRENT NAMED INVENTORS

Dear Commissioner:

I, the current named first inventor of the above referenced patent, agree to adding William H. Bares of 1655 Parkview Green Circle, San Jose, CA, 95131, a Citizen of US, as the third inventor of the above referenced patent and application.

Respectfully Submitted,

FIRST INVENTOR:

Haig A Sarkissian   July 31, 2014

Inventor's Signature   DATE

Inventor's Printed Name: Haig A. Sarkissian

Address for correspondence:
Customer No. 96039
docketing@inceiplaw.com
404.645.7700 Phone
404.645.7707 Fax

Attorney Docket No. 10354-005US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Haig A. Sarkissian	Art Unit	:	2662
Serial No.	:	09/608,266	Examiner	:	Nguyen, Alan V.
Filed	:	June 30, 2000	Conf. No.	:	9867
Title	:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR			

Mail Stop Petition
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST TO CORRECT INVENTORSHIP:
STATEMENT FROM CURRENT NAMED INVENTORS

Dear Commissioner:

I, the current named second inventor of the above referenced patent, agree to adding William H. Bares of 1655 Parkview Green Circle, San Jose, CA, 95131, a Citizen of US, as the third inventor of the above referenced patent and application.

Respectfully Submitted,

SECOND INVENTOR:

Inventor's Signature — DATE: July 18th, 2014

Inventor's Printed Name: Russell S. Dietz

Address for correspondence:
Customer No. 96039
docketing@mcciplaw.com
404.645.7700 Phone
404.645.7707 Fax

COMBINED DECLARATION AND POWER OF ATTORNEY

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 , the specification of which:

[] is attached hereto.
[X] was filed on June 30, 2000 as Application Serial No. 09/608,266 and was amended on ________________.
[] was described and claimed in PCT International Application No. _______________ filed on _______________ and as amended under PCT Article 19 on _______________.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose all information I know to be material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim the benefit under Title 35, United States Code, §119(e)(1) of any United States provisional application(s) listed below:

U.S. Serial No.	Filing Date	Status
60/141,903	June 30, 1999	Expired

I hereby claim the benefit under Title 35, United States Code, §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, §112, I acknowledge the duty to disclose all information I know to be material to patentability as defined in Title 37, Code of Federal Regulations, §1.56(a) which became available between the filing date of the prior application and the national or PCT international filing date of this application:

U.S. Serial No.	Filing Date	Status
N/A	N/A	N/A

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

Country	Application No.	Filing Date	Priority Claimed
N/A	N/A	N/A	[] Yes [] No

I hereby appoint the following attorneys and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Lawrence A. Aaronson, Reg. No. 38,369

Direct all telephone calls to LAWRENCE A. AARONSON at telephone number 404-645-7713.

Direct all correspondence to the following:

96039
PTO Customer Number

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 patents issued thereon.

Full Name of Inventor:	HAIG SARKISSIAN		
Inventor's Signature:	______________________	Date:	____________
Residence Address:	Conwall on Hudson, NY, US		
Citizenship:	US		
Post Office Address:	11 Braden Place Conwall on Hudson, NY 12520		

Full Name of Inventor:	RUSSELL DIETZ		
Inventor's Signature:	______________________	Date:	____________
Residence Address:	San Jose, CA, US		
Citizenship:	US		
Post Office Address:	6475 Deer Hollow Drive San Jose, CA 95120		

Attorney's Docket No.: 10354-005US1

Full Name of Inventor:	WILLIAM BARES
Inventor's Signature:	William Bares — Date: 8/1/14
Residence Address:	San Jose, CA, US
Citizenship:	US
Post Office Address:	1655 Parkview Green Circle San Jose, CA, 95131

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE

Patent No. 6,771,646 B2
Patented: August 3, 2004

On petition requesting issuance of a certificate for correction of inventorship pursuant to 35 U.S.C. 256, it has been found that the above-identified patent, through error and without deceptive intent, improperly sets forth the inventorship. Accordingly, it is hereby certified that the correct inventorship of this patent is:

William H. Bares from San Jose, California; Haig A. Sarkissian from San Antonio, Texas; Russell S. Dietz from San Jose, California.

Roberto Velez
Supervisory Patent Examiner
Art Unit 2662
Technology Center 2600

PTO/SB/44 (09-07)
Approved for use through 08/31/2013. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

Page 1 of 2

PATENT NO. : 6,771,646 B1

APPLICATION NO. : 09/608,266

ISSUE DATE : August 3, 2004

INVENTOR(S) Haig A. Sarkissian, Russell S. Dietz

It is certified that an error appears or errors appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS:

Column 36, lines 58 and 59, claim 1, change "to looking up being the cache subsystem" to --the looking up being via the cache subsystem--.

Column 36, lines 65, 66 and 67, claim 1, change "perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow" to --perform any state operations required for the initial state of the new flow in the case that the packet is not from an existing flow--.

Column 39, line 7, claim 7, change "to storing" to --for storing--.

MAILING ADDRESS OF SENDER (Please do not use customer number below):

Meunier Carlin & Curfman, LLC
817 W. Peachtree St., NW, Suite 500
Atlanta, GA 30308

Electronic Acknowledgement Receipt

EFS ID:	19822864
Application Number:	09608266
International Application Number:	
Confirmation Number:	9867
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Customer Number:	96039
Filer:	Lawrence Aaronson
Filer Authorized By:	
Attorney Docket Number:	10354-005US1
Receipt Date:	09-AUG-2014
Filing Date:	30-JUN-2000
Time Stamp:	13:56:43
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Electronic Funds Transfer
Payment was successfully received in RAM	$130
RAM confirmation Number	6290
Deposit Account	
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

1	Petition for review by the Office of Petitions.	10354-005US_2014-08-09_Petition_to_Correct_Inventorship.pdf	71608 f8b6e4c5153c0ea8af5b7d20590e0f0366d4ce93	no	2
Warnings:					
Information:					
2	Request for Certificate of Correction	10354-005US1__2014_08_09_Request_for_COC_CoverSheet.pdf	106437 ec1d62809debda0236d62ea8abdb7f7bcb35f93a	no	2
Warnings:					
Information:					
3	Request for Certificate of Correction	10354-005US_2014-08-09_COC.PDF	164378 8754533b2176d29b62db92c2fd44c85167aafa10	no	2
Warnings:					
Information:					
4	Miscellaneous Incoming Letter	10354-005US_2014-08-01_Executed_Consent_of_Assignee.PDF	172294 a132ac89e592b7797cbf4e2ea6607712d8381345	no	1
Warnings:					
Information:					
5	Assignee showing of ownership per 37 CFR 3.73.	10354-005US_2014-08-01_Executed_Statement_Under_373.PDF	178790 9cb75299a22d793d7a66de26b5b13495e3cbcf48	no	2
Warnings:					
Information:					
6	Miscellaneous Incoming Letter	10354-005US1__2014_08_09_Petition_to_Correct_Inventorship_executed.PDF	108009 d5d3e86cd9c311c6df167ff2383657505df66945	no	2
Warnings:					
Information:					
7	Miscellaneous Incoming Letter	10354-005US1_2014-08-09_Request_for_Correction_of_Inventorship_Sarkissian.PDF	328652 e3980523a338e6637cf4e89f2f99c63471920ed2	no	1
Warnings:					
Information:					
8	Maintenance Fee Address Change	10354-005US1_2014-08-09_Request_for_Correction_of_Inventorship_Dietz.PDF	192916 6605e2ba4a8505876daa707d9c69ace3e1cca4f7	no	1
Warnings:					
Information:					
9	Oath or Declaration filed	10354-005US1__2094-08-09__Declaration_excuted_Bares.PDF	213957 4964fe32e41d811571482ffcc9758ff162339b87	no	3
Warnings:					
Information:					

10	Fee Worksheet (SB06)	fee-info.pdf	30317 4459761af871f08e5dc1c1fc6c5502e5a5fe2a82	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	1567358		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Electronic Patent Application Fee Transmittal

Application Number:	09608266
Filing Date:	30-Jun-2000
Title of Invention:	ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR
First Named Inventor/Applicant Name:	Haig A. Sarkissian
Filer:	Lawrence Aaronson
Attorney Docket Number:	10354-005US1

Filed as Large Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Processing Fee Correcting Inventorship	1816	1	130	130
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Miscellaneous:				
		Total in USD ($)		130

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P O Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

#18

MEUNIER CARLIN & CURFMAN, LLC
817 W. PEACHTREE STREET, NW, SUITE 500
ATLANTA, GA 30308

In re Patent No. 6,771,646
Issue Date: August 3, 2004
Appl. No: 09/608,266
Filed: June 30, 2000
For: Correction of Inventorship

This is a decision on the petition filed August 9, 2014, to correct inventorship under 37 CFR 1.324.

The petition is **GRANTED**.

The patented filed is being forwarded to Certificate of Corrections Branch for issuance of a certificate naming only the actual inventor or inventors.

/Roberto Velez/

Roberto Velez
Supervisory Patent Examiner Art Unit 2662
Technology Center 2600

CODE SHEET FOR CONTINUING DATA

Line	Code	Serial No.	Filing Date	Status	Document No.	Issue Date
104	68	60/141,903	6/30/99			
105						
106						
107						
108						
109						
110						
111						
112						
113						
114						
115						
116						
117						

Condition and Status Codes for Continuing Data

CONDITION CODE:

71	Continuation of application No.
81	which is a continuation of application No.
91	and a continuation of application No.
72	Continuation-in-part of application No.
82	which is a continuation-in-part of application No.
75	and a continuation-in-part of application No.
74	Division of application No.
84	which is a division of application No.
76	and a division of application No.
86	, said application No.
89	Application No.
90	and application No.
92	each
65	filed as application No.
66	substitute for application No.
68	Provisional application No.

STATUS CODE

01	Patent No. /
03	abandoned
04	SIR No.

NOTE I: When the code 86 and 92 are used, they must be followed by 81, 82 or 84 – condition beginning with "which is"
NOTE II: Codes 71, 72 and 74 may be used only on the first line; one of them must be used on the first line in regular continuing data. 66 or 68 may be used on the first line in Substitute or Provisional cases. Remember, however, that if there is a Provisional and other continuing data, the Provisional is always listed last.

E-6 (Revised 10/05/00)

PATENT APPLICATION FEE DETERMINATION RECORD

Effective December 29, 1999

Application or Docket Number: 09/608266

CLAIMS AS FILED - PART I

FOR	NUMBER FILED (Column 1)	NUMBER EXTRA (Column 2)	SMALL ENTITY TYPE ☐ RATE	FEE	OR	OTHER THAN SMALL ENTITY RATE	FEE
BASIC FEE				345.00	OR		690.00
TOTAL CLAIMS	20 minus 20=	*	X$ 9=		OR	X$18=	
INDEPENDENT CLAIMS	3 minus 3 =	*	X39=		OR	X78=	
MULTIPLE DEPENDENT CLAIM PRESENT			+130=		OR	+260=	
			TOTAL		OR	TOTAL	690.00

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

AMENDMENT A

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
Total	*	Minus	**	=	X$ 9=		OR	X$18=	
Independent	*	Minus	***	=	X39=		OR	X78=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
Total	*	Minus	**	=	X$ 9=		OR	X$18=	
Independent	*	Minus	***	=	X39=		OR	X78=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
Total	*	Minus	**	=	X$ 9=		OR	X$18=	
Independent	*	Minus	***	=	X39=		OR	X78=	
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
					TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropnate box in column 1.

FORM PTO-875
(Rev. 12/99)

Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE
*U.S. GPO: 2000-463-433/29044

US006301245B1

(12) **United States Patent**
Luzeski et al.

(10) Patent No.: **US 6,301,245 B1**
(45) Date of Patent: **Oct. 9, 2001**

(54) **UNIVERSAL MESSAGING SYSTEM PROVIDING INTEGRATED VOICE, DATA AND FAX MESSAGING SERVICES TO PC/WEB-BASED CLIENTS, INCLUDING A LARGE OBJECT SERVER FOR EFFICIENTLY DISTRIBUTING VOICE/FAX MESSAGES TO WEB-BASED CLIENTS**

(75) Inventors: **Nicholas M. Luzeski**, Paoli; **Allie A. Murphy**, Frazer; **John L. Homan**, Ephrata; **Gary Paul Russell**, King of Prussia, all of PA (US)

(73) Assignee: **Unisys Corporation**, Blue Bell, PA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/094,266**

(22) Filed: **Jun. 9, 1998**

(51) **Int. Cl.**[7] **H04L 12/66**; H04M 1/64

(52) **U.S. Cl.** **370/352**; 379/88.13

(58) **Field of Search** 370/352, 353, 370/355, 356, 401; 379/88.13, 88.22, 93.09, 93.25, 220

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,737,395	*	4/1998	Irribarren	379/88.13
5,951,638	*	9/1999	Hoss et al.	709/206
6,025,931	*	2/2000	Bloomfield	358/402
6,058,435	*	5/2000	Sassin et al.	709/305
6,134,235	*	10/2000	Goldman et al.	370/352

FOREIGN PATENT DOCUMENTS

0 845 895 A	6/1998	(EP) .
2 301 260 A	11/1996	(GB) .
WO 96 34341 A	10/1996	(WO) .

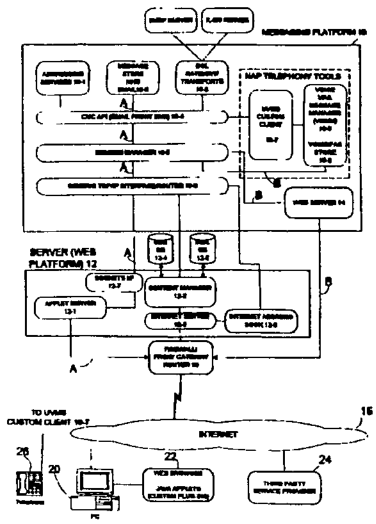
* cited by examiner

Primary Examiner—Wellington Chin
Assistant Examiner—Brenda Pham
(74) *Attorney, Agent, or Firm*—Woodcock, Washburn, Kurtz, Mackiewicz & Norris; Mark T. Starr; Lisa A. Rode

(57) **ABSTRACT**

A Universal Messaging system provides e-mail, voice-mail and fax-mail services to subscribers that may utilize the Internet to access their messages. The system integrates an e-mail messaging system with a voice/fax messaging system on a messaging platform computer. E-mail messages are stored in an e-mail message store, and voice and/or fax messages are stored in a separate store controlled, e.g., by a Voice Mail Message Manager (VMMM). Subscribers can access messages from a personal computer via the Internet using a standard Web browser with an applet that present each subscriber with a "universal inbox" that displays all of that subscriber's voice, fax, and e-mail messages. A Web platform controls the Web browser interface to the messaging platform, accepting requests from the Web browser (such as a request to read an e-mail or listen to a voice mail) and passing prescribed types of information back to the Web browser. The Web platform interfaces with the messaging platform via a generic TCP/IP interface/router. A Session Manager application manages the Web browser's "session" with the messaging system. A CMC layer in the messaging platform provides the "glue" to enable communication and control between and among the different message stores. The CMC layer provides an industry standard mechanism for providing a standard API through which access to proprietary message stores can be made.

9 Claims, 15 Drawing Sheets



[illegible]

12/6-80

US006771646B1

(12) United States Patent
Sarkissian et al.

(10) Patent No.: US 6,771,646 B1
(45) Date of Patent: Aug. 3, 2004

(54) **ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR**

(75) Inventors: **Haig A. Sarkissian**, San Antonio, TX (US); **Russell S. Dietz**, San Jose, CA (US)

(73) Assignee: **Hi/fn, Inc.**, Los Gatos, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 591 days.

