Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2240616 C 2002/01/29

(11)(21) 2 240 616

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 1998/06/11

(41) Mise à la disp. pub./Open to Public Insp.: 1998/12/12

(45) Date de délivrance/Issue Date: 2002/01/29

(30) Priorité/Priority: 1997/06/12 (873,783) US

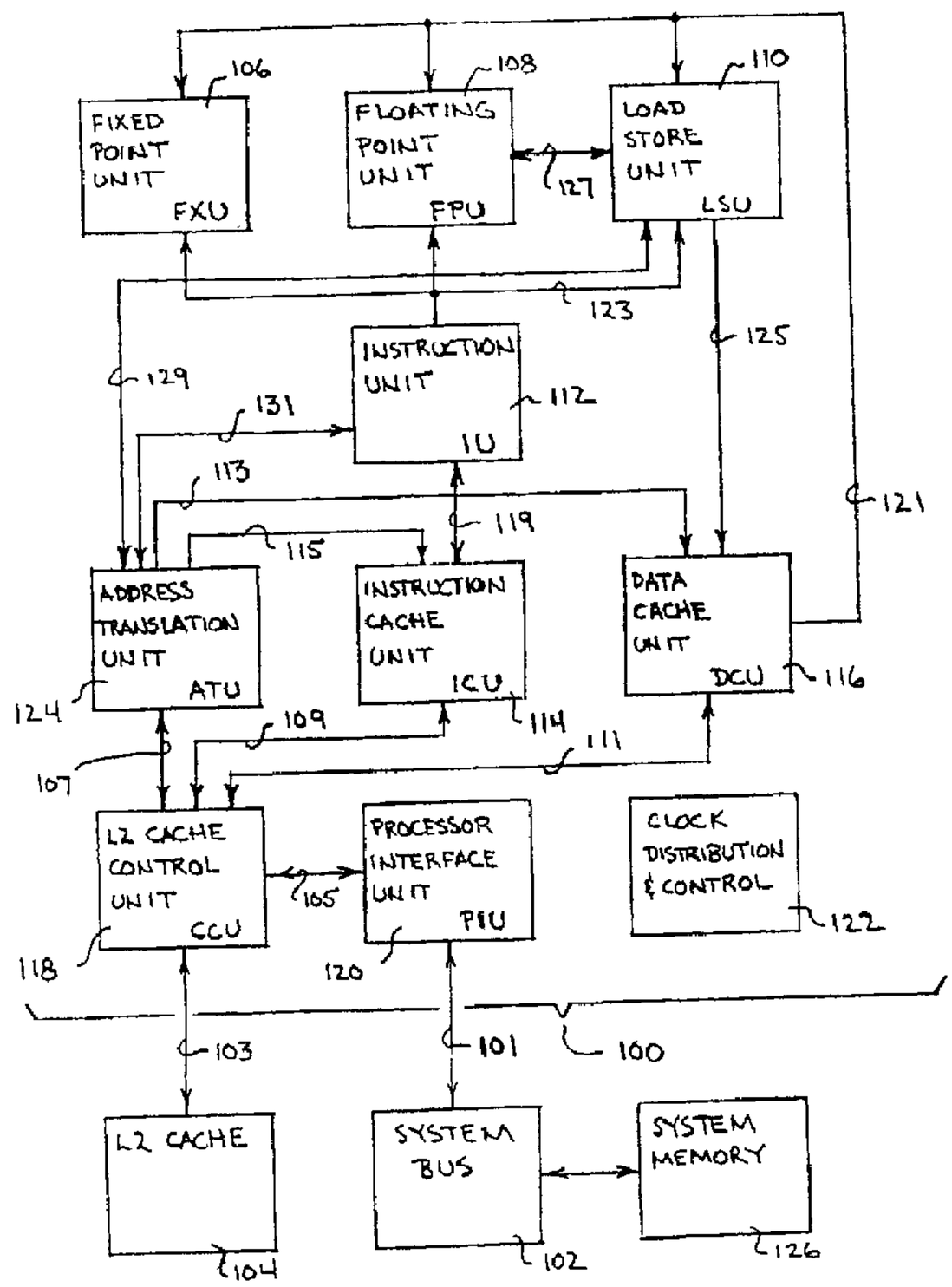
(51) Cl.Int.^{6}/Int.Cl.^{6} G06F 12/10, G06F 13/20

(72) Inventeurs/Inventors:
JEREMIAH, Thomas Leo, US;
EICKEMEYER, Richard James, US

(73) Propriétaire/Owner:
INTERNATIONAL BUSINESS MACHINES CORPORATION, US

(74) Agent: SAUNDERS, RAYMOND H.

(54) Titre : PRODUCTION D'ADRESSES DE CACHE
(54) Title: CACHE ADDRESS GENERATION



(57) Abrégé/Abstract:
A cache system provides for accessing set associative caches with no increase in critical path delay, for reducing the latency penalty for cache accesses, for reducing snoop busy time, and for responding to MRU misses and cache misses. The cache

CA 2240616 C 2002/01/29

(11)(21) 2 240 616

(13) C

(57) Abrégé(suite)/Abstract(continued):
array is accessed by multiplexing two most-recently-used (MRU) arrays which are addressed and accessed substantially in parallel with effective address generation, the outputs of which MRU arrays are generated, one by assuming a carryin of zero, and the other by assuming a carryin of one to the least significant bit of the portion of the effective addressed used to access the MRU arrays. The hit rate in the MRU array is improved by hashing within an adder the adder's input operands with predetermined additional operand bits.

CACHE ADDRESS GENERATION

Abstract of the Disclosure

A cache system provides for accessing set associative caches with no increase in critical path delay, for reducing the latency penalty for cache accesses, for reducing snoop busy time, and for responding to MRU misses and cache misses. The cache array is accessed by multiplexing two most-recently-used (MRU) arrays which are addressed and accessed substantially in parallel with effective address generation, the outputs of which MRU arrays are generated, one by assuming a carryin of zero, and the other by assuming a carryin of one to the least significant bit of the portion of the effective addressed used to access the MRU arrays. The hit rate in the MRU array is improved by hashing within an adder the adder's input operands with predetermined additional operand bits.

CACHE ADDRESS GENERATION

Background of the Invention

Technical Field of the Invention

This invention generally relates to caches for computer systems, such as set