(21) Appl. No.: **09/608,266**

(22) Filed: **Jun. 30, 2000**

Related U.S. Application Data

(60) Provisional application No. 60/141,903, filed on Jun. 30, 1999.

(51) **Int. Cl.**[7] **G01R 31/08**
(52) **U.S. Cl.** **370/392**; 370/412; 370/252; 370/352; 709/223; 711/119
(58) **Field of Search** 370/241.1, 252, 370/253, 352, 353, 355, 389, 392, 401, 395.1; 709/223, 224

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,949,369 A * 4/1976 Churchill, Jr. 711/128
4,458,310 A * 7/1984 Chang 711/119
4,559,618 A * 12/1985 Houseman et al. 365/49
4,736,320 A 4/1988 Bristol 364/300
4,891,639 A 1/1990 Nakamura 340/825.5
4,910,668 A * 3/1990 Okamoto et al. 711/207
4,972,453 A 11/1990 Daniel, III et al. 379/10
5,101,402 A 3/1992 Chiu et al.
5,247,517 A 9/1993 Ross et al. 370/85.5
5,247,693 A 9/1993 Bristol 395/800
5,315,580 A 5/1994 Phaal
5,339,268 A 8/1994 Machida 365/49
5,351,243 A 9/1994 Kalkunte et al.
5,365,514 A 11/1994 Hershey et al.
5,375,070 A 12/1994 Hershey et al. 364/550
5,394,394 A 2/1995 Crowther et al.
5,414,650 A 5/1995 Hekhuis 364/715.02
5,414,704 A 5/1995 Spinney
5,430,709 A 7/1995 Galloway

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

JP 02003044510 A * 2/2003 .... G06F/17/30

OTHER PUBLICATIONS

R. Periakaruppam and E. Nemeth. "GTrace–A Graphical Traceroute Tool." 1999 Usenix LISA. Available on www.caida.org, URL: http://www.caida.org/outreach/papers/1999/GTrace.pdf.

W. Stallings. "Packet Filtering in the SNMP Remote Monitor." Nov. 1994. Available on www.ddj.com, URL: http://www.ddj.com/documents/s=1013/ddj9411h/9411h.htm.

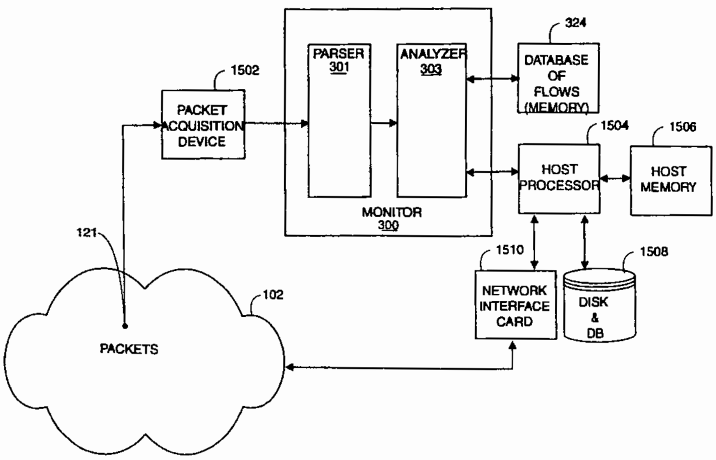
"Technical Note: the Narus System," Downloaded Apr. 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.

Primary Examiner—Ricky Ngo
Assistant Examiner—Alan V. Nguyen
(74) *Attorney, Agent, or Firm*—Dov Rosenfeld; Inventek

(57) **ABSTRACT**

A cache system for looking up one or more elements of an external memory includes 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 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 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.

20 Claims, 21 Drawing Sheets



U.S. PATENT DOCUMENTS

5,432,776 A		7/1995	Harper	
5,493,689 A		2/1996	Waclawsky et al.	395/821
5,500,855 A		3/1996	Hershey et al.	
5,511,215 A		4/1996	Terasaka et al.	395/800
5,530,834 A	*	6/1996	Colloff et al.	711/136
5,530,958 A	*	6/1996	Agarwal et al.	711/3
5,535,338 A		7/1996	Krause et al.	395/200.2
5,568,471 A		10/1996	Hershey et al.	
5,574,875 A		11/1996	Stansfield et al.	395/403
5,586,266 A		12/1996	Hershey et al.	395/200.11
5,606,668 A		2/1997	Shwed	395/200.11
5,608,662 A		3/1997	Large et al.	364/724.01
5,634,009 A		5/1997	Iddon et al.	395/200.11
5,651,002 A		7/1997	Van Seters et al.	370/392
5,684,954 A		11/1997	Kaiserswerth et al.	395/200.2
5,703,877 A		12/1997	Nuber et al.	370/395
5,720,032 A		2/1998	Picazo, Jr. et al.	395/200.2
5,732,213 A		3/1998	Gessel et al.	395/200.11
5,740,355 A		4/1998	Watanabe et al.	395/183.21
5,749,087 A	*	5/1998	Hoover et al.	711/108
5,761,424 A		6/1998	Adams et al.	395/200.47
5,764,638 A		6/1998	Ketchum	370/401
5,781,735 A		7/1998	Southard	395/200.54
5,784,298 A		7/1998	Hershey et al.	364/557
5,787,253 A		7/1998	McCreery et al.	395/200.61
5,802,054 A		9/1998	Bellenger	370/401
5,805,808 A		9/1998	Hasani et al.	395/200.2
5,812,529 A		9/1998	Czarnik et al.	370/245
5,819,028 A		10/1998	Manghirmalani et al.	395/185.1
5,825,774 A		10/1998	Ready et al.	370/401
5,835,726 A		11/1998	Shwed et al.	395/200.59
5,835,963 A		11/1998	Yoshioka et al.	711/207
5,838,919 A		11/1998	Schwaller et al.	395/200.54
5,841,895 A		11/1998	Huffman	382/155
5,850,386 A		12/1998	Anderson et al.	370/241
5,850,388 A		12/1998	Anderson et al.	370/252
5,860,114 A		1/1999	Sell	711/146
5,862,335 A		1/1999	Welch, Jr. et al.	395/200.54
5,878,420 A		3/1999	de la Salle	707/10
5,893,155 A		4/1999	Cheriton	711/144
5,903,754 A		5/1999	Pearson	395/680
5,917,821 A		6/1999	Gobuyan et al.	370/392
6,003,123 A	*	12/1999	Carter et al.	711/207
6,014,380 A		1/2000	Hendel et al.	370/392
6,115,393 A		9/2000	Engel et al.	370/469
6,279,113 B1		8/2001	Vaidya	713/201
6,330,226 B1		12/2001	Chapman et al.	370/232
6,363,056 B1		3/2002	Beigi et al.	370/252
6,424,624 B1		7/2002	Galand et al.	370/231
6,625,657 B1		9/2003	Bullard	709/237
6,651,099 B1		11/2003	Dietz et al.	709/224

* cited by examiner

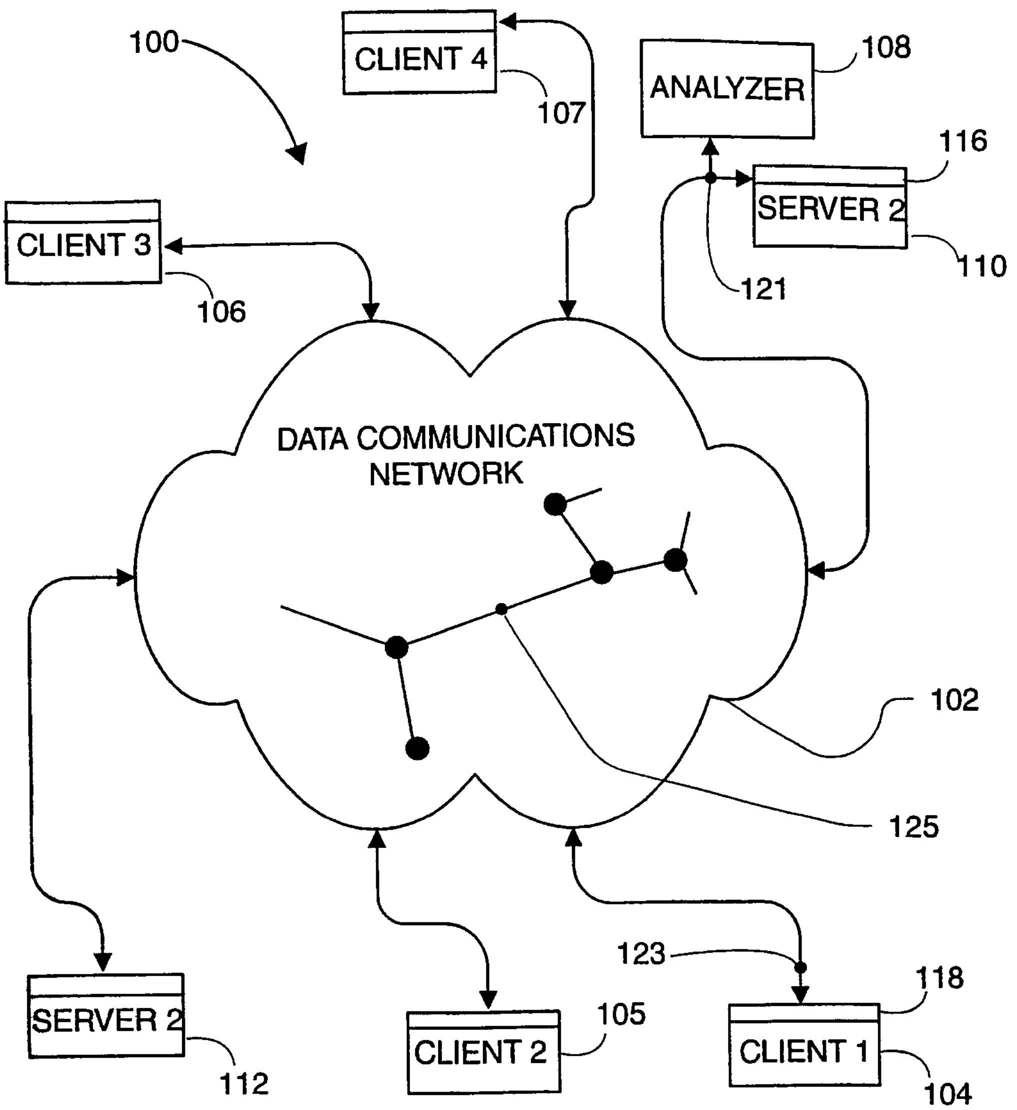

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS
NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

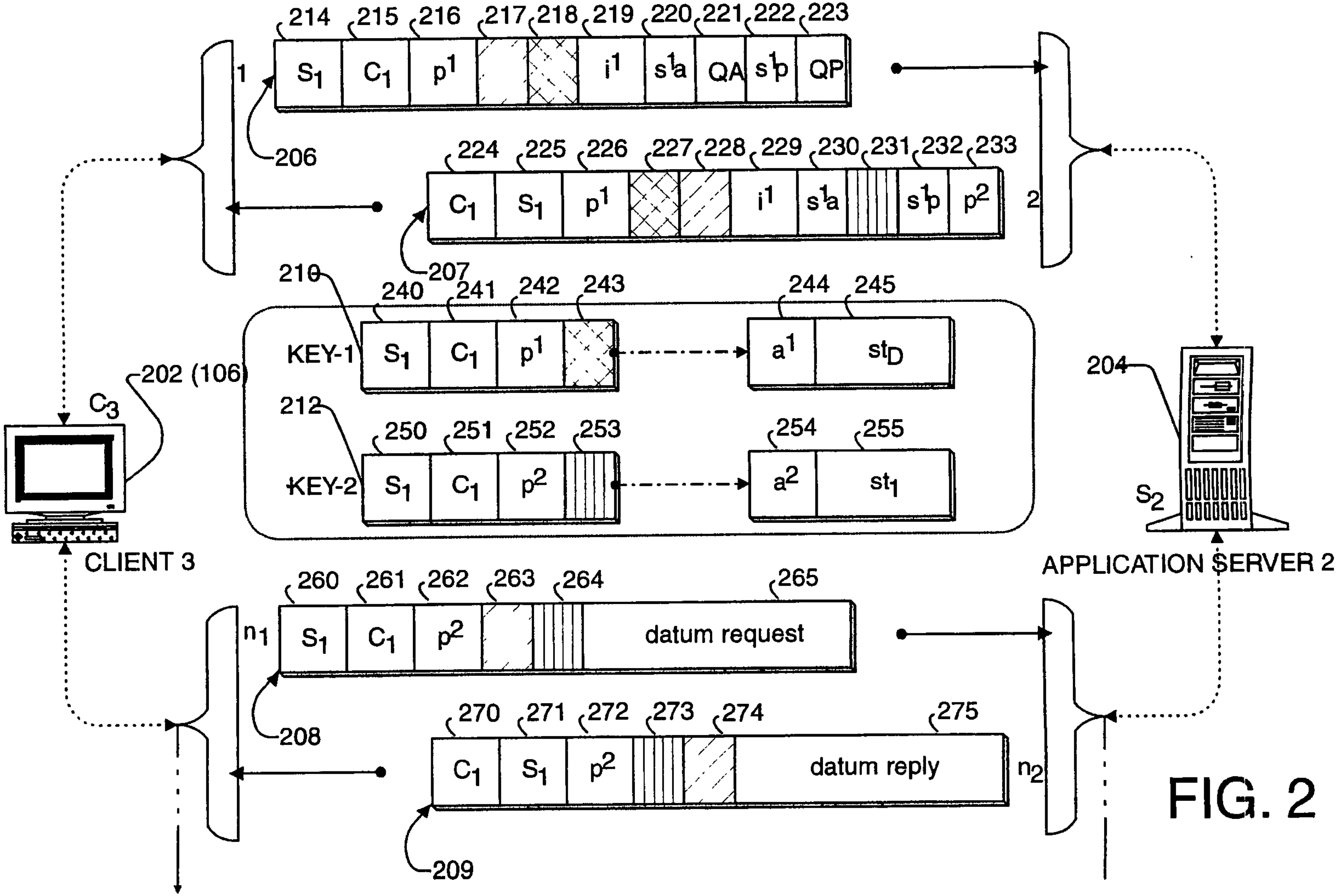

214
215
216
217
218
219
220
221
222
223
S1
C1
p1
i1
s1a
QA
s1p
QP
1
206
224
225
226
227
228
229
230
231
232
233
C1
S1
p1
i1
s1a
s1p
p2
2
207
210
240
241
242
243
244
245
KEY-1
S1
C1
p1
a1
stD
212
250
251
252
253
254
255
KEY-2
S1
C1
p2
a2
st1
202 (106)
C3
CLIENT 3
204
S2
APPLICATION SERVER 2
260
261
262
263
264
265
n1
S1
C1
p2
datum request
208
270
271
272
273
274
275
C1
S1
p2
datum reply
n2
209

FIG. 2

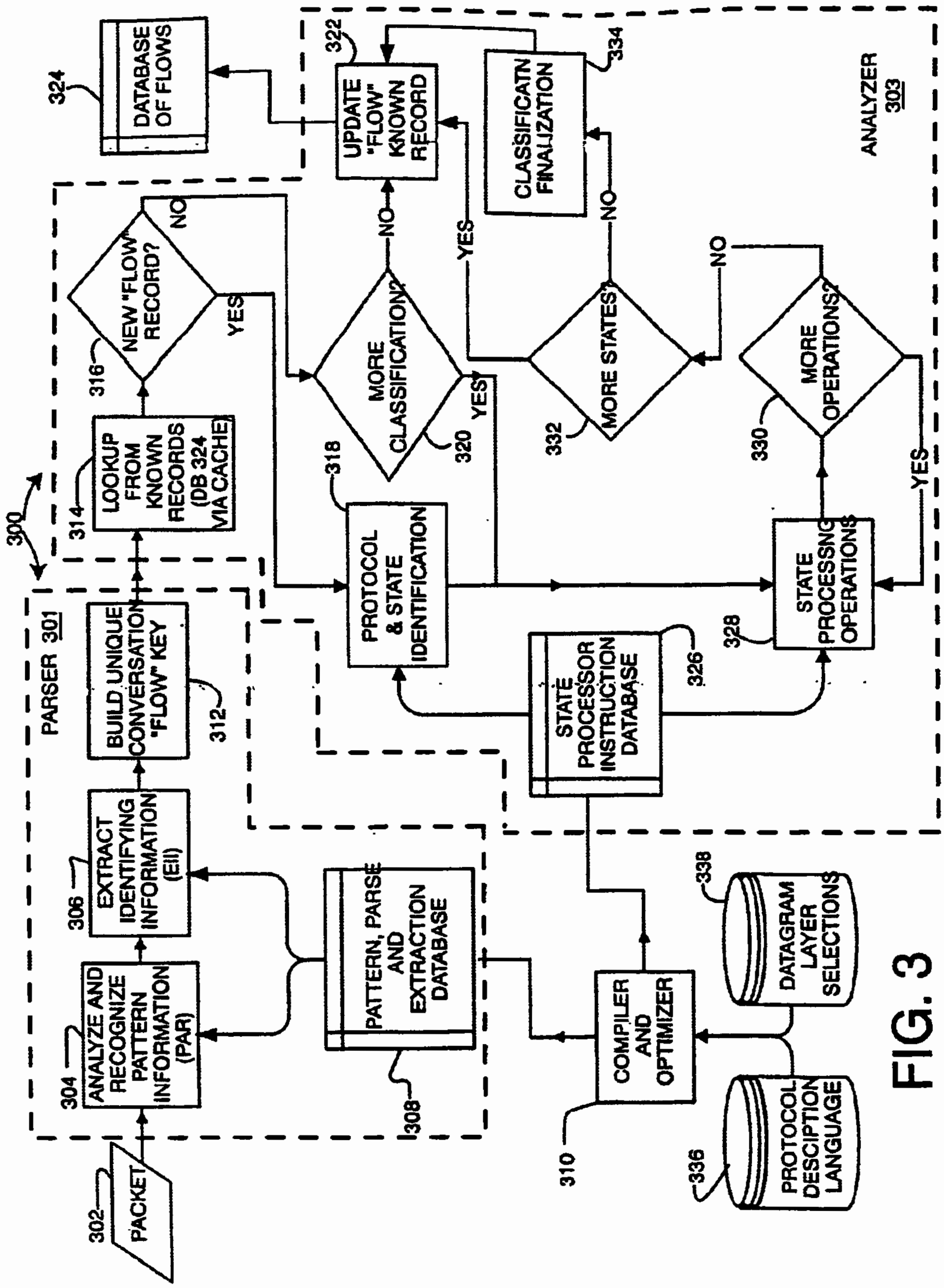

300
PACKET
302
PARSER 301
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
304
EXTRACT IDENTIFYING INFORMATION (EII)
306
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
PATTERN, PARSE AND EXTRACTION DATABASE
308
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
314
NEW "FLOW" RECORD?
316
YES
NO
PROTOCOL & STATE IDENTIFICATION
318
MORE CLASSIFICATION?
320
YES
NO
UPDATE "FLOW" KNOWN RECORD
322
DATABASE OF FLOWS
324
CLASSIFICATN FINALIZATION
334
MORE STATES?
332
YES
NO
STATE PROCESSOR INSTRUCTION DATABASE
326
STATE PROCESSING OPERATIONS
328
MORE OPERATIONS?
330
YES
NO
ANALYZER 303
COMPILER AND OPTIMIZER
310
PROTOCOL DESCRIPTION LANGUAGE
336
DATAGRAM LAYER SELECTIONS
338

FIG. 3

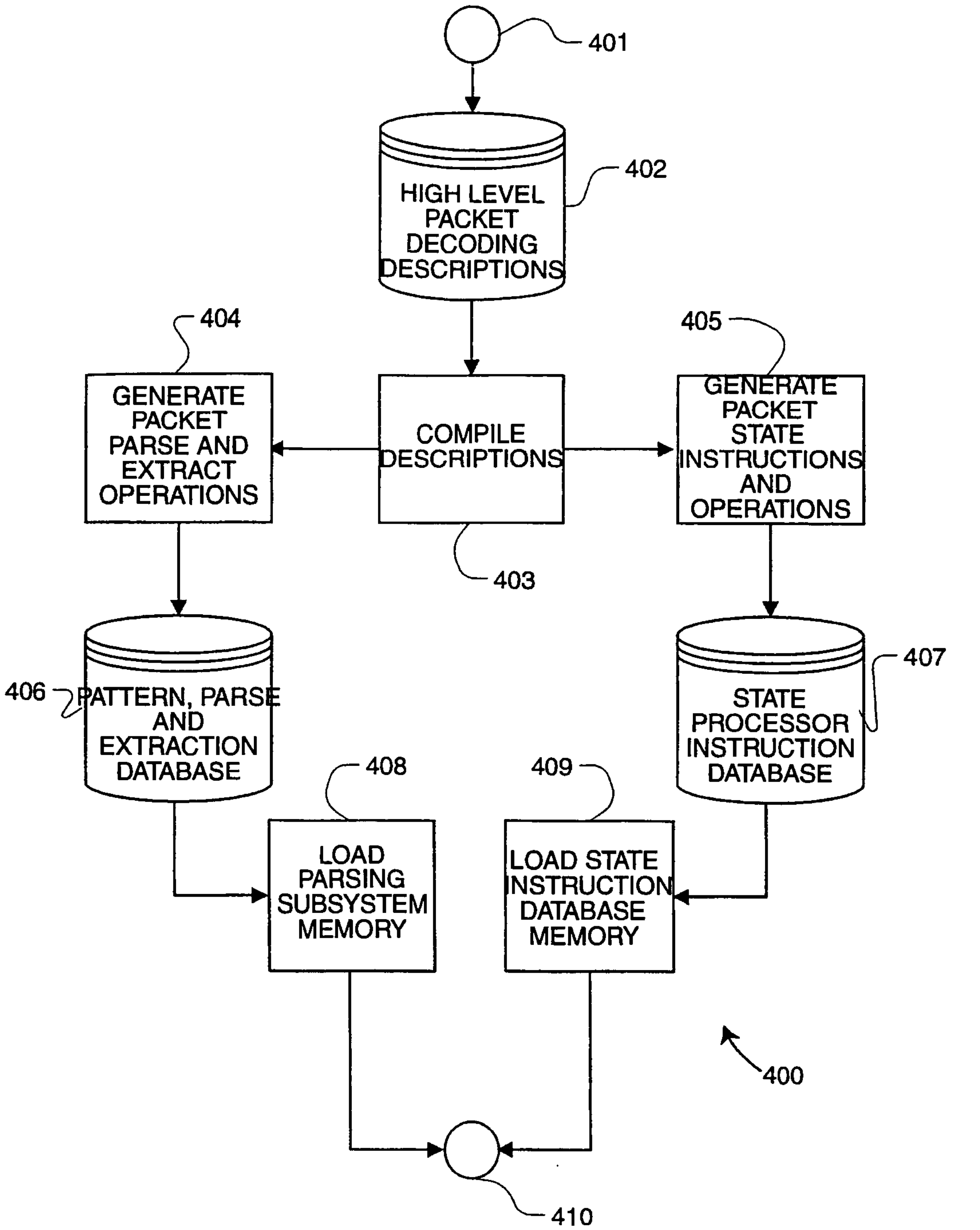
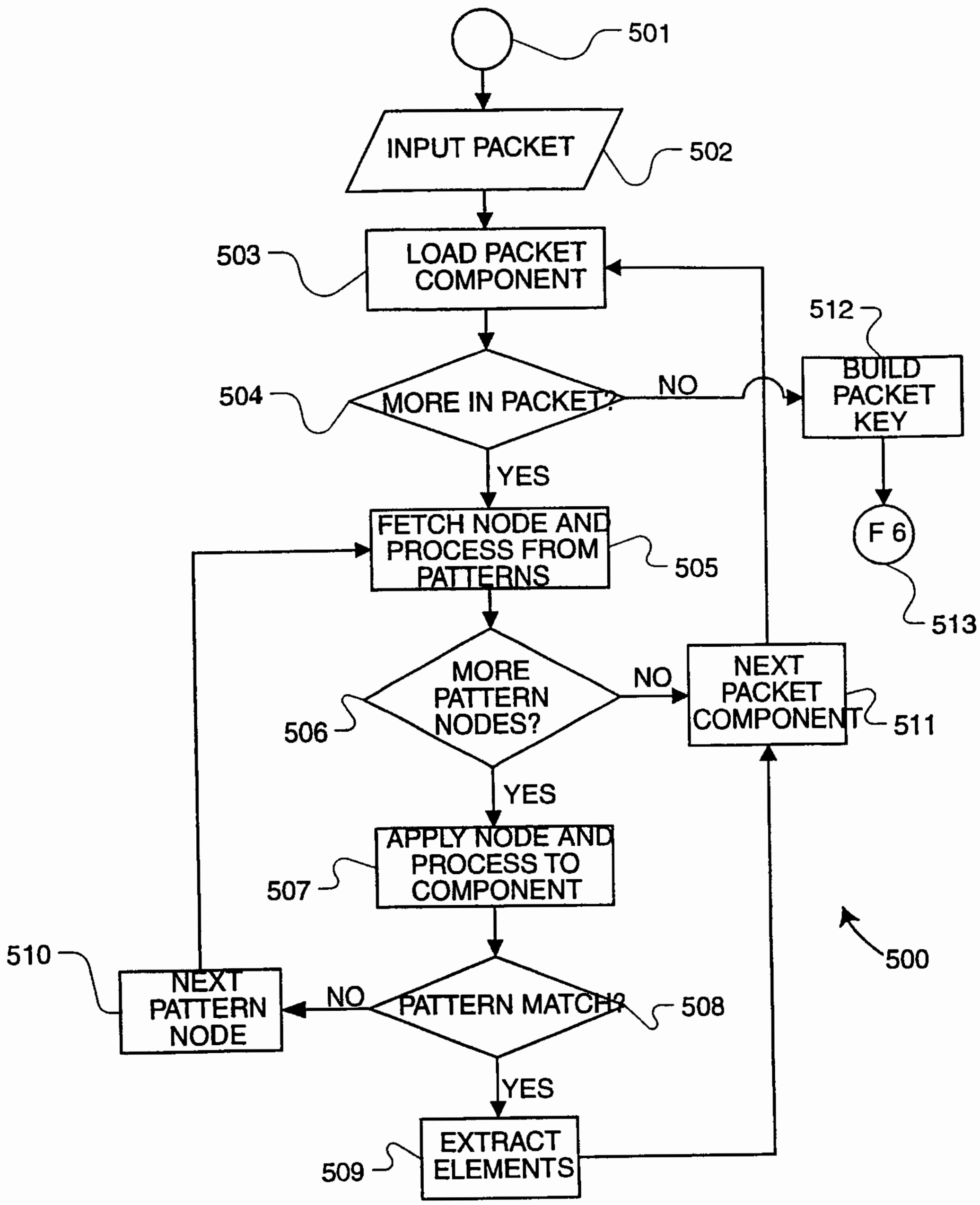

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
403
405
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
LOAD PARSING SUBSYSTEM MEMORY
409
LOAD STATE INSTRUCTION DATABASE MEMORY
400
410

FIG. 4



501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
504
MORE IN PACKET?
NO
512
BUILD PACKET KEY
F 6
513
YES
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
506
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT ELEMENTS
509

FIG. 5

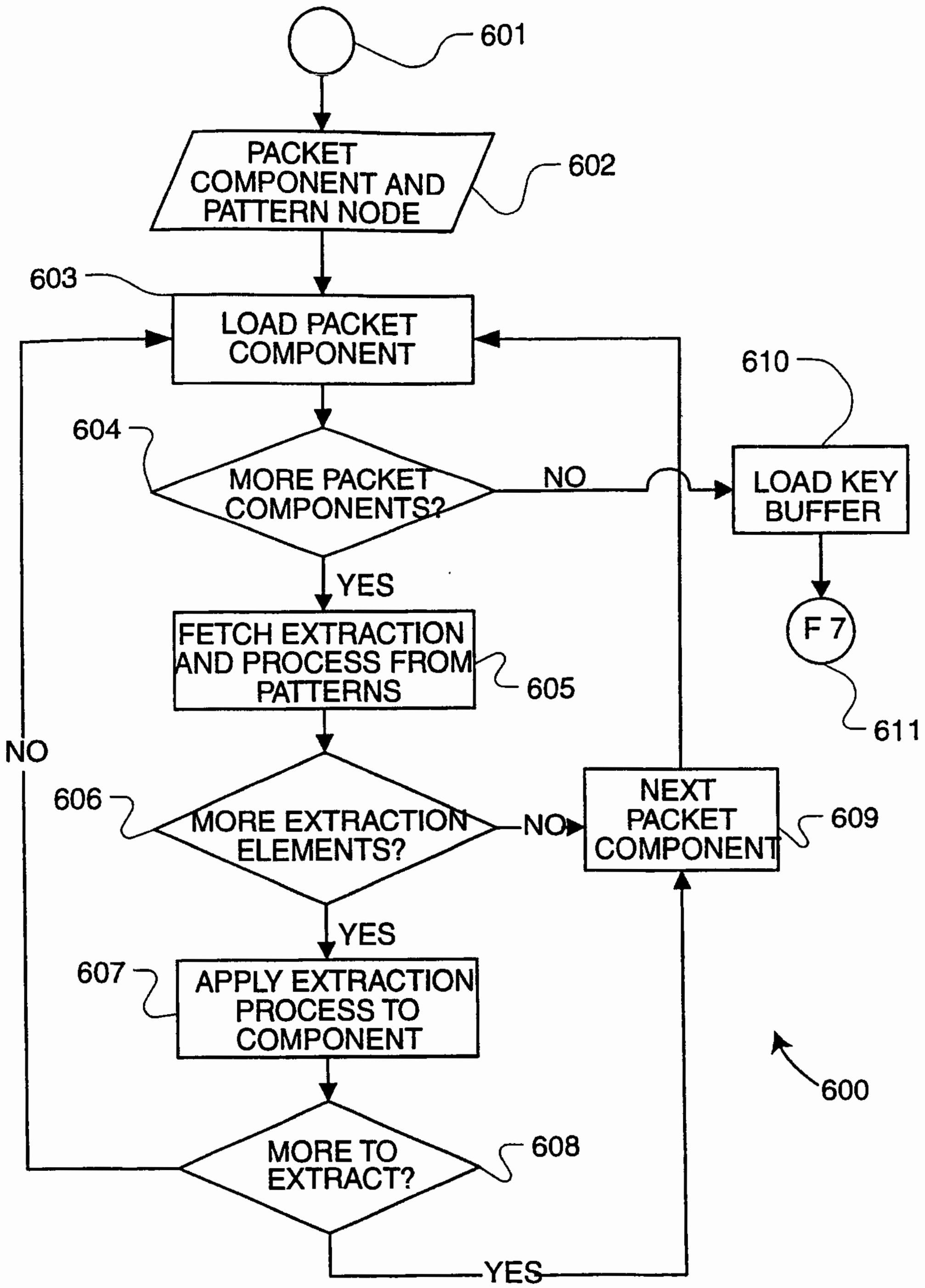

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

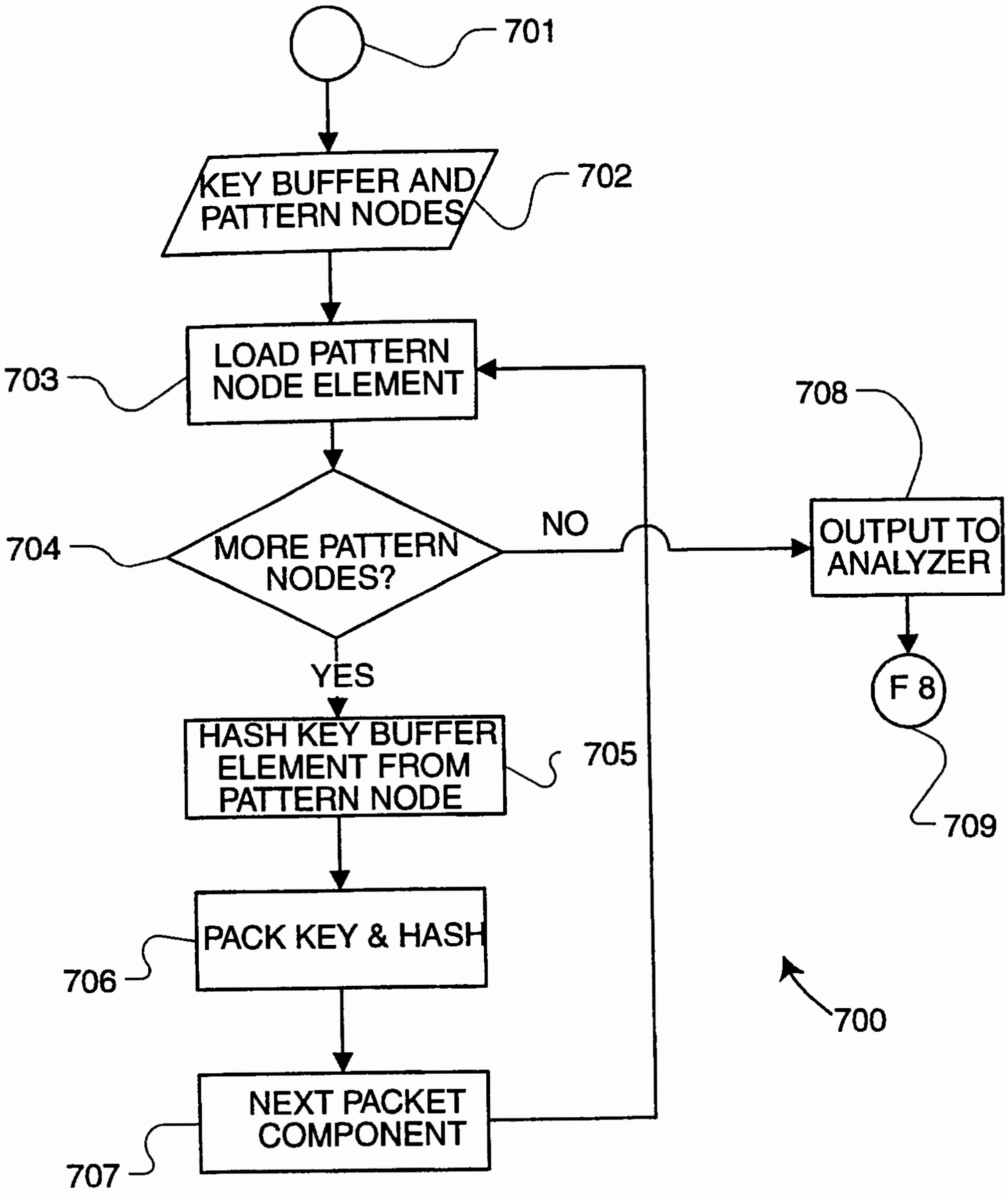

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
704
MORE PATTERN NODES?
NO
YES
708
OUTPUT TO ANALYZER
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

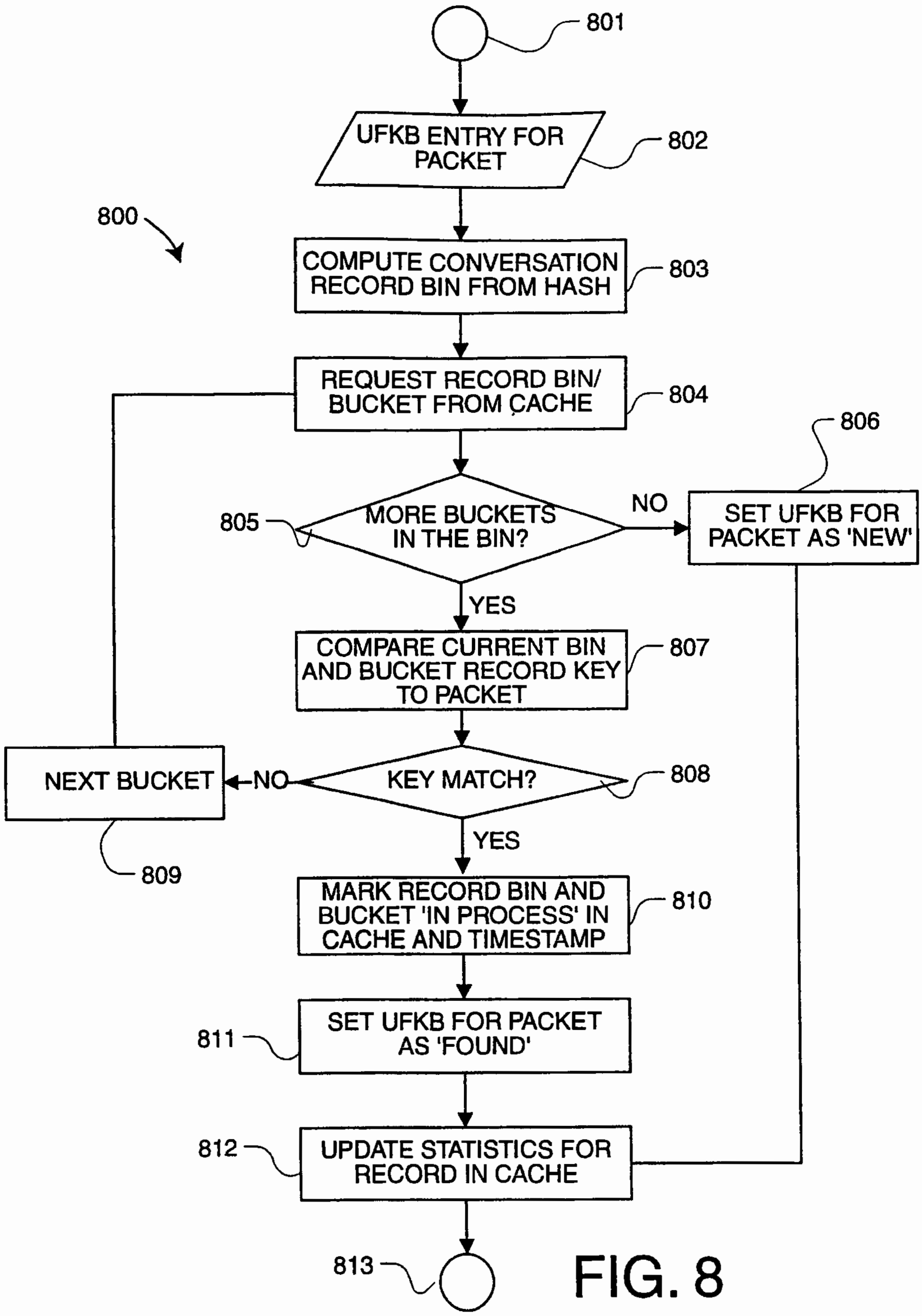
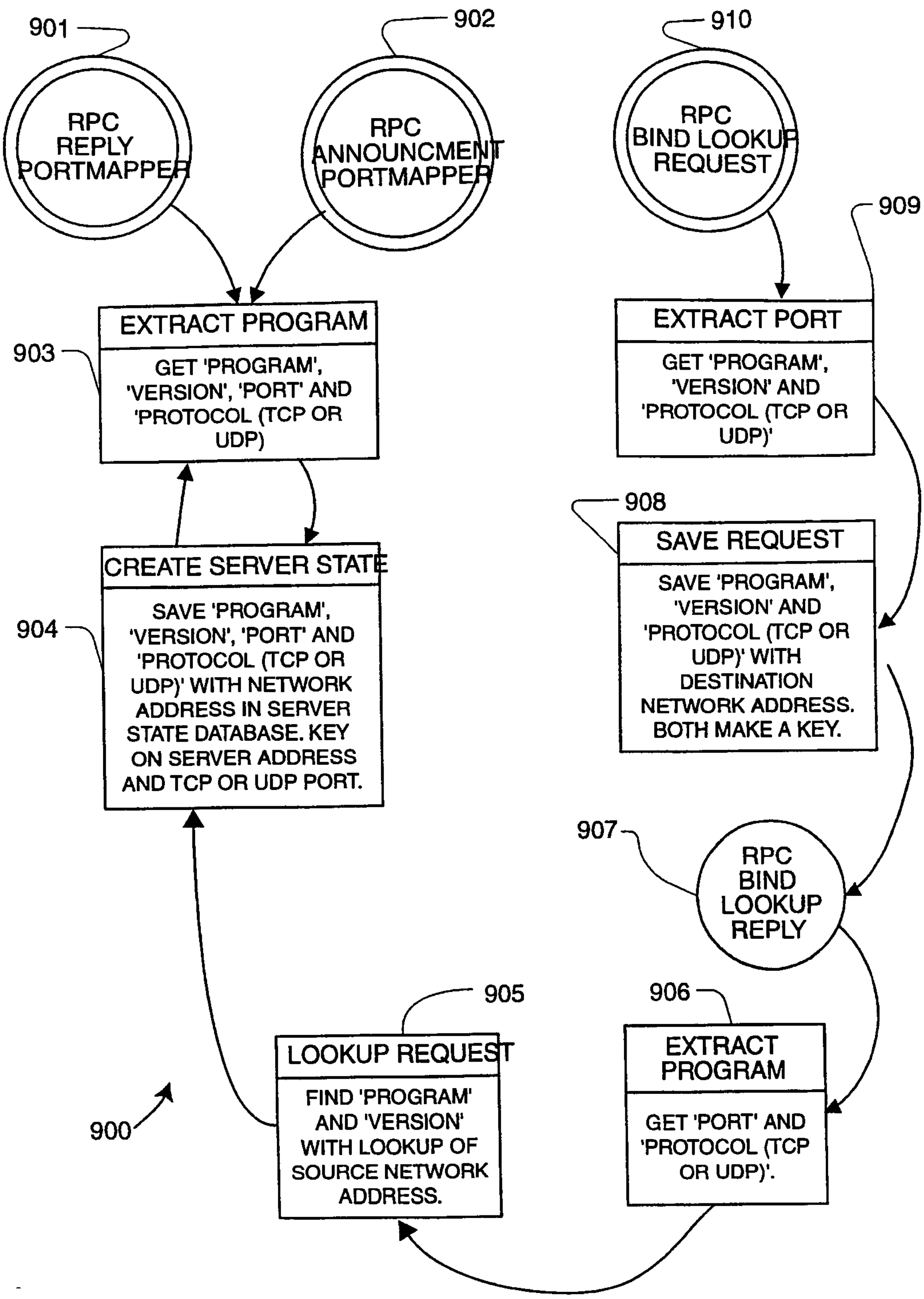

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
809
YES
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
811
SET UFKB FOR PACKET AS 'FOUND'
812
UPDATE STATISTICS FOR RECORD IN CACHE
813

FIG. 8



901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
906
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

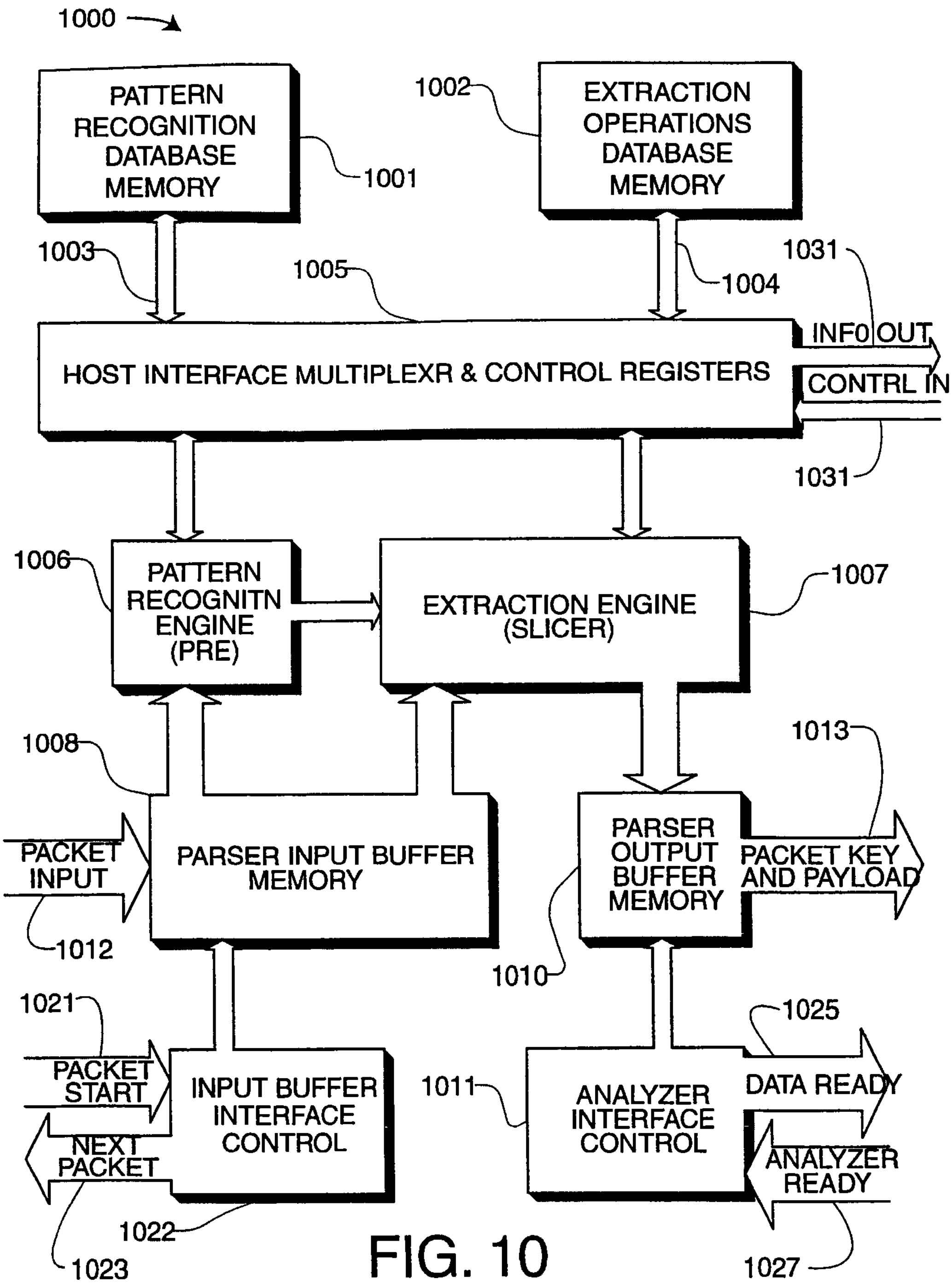

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
INFO OUT
CONTRL IN
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

FIG. 10

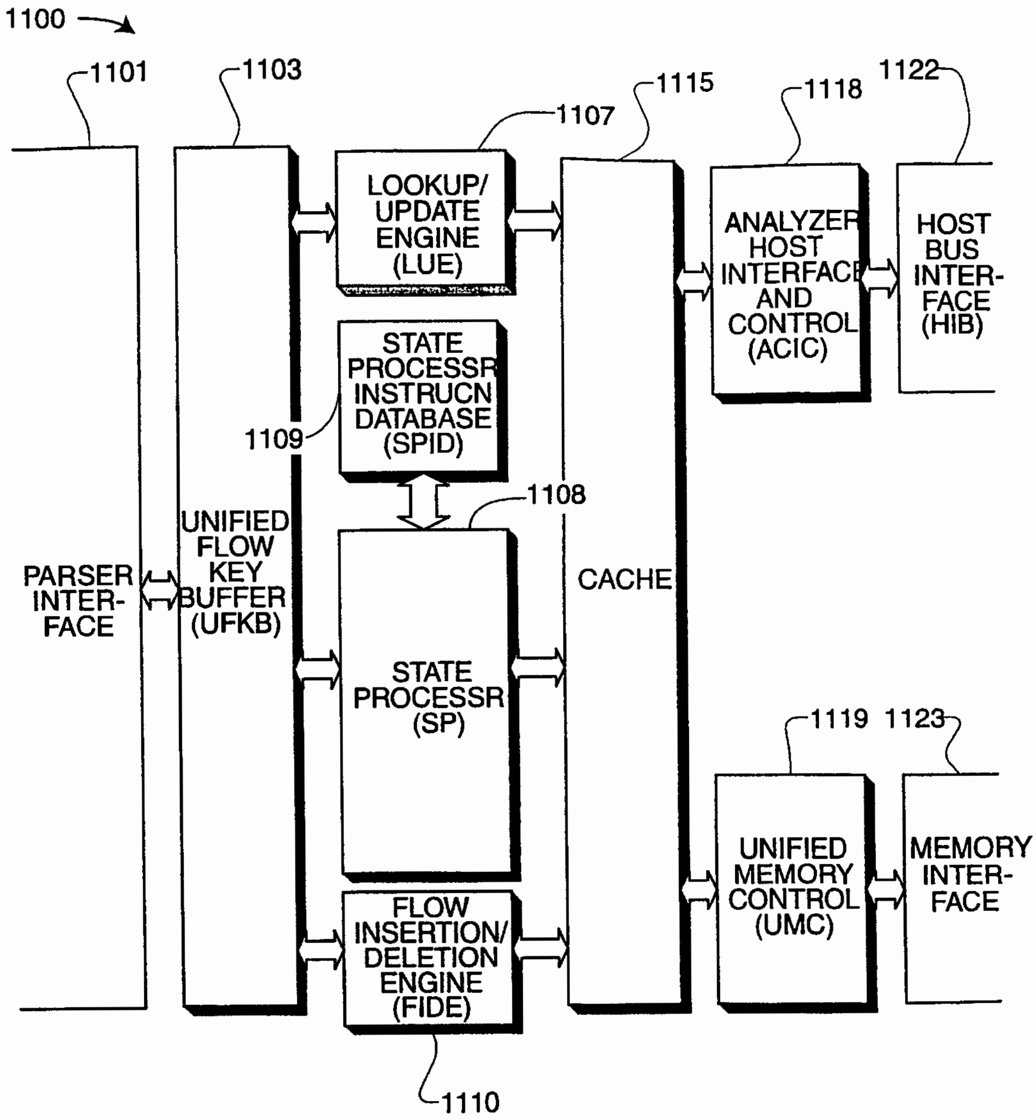

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER- FACE (HIB)
PARSER INTER- FACE
UNIFIED FLOW KEY BUFFER (UFKB)
CACHE
STATE PROCESSR (SP)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY INTER- FACE
FLOW INSERTION/ DELETION ENGINE (FIDE)
1110

FIG. 11

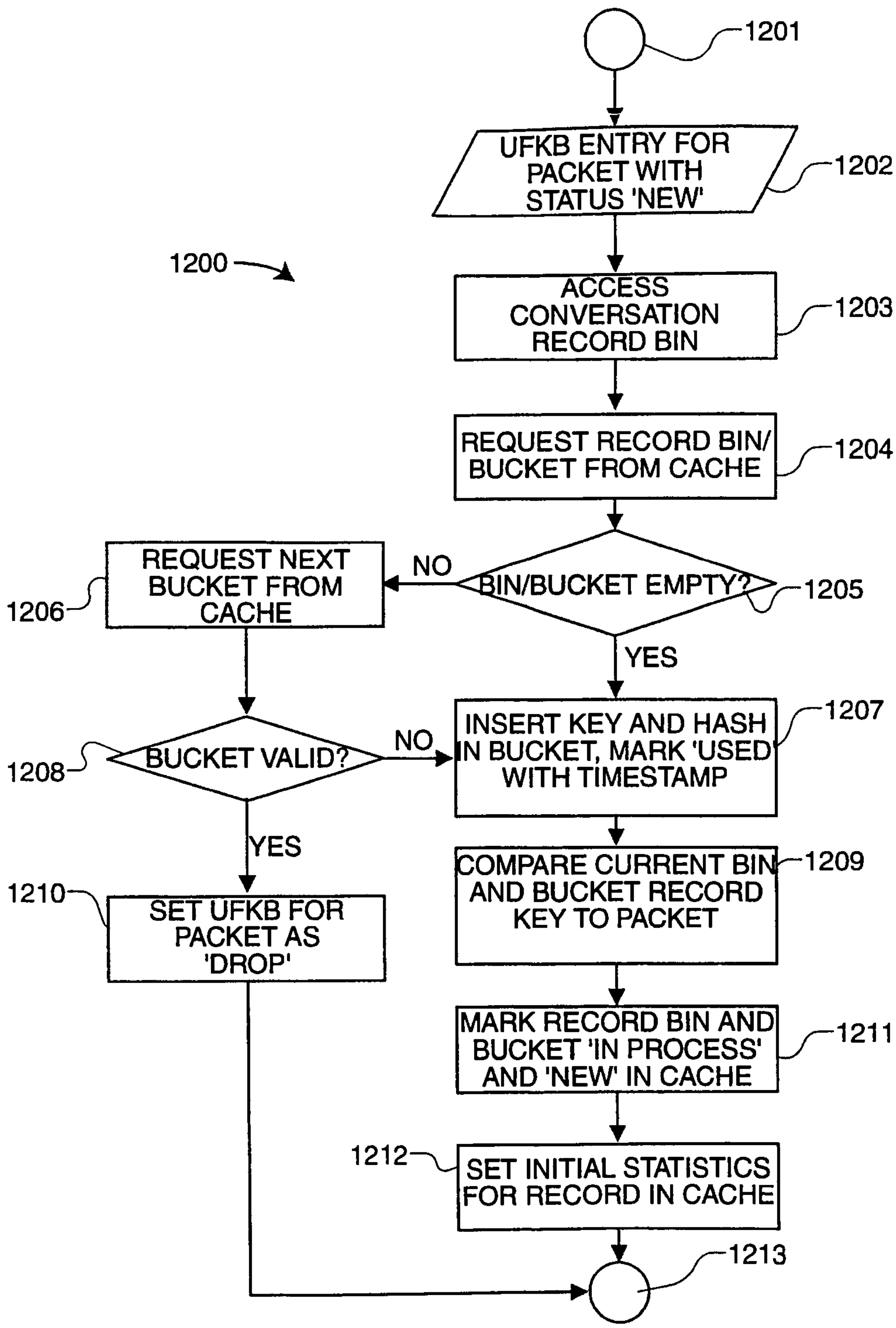

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
BIN/BUCKET EMPTY?
1205
NO
REQUEST NEXT BUCKET FROM CACHE
1206
YES
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
BUCKET VALID?
1208
NO
YES
SET UFKB FOR PACKET AS 'DROP'
1210
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
SET INITIAL STATISTICS FOR RECORD IN CACHE
1212
1213

FIG. 12

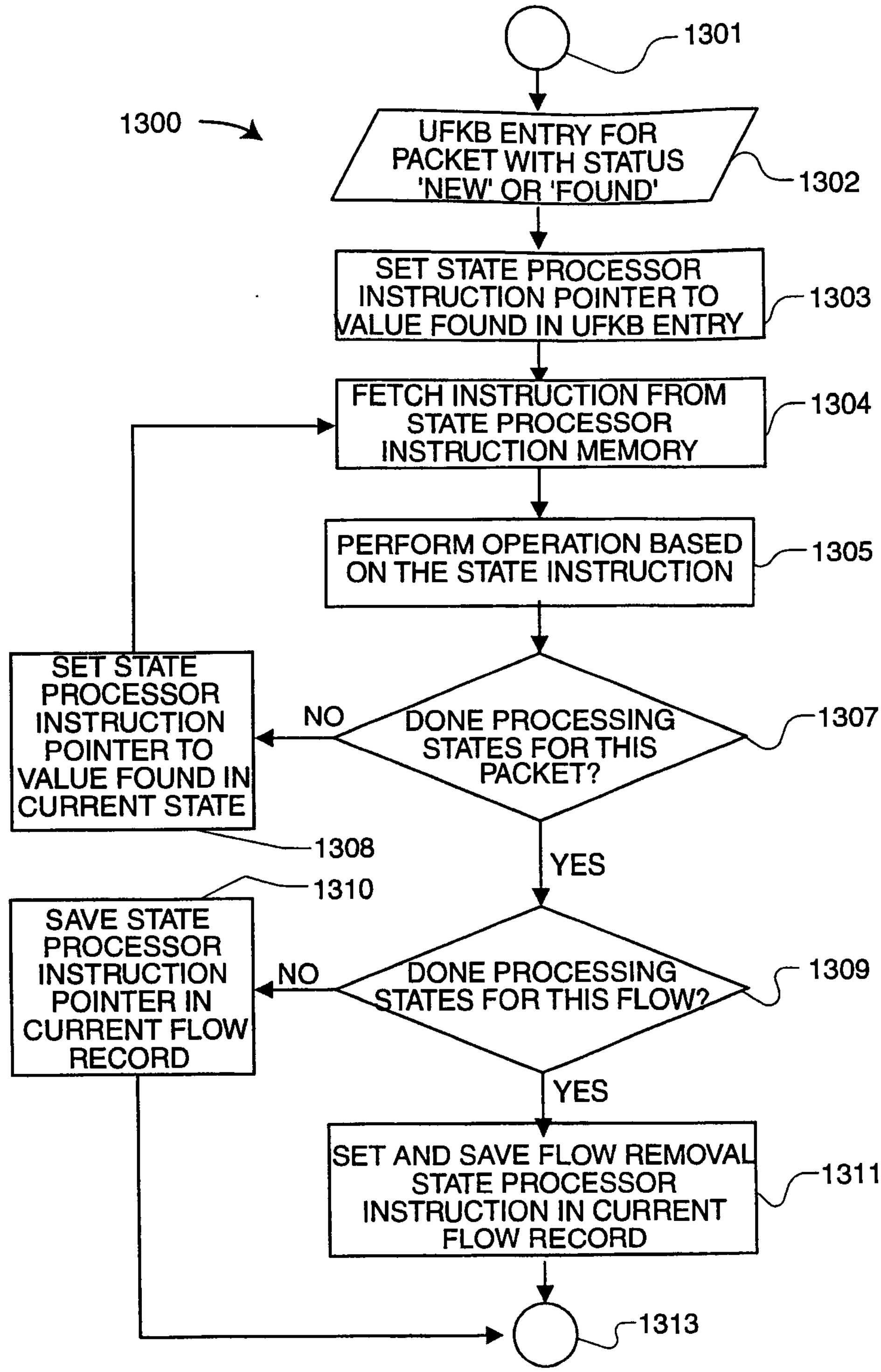

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
NO
DONE PROCESSING STATES FOR THIS PACKET?
1307
1308
YES
1310
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
NO
DONE PROCESSING STATES FOR THIS FLOW?
1309
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

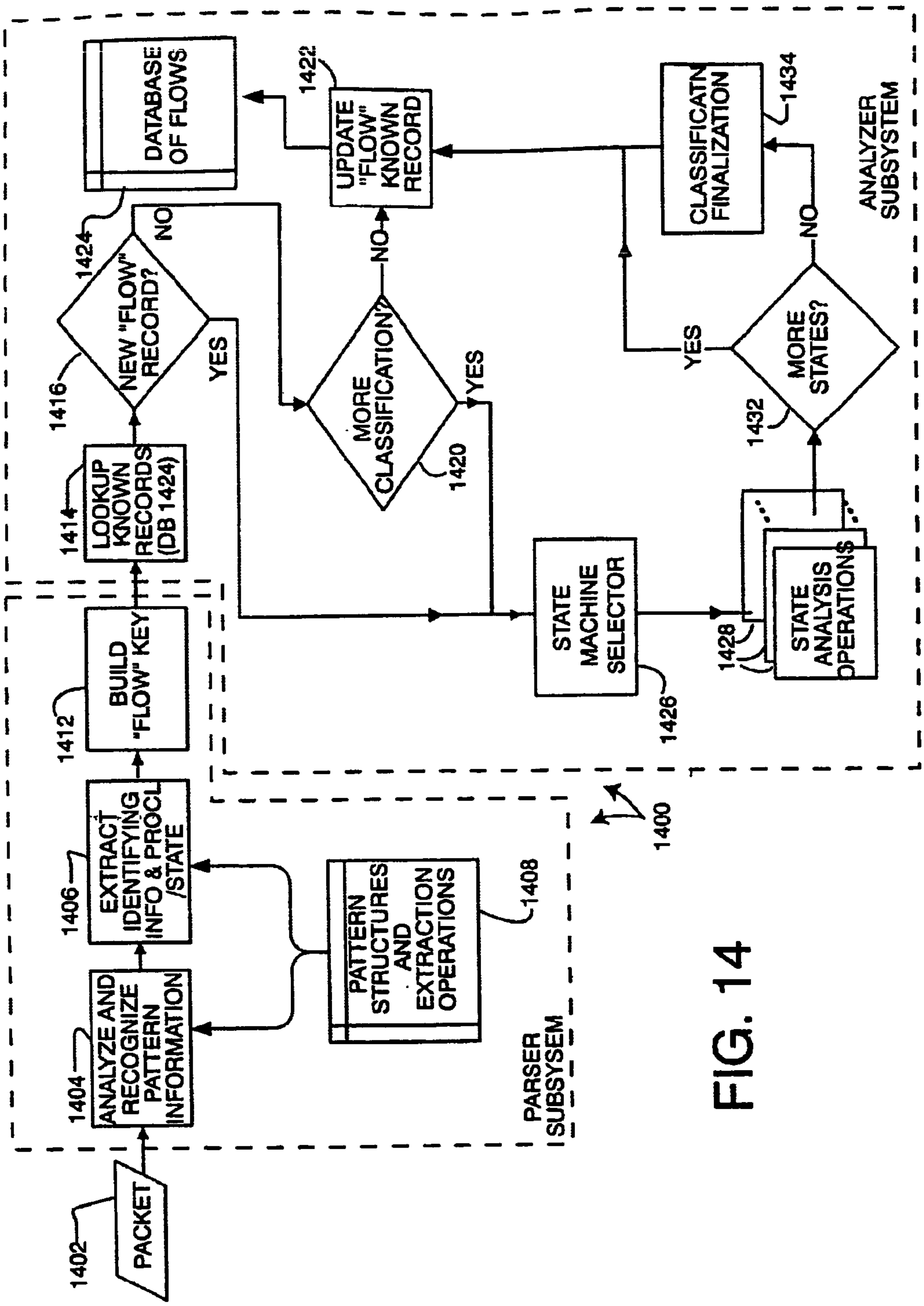

PACKET
1402
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1404
EXTRACT IDENTIFYING INFO & PROCL /STATE
1406
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
BUILD "FLOW" KEY
1412
LOOKUP KNOWN RECORDS (DB 1424)
1414
NEW "FLOW" RECORD?
1416
YES
NO
DATABASE OF FLOWS
1424
MORE CLASSIFICATION?
1420
YES
NO
UPDATE "FLOW" KNOWN RECORD
1422
STATE MACHINE SELECTOR
1426
STATE ANALYSIS OPERATIONS
1428
MORE STATES?
1432
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM
1400

FIG. 14

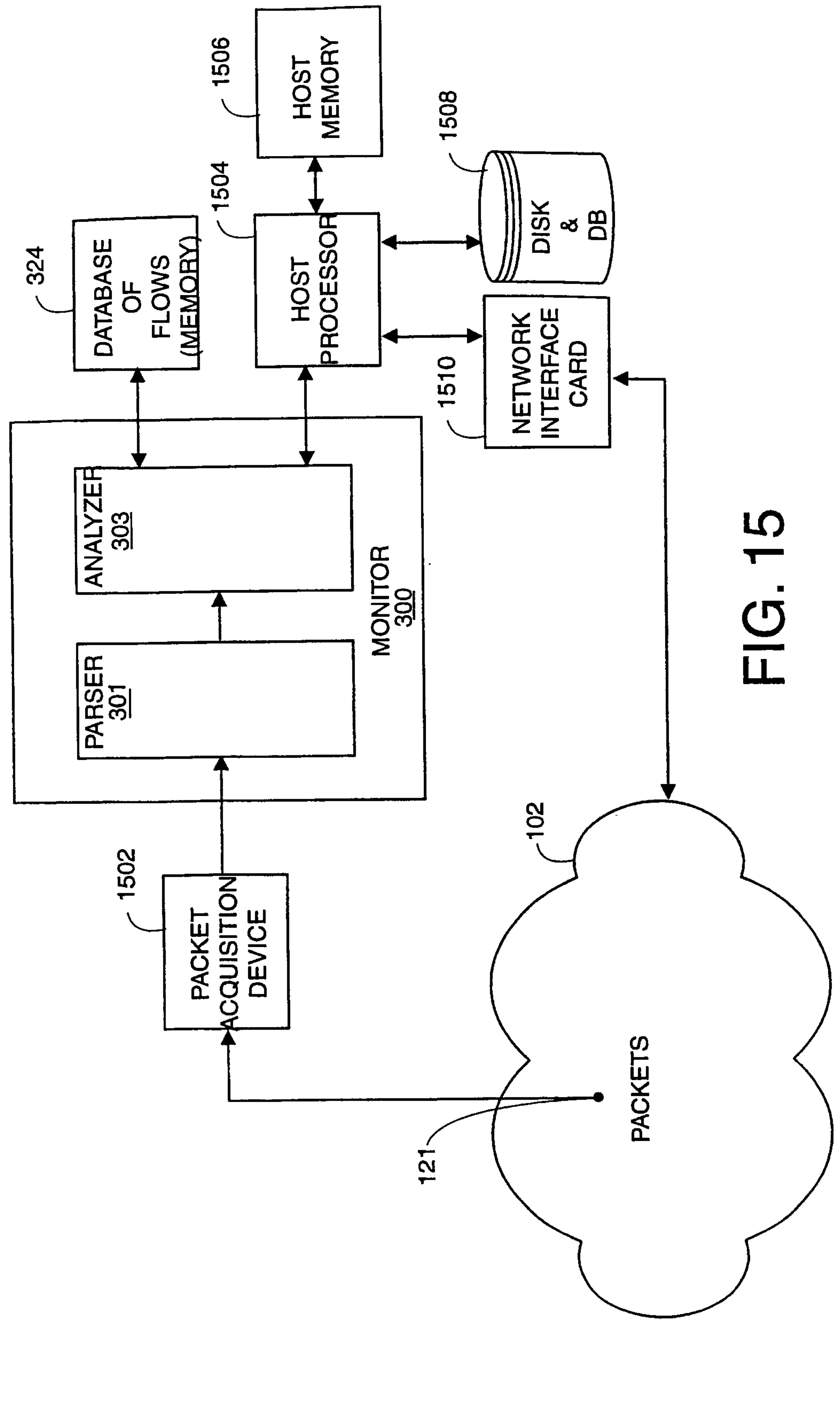

1506
HOST MEMORY
1508
DISK & DB
1504
HOST PROCESSOR
324
DATABASE OF FLOWS (MEMORY)
1510
NETWORK INTERFACE CARD
ANALYZER 303
PARSER 301
MONITOR 300
1502
PACKET ACQUISITION DEVICE
102
PACKETS
121

FIG. 15

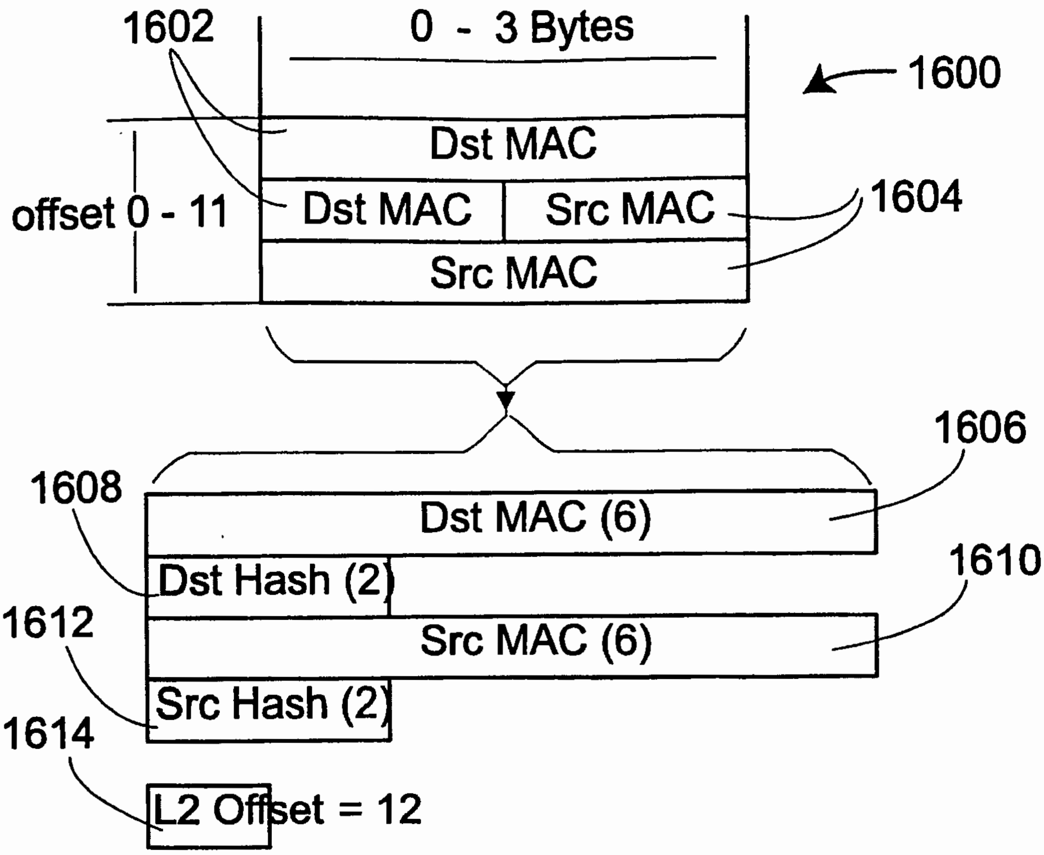

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

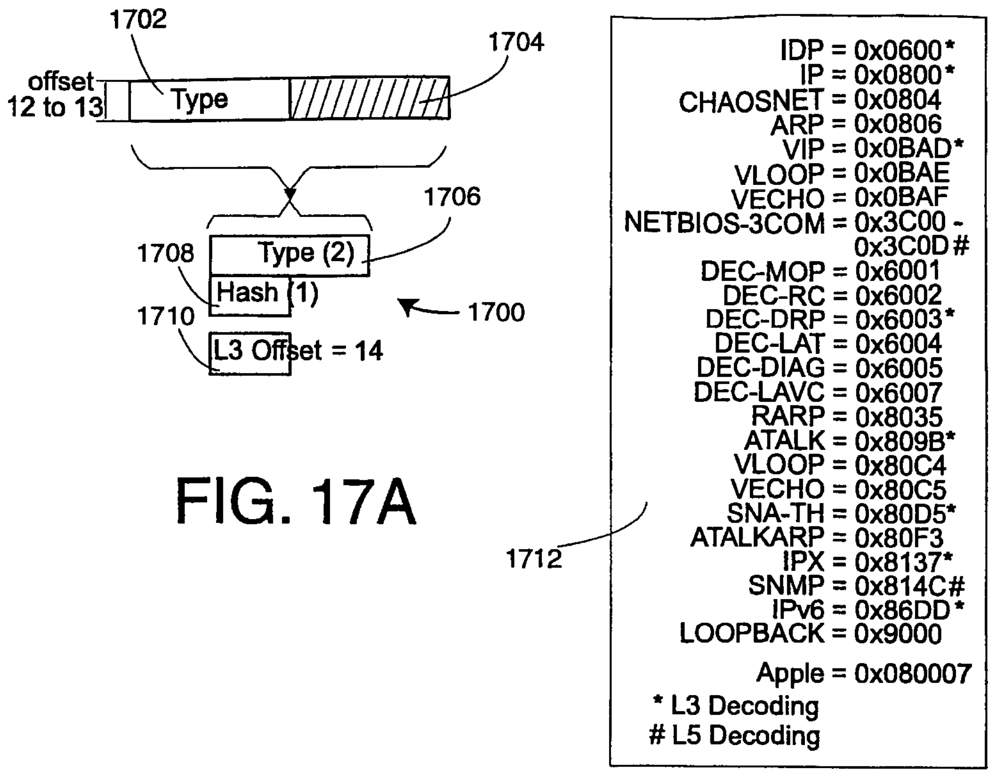


FIG. 17A

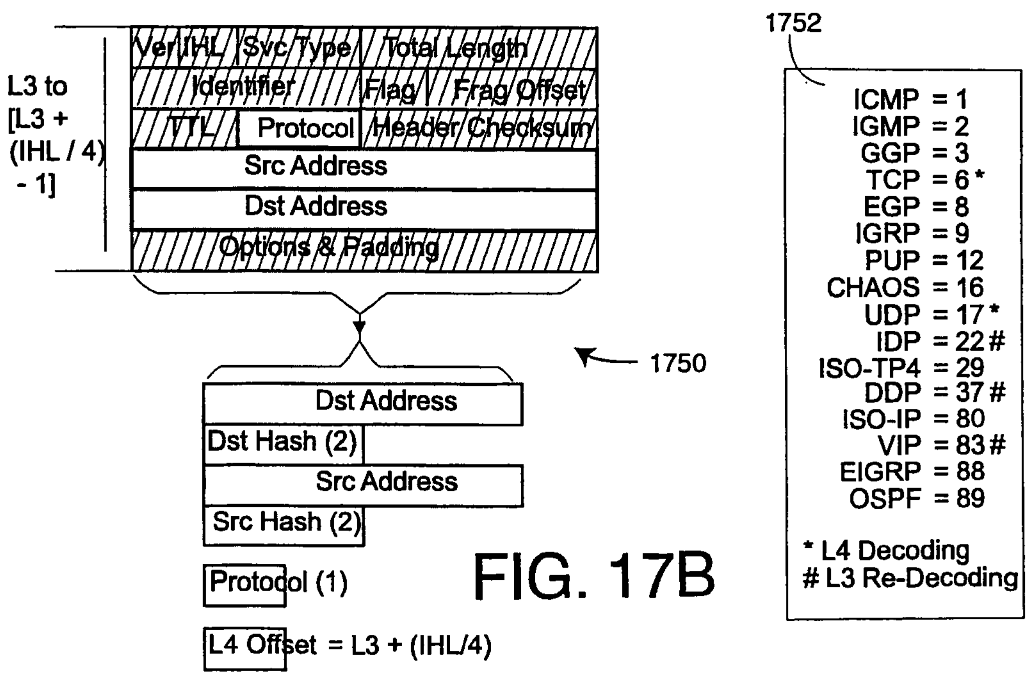


FIG. 17B

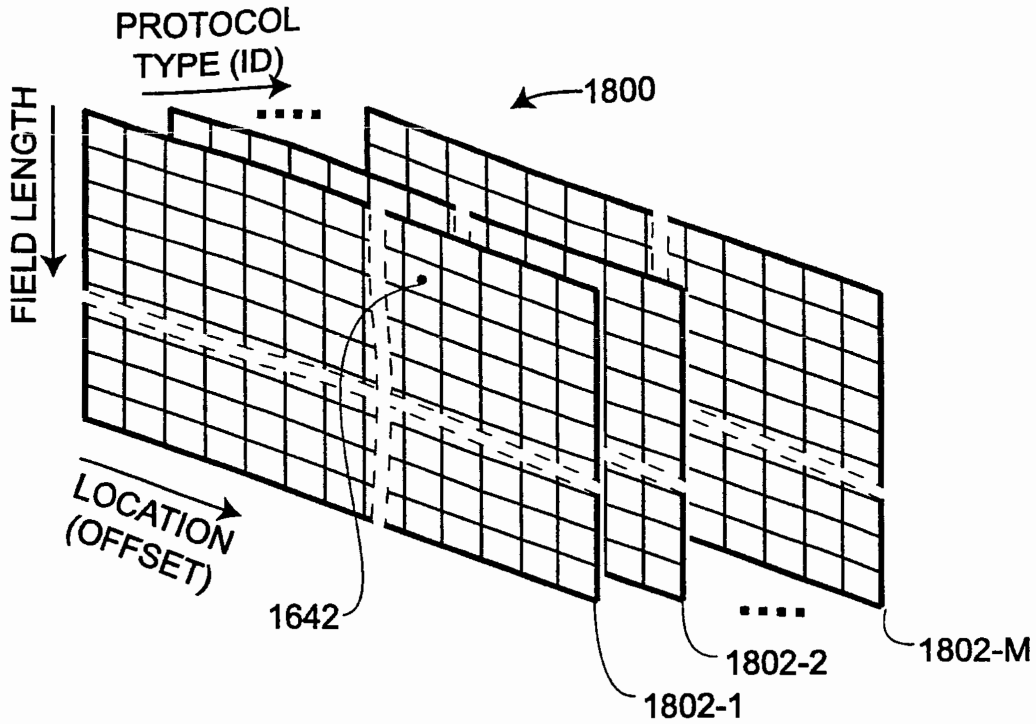


FIG. 18A

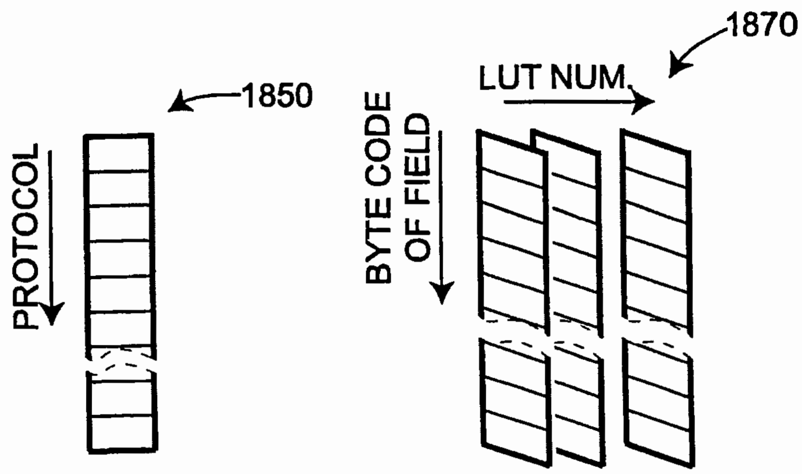


FIG. 18B

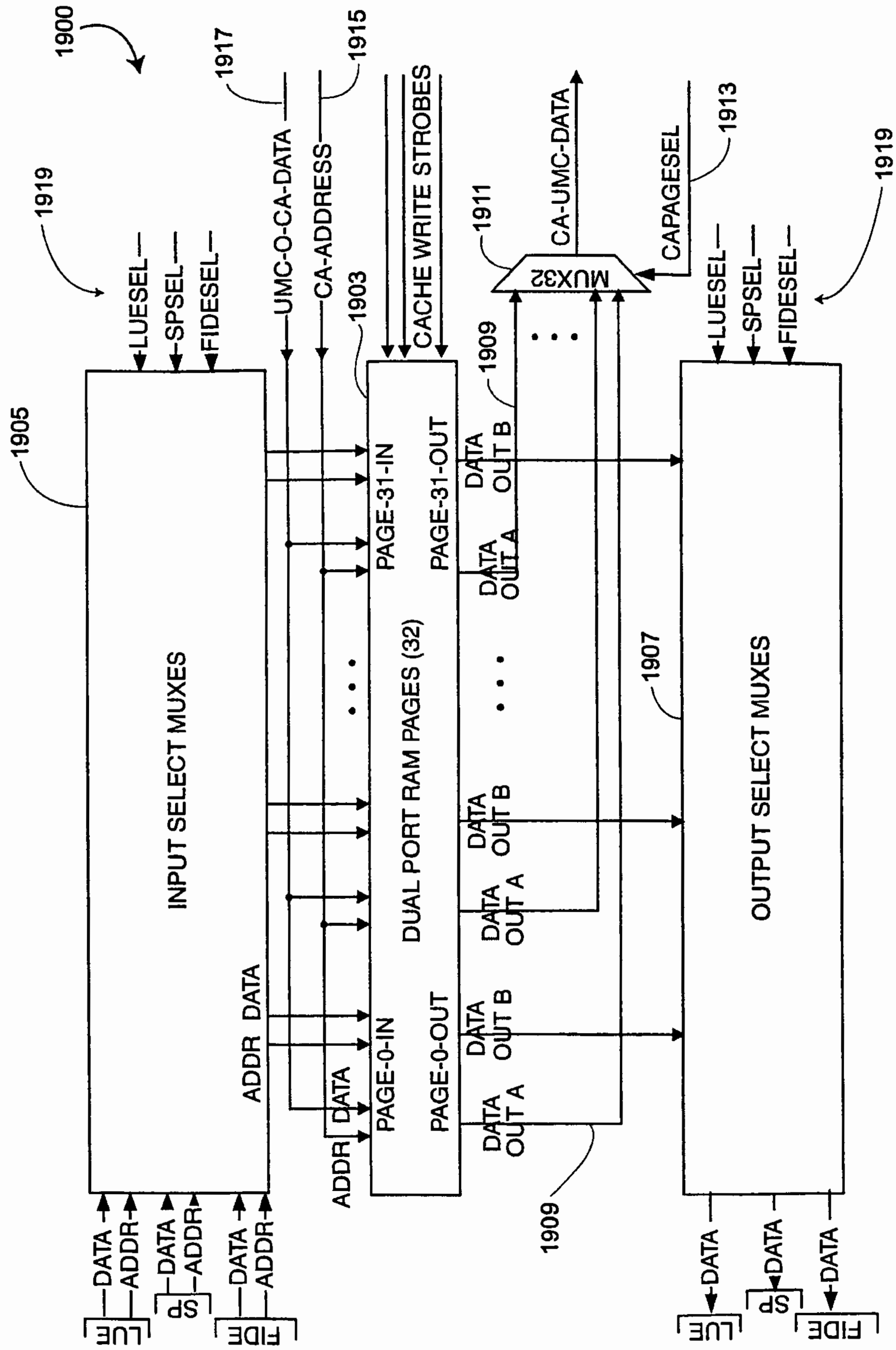

1900
1905
INPUT SELECT MUXES
1919
LUESEL
SPSEL
FIDESEL
UMC-O-CA-DATA
CA-ADDRESS
1917
1915
1903
CACHE WRITE STROBES
DUAL PORT RAM PAGES (32)
PAGE-31-IN
PAGE-31-OUT
PAGE-0-IN
PAGE-0-OUT
ADDR
DATA
DATA OUT A
DATA OUT B
1909
1911
MUX32
CA-UMC-DATA
CAPAGESEL
1913
1907
OUTPUT SELECT MUXES
LUE
SP
FIDE

FIG. 19

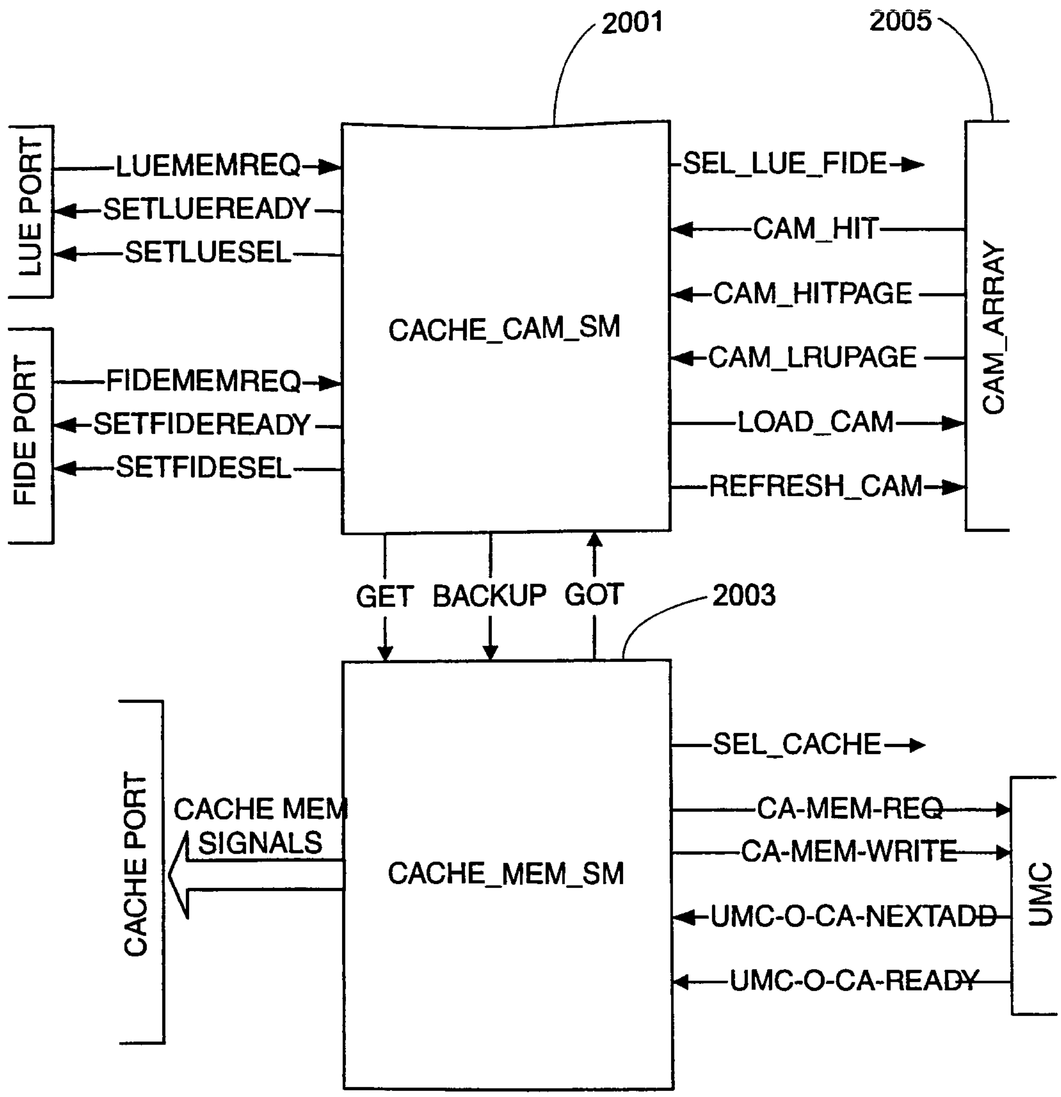

2001
2005
LUE PORT
LUEMEMREQ
SETLUEREADY
SETLUESEL
FIDE PORT
FIDEMEMREQ
SETFIDEREADY
SETFIDESEL
CACHE_CAM_SM
SEL_LUE_FIDE
CAM_HIT
CAM_HITPAGE
CAM_LRUPAGE
LOAD_CAM
REFRESH_CAM
CAM_ARRAY
GET
BACKUP
GOT
2003
CACHE PORT
CACHE MEM
SIGNALS
CACHE_MEM_SM
SEL_CACHE
CA-MEM-REQ
CA-MEM-WRITE
UMC-O-CA-NEXTADD
UMC-O-CA-READY
UMC

FIG. 20

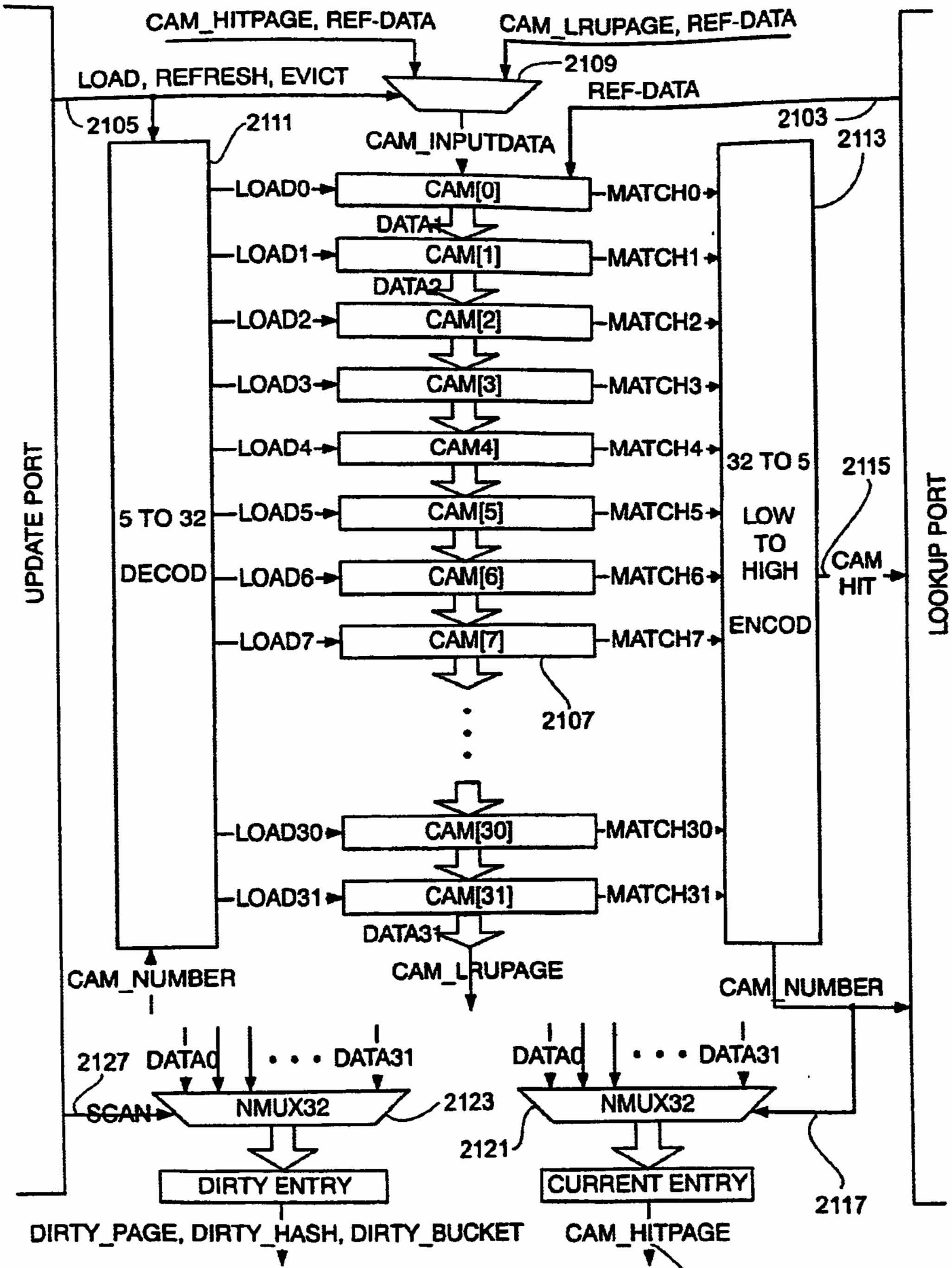

CAM_HITPAGE, REF-DATA
CAM_LRUPAGE, REF-DATA
LOAD, REFRESH, EVICT
2109
REF-DATA
2105
2111
CAM_INPUTDATA
2103
2113
UPDATE PORT
5 TO 32
DECOD
LOAD0
LOAD1
LOAD2
LOAD3
LOAD4
LOAD5
LOAD6
LOAD7
LOAD30
LOAD31
CAM[0]
DATA1
CAM[1]
DATA2
CAM[2]
CAM[3]
CAM4]
CAM[5]
CAM[6]
CAM[7]
CAM[30]
CAM[31]
DATA31
MATCH0
MATCH1
MATCH2
MATCH3
MATCH4
MATCH5
MATCH6
MATCH7
MATCH30
MATCH31
32 TO 5
LOW
TO
HIGH
ENCOD
2115
CAM
HIT
LOOKUP PORT
2107
CAM_NUMBER
CAM_LRUPAGE
CAM_NUMBER
2127
DATA0
DATA31
SCAN
NMUX32
2123
DATA0
DATA31
NMUX32
2121
DIRTY ENTRY
CURRENT ENTRY
2117
DIRTY_PAGE, DIRTY_HASH, DIRTY_BUCKET
CAM_HITPAGE
2119

FIG. 21

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 Ser. No.: 60/141,903 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK to inventors Dietz, et al., filed Jun. 30, 1999, the contents of which are incorporated herein by reference.

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

U.S. Pat. No. 6,651,099 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al., and incorporated herein by reference.

U.S. Pat. No. 6,665,725 for PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE, to inventors Koppenhaver, et al.,-filed and incorporated herein by reference.

U.S. patent application Ser. No. 09/608,126 for RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING, to inventors Dietz, et al., filed and incorporated herein by reference.

U.S. patent application Ser. No. 09/608,267 for STATE PROCESSOR FOR PATTERN MATCHING IN A NETWORK MONITOR DEVICE, to inventors Sarkissian, et al., 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. Pat. No. 6,51,099 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al, 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.

The flow signature of the packet, the hash and at least some of the payload are passed to an analyzer subsystem. In a hardware embodiment, the analyzer subsystem includes a unified flow key buffer (UFKB) for receiving parts of packets from the parser subsystem and for storing signatures in process, a lookup/update engine (LUE) to lookup a database of flow records for previously encountered conversational flows to determine whether a signature is from an existing flow, a state processor (SP) for performing state processing, a flow insertion and deletion engine (FIDE) for inserting new flows into the database of flows, a memory for storing the database of flows, and a cache for speeding up access to the memory containing the flow database. The LUE, SP, and FIDE are all coupled to the UFKB, and to the cache.

Each flow-entry includes one or more statistical measures, e.g., the packet count related to the flow, the time of arrival of a packet, the time differential.

In the preferred hardware embodiment, each of the LUE, state processor, and FIDE operate independently from the other two engines. The state processor performs one or more operations specific to the state of the flow.

Because of the high speed that packets may be entering the system, it is desirable to maximize the hit rate in a cache system. Typical prior-art cache systems are used to expediting 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 lookahead mechanisms are not available for a cache subsystem for the packet monitoring application. When a new packet enters the monitor, the next cache access, for example from the lookup engine, may be for a totally different conversational flow than the last cache lookup, and there is no way ahead of time of knowing what flow the next packet will belong to.

Thus there is a need in the art for a cache subsystem suitable for use in a packet monitor. One desirable property of such a cache system is a least recently used (LRU) replacement policy that replaces the 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 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 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 (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 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.

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 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.

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 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 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.

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 of the invention.

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

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 tha provides a framework for design of communication protocol layers. This model, shown in table form below, serves as a basic reference for understanding the functionaly 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, etc.
4	Transport	TCP, Novel SPX, UDP, etc.
3	Network	IP, Novell IPX, VIP, AppleTalk, etc.
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 layerd 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 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.

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 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.

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 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 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.

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 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 conversational flow, and more packets are necessary to identify the associated application program. In such a case, a subsequent packet of a second type—buut 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 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.

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 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 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 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 **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 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 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 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.

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.

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 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 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 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 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 if 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.

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.

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 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 state to the next state according to a set of state instructions obtained form 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 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 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 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 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 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 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 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 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 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 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 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 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 (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 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 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 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 is 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 (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 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 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.

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 **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.

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 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 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 **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 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 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.

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 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 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 **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** 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 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 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 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 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 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 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 a find the instructions in the extraction operations database memory (i.e., slicer instruction database) **1002**.

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 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 **1 108** 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 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 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 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 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 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.

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 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 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 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 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.

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 in set in the flow key of the UFKB record, a protocol 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 form 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 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-or-ing them together.

Thus, after the UFKB sets the program counter, a sequence of one or more state operations are be 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 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 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 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 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 **1108** can now obtain 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 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 (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 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 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 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 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.

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 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 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 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 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 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 **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 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 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 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 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.

Sun-RPC is the implementation by Sun Microsystems, Inc. (Palo Alto, Calif.) 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.

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, 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.

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.

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).

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.

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'.

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'.

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 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.

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.

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-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 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 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 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 "C_1". If such address identifies the server **110** (shown also as server **204**), the label used in the drawing is "S_1". The first two fields **214** and **215** in packet **206** are "S_1" and "C_1" because packet **206** is provided from the server **110** and is destined for the client **106**. Suppose for this example, "S_1" is an address numerically less than address "C_1". A third field "p^1" **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 "i_1" **219** is an element that is being requested by the client from the server. A seventh field denoted "s_1a" **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 "s_1a". 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**.

Packet **207** includes ten fields **224**–**233**. The destination and source addresses are carried in fields **224** and **225**, e.g., indicated "C_1" and "S_1", respectively. Notice the order is now reversed, since the client-server message direction is from the server **110** to the client **106**. The protocol "p^1" is used as indicated in field **226**. The request "i^1" is in field **229**. Values have been filled in for the application port number, e.g., in field **233** and protocol "p^2" 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 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 operations. In this case, therefore, since we assume "S_1"<"C_1", the order is address "S_1" followed by client address "C_1". 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 "p^1". 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 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 p_1 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 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 "a^1" (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 "S_1" followed by (the numerically higher address) client "C_1". A protocol field **252** defines the protocol to be used, e.g., "p^2" 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 "a^2". 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 "st^1". 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 "a^2". 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 "S_1" followed by client "C_1". 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 Sand C_1, in a pair of fields **260** and **261**. A field **262** defines the protocol as "p^2", 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.

Thus the flow signature for the recognition of application "a^2" 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 complex state transition diagram to be traversed before a "final" resting state such as "st_1" 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 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.

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 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.

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 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.

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 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 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 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.

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**.

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 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 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 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 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 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, . . . , 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 $hash_4$, and a lookup with $hash_4$ will produce a match and the address $page_4$ in cache memory. Furthermore, $page_4$ in cache memory contains the flow-entry, $entry_4$, that in this notation is the flow-entry matching hash value $hash_4$. This table also indicates that $hash_0$ was more recently used than $hash_1$, $hash_5$ more recently than $hash_2$, and so forth, with $hash_{31}$ 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 $hash_{31}$. The LUE looks up the cache subsystem via the CAM array. CAM_{31} gets a hit and returns the page number of the hit, i.e., $page_{31}$. The cache subsystem now indicates to the LUE **1007** that the supplied hash value produced a hit and provides a pointer to $page_{31}$ of the cache memory which contains the flow-entry corresponding to $hash_{31}$, i.e., $flow_{31}$. The LUE now retrieve the flow-entry $flow_{31}$ from the cache memory at address $page_{31}$. In the preferred embodiment, the lookup of the cache takes only one clock cycle.

The value $hash_{31}$ 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 $hash_{31}$ is put into CAM_0 (pointing to $page_{31}$). Furthermore, $hash_{30}$ is now the LRU hash value, so is moved to CAM_{31}. The next least recently used hash value, $hash_{29}$ is now moved to CAM_{30}, 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 $hash_5$. This produces a hit in CAM_6 pointing to $page_5$ 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 $hash_5$, so $hash_5$ and cache memory address $page_6$ are entered into CAM_0. The contents of the remaining CAMs are all shifted down one up to and including the entry that contained $hash_5$. That is, CAM_7, CAM_8, . . . , CAM_{31} 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 $hash_{43}$. 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 $hash_{43}$. 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—$entry_{43}$—in database **324** matching hash value $hash_{43}$. 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 CAM_{31} 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 $hash_{43}$ is put in CAM_{31} and flow-entry $entry_{43}$ is placed in the cache page pointed to by CAM **31**. The CAM array and 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 CAM_{31} are now moved to CAM_0 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 CAM_{31} are now moved to CAM_0 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, $hash_{43}$ is in MRU CAM_0, CAM_0 correctly points to $page_{30}$, but the information in $page_{30}$ of the cache, $entry_{43}$, does not correspond to $entry_{43}$ in database **324**. That is, the contents of cache page $page_{30}$ 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 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.

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 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:

(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**.

(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 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 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 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 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 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 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 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 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.

What is claimed is:

1. A packet monitor for examining packet 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 flow-entries for previously encountered conversational flows to which a received packet may belong, a conversational flow being an exchange of one or more packets in any direction as a result of an activity corresponding to the flow;

(c) a cache subsystem coupled to the flow-entry database memory providing for fast access of flow-entries from the flow-entry database;

(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, to looking up being the cache subsystem; and

(e) a state processor coupled to the lookup engine and to the flow-entry-database memory, the state processor being to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

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 a 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 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 address and pointer contents of the CAMs above the CAM that produced the asserted match output arc 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 packet monitor for examining packet passing through a connection point on a computer network, each packets conforming to one or more protocols, the monitor comprising:

a packet acquisition device coupled to the connection point and configured to receive packets passing through the connection point;

an input buffer memory coupled to and configured to accept a packet from the packet acquisition device;

a parser subsystem coupled to the input buffer memory, the parsing subsystem configured to extract selected portions of the accepted packet and to output a parser record containing the selected portions;

a memory to storing a database of one or more flow-entries for any previously encountered conversational flows, each flow-entry identified by identifying information stored in the flow-entry;

a lookup engine coupled to the output of the parser subsystem and to the flow-entry memory and configured to lookup whether the particular packet whose parser record is output by the parser subsystem has a matching flow-entry, the looking up using at least some of the selected packet portions and determining if the packet is of an existing flow;

a cache subsystem coupled to and between the lookup engine and the flow-entry database memory providing for fast access of a set of likely-to-be-accessed flow-entries from the flow-entry database; and

a flow insertion engine coupled to the flow-entry memory and to the lookup engine and configured to create a flow-entry in the flow-entry database, the flow-entry including identifying information for future packets to be identified with the new flow-entry,

the lookup engine configured such that if the packet is of an existing flow, the monitor classifies the packet as belonging to the found existing flow; and if the packet is of a new flow, the flow insertion engine stores a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the operation of the parser subsystem depends on one or more of the protocols to which the packet conforms.

8. A monitor according to claim 7, wherein the lookup engine updates the flow-entry of an existing flow in the case that the lookup is successful.

9. A monitor according to claim 7, further including a mechanism for building a hash from the selected portions, wherein the hash is included in the input for a particular packet to the lookup engine, and wherein the hash is used by the lookup engine to search the flow-entry database.

10. A monitor according to claim 7, further including a memory containing a database of parsing/extraction operations, the parsing/extraction database memory coupled to the parser subsystem, wherein the parsing/extraction operations are according to one or more parsing/extraction operations looked up from the parsing/extraction database.

11. A monitor according to claim 10, wherein the database 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.

12. A method according to claim 7, further including a state processor coupled to the lookup engine and to the flow-entry-database memory, and configured to perform any state operations specified for the state of the flow starting from the last encountered state of the flow in the case that the packet is from an existing flow, and to perform any state operations required for the initial state of the new flow in the case that the packet is from an existing flow.

13. A method according to claim 12, wherein the set of possible state operations that the state processor is configured to perform includes searching for one or more patterns in the packet portions.

14. A monitor according to claim 12, wherein the state processor is programmable, the monitor further including a state patterns/operations memory coupled to the state processor, the state operations memory configured to store a database of protocol dependent state patterns/operations.

15. A monitor according to claim 12, wherein the state operations include updating the flow-entry, including identifying information for future packets to be identified with the flow-entry.

16. A method of examining packets passing through a connection point on a computer network, each packets conforming to one or more protocols, the method comprising:

(a) receiving a packet from a packet acquisition device;

(b) performing one or more parsing/extraction operations on the packet to create a parser record comprising a function of selected portions of the packet;

(c) looking up a flow-entry database comprising none or more flow-entries for previously encountered conversational flows, the looking up using at least some of the selected packet portions and determining if the packet is of an existing flow, the lookup being via a cache;

(d) if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

(e) if the packet is of a new flow, storing a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry,

wherein the parsing/extraction operations depend on one or more of the protocols to which the packet conforms.

17. A method according to claim 16, wherein classifying the packet as belonging to the found existing flow includes updating the flow-entry of the existing flow.

18. A method according to claim 16, wherein the function of the selected portions of the packet forms a signature that includes the selected packet portions and that can identify future packets, wherein the lookup operation uses the signature and wherein the identifying information stored in the new or updated flow-entry is a signature for identifying future packets.

19. A method according to claim 16, wherein the looking up of the flow-entry database uses a hash of the selected packet portions.

20. A method according to claim 16, wherein step (d) includes if the packet is of an existing flow, obtaining the last encountered state of the flow and performing any state operations specified for the state of the flow starting from the last encountered state of the flow; and wherein step (e) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow.

* * * * *

☆ U.S. GOVERNMENT [illegible] 1997—424-353

PATENT NUMBER

ORIGINAL CLASSIFICATION

CLASS	SUBCLASS
452	117

APPLICATION SERIAL NUMBER

08/780,248

APPLICANT'S NAME (PLEASE PRINT)

HAZENBROEK, ET AL

IF REISSUE, ORIGINAL PATENT NUMBER

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)			

INTERNATIONAL CLASSIFICATION

A	2	2	C		21/06
					/
					/
					/

GROUP ART UNIT: 3616

ASSISTANT EXAMINER (PLEASE STAMP OR PRINT FULL NAME)

PRIMARY EXAMINER (PLEASE STAMP OR PRINT FULL NAME)

WILLIS LITTLE
PRIMARY EXAMINER
ART UNIT 3616

PTO 270
(REV. 5-91)

ISSUE CLASSIFICATION SLIP

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION	DT		2-2-09
O.I.P.E. CLASSIFIER			
FORMALITY REVIEW	LL	75353	9/1/10
RESPONSE FORMALITY REVIEW			

LL 11/2/10

INDEX OF CLAIMS

✓ .... Rejected
= .... Allowed
— (Through numeral)... Canceled
÷ .... Restricted
N .... Non-elected
I .... Interference
A .... Appeal
O .... Objected

FINAL

Claim Final	Claim Original	Date 9/2/09	Date 4/19/10
1	1	✓	=
2	2	✓	=
3	3	✓	=
4	4	✓	=
5	5	✓	=
6	6	✓	=
	7	✓	
	8	✓	
	9	✓	
	10	✓	
	11	✓	
	12	✓	
	13	✓	
	14	✓	
	15	✓	
	16	✓	
	17	✓	
	18	✓	
	19	✓	
	20	✓	
7	21		=
8	22		=
9	23		=
10	24		=
11	25		=
12	26		=
13	27		=
14	28		=
15	29		=
16	30		=
17	31		=
18	32		=
19	33		=
20	34		=
	35		
	36		
	37		
	38		
	39		
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		
	48		
	49		
	50		

Claim Final	Claim Original	Claim Final	Claim Original
	51		101
	52		102
	53		103
	54		104
	55		105
	56		106
	57		107
	58		108
	59		109
	60		110
	61		111
	62		112
	63		113
	64		114
	65		115
	66		116
	67		117
	68		118
	69		119
	70		120
	71		121
	72		122
	73		123
	74		124
	75		125
	76		126
	77		127
	78		128
	79		129
	80		130
	81		131
	82		132
	83		133
	84		134
	85		135
	86		136
	87		137
	88		138
	89		139
	90		140
	91		141
	92		142
	93		143
	94		144
	95		145
	96		146
	97		147
	98		148
	99		149
	100		150

If more than 150 claims or 10 actions staple additional sheet here

(LEFT INSIDE)

SEARCHED

Class	Sub.	Date	Exmr.
370	241.1	8/27/03	AN
|	252	|	|
	253		
709	223	|	|
|	224	|	|
370	352	4/14/04	AN
|	353	|	|
	355		
	381		
	392		
	401		
	395.1		
Update	Search	4/14/04	AN

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.
370	241.1, 252, 253, 352, 353, 355, 384, 392, 401, 395.1	4/14/04	AN
709	223, 224	|	|

SEARCH NOTES
(INCLUDING SEARCH STRATEGY)

	Date	Exmr.
EAST PTO EPO JPO	8/28/03	AN
IEEE	|	|
Consultation with SPE	8/28/03 8/29/03	|
Ricky Ngo		

(RIGHT OUTSIDE)

PATENT APPLICATION

09608266

JC836 U.S. PTO
09/608266
06/30/00

INITIALS

JUL 17 0004 5

CONTENTS

		Date Received (Incl. C. of M.) or Date Mailed			Date Received (Incl. C. of M.) or Date Mailed
1.	Application papers.		42.		
2.	Ltr on Dec, Fee & Drawings	9/11/00	43.		
3.	Dec & Fees	10/24/00	44.		
4.	IDS w/References	4-11-01	45.		
5.	Prior Art	4-12-02	46.		
6.	Rej.	9/10/03	47.		
7.	Ext of Time 2, Amdt A	2/10/04, 2/10/04	48.		
8.	Suppl. Amdt B	2/26/04	49.		
9.	IDS	3/8/04	50.		
10.	Ex Amdt/C	4/20/04	51.		
11.	Formal Drawings (2 / shts) set	6/4/04	52.		
12.	Req for CC	8/13/04	53.		
13.	Req for CC	8/30/04	54.		
14.	Petition to Correct Inventorship under 1.324	11/26/2007	55.		
15.	Petition Dismissed	10/4/2012	56.		
16.	Request for C of C	9/14/2013	57.		
17.	Petition to Correct Inventorship under 1.324	8/19/2014	58.		
18.	Petition Granted	10/2/2014	59.		
19.			60.		
20.			61.		
21.			62.		
22.			63.		
23.			64.		
24.			65.		
25.			66.		
26.			67.		
27.			68.		
28.			69.		
29.			70.		
30.			71.		
31.			72.		
32.			73.		
33.			74.		
34.			75.		
35.			76.		
36.			77.		
37.			78.		
38.			79.		
39.			80.		
40.			81.		
41.			82.		

(LEFT OUTSIDE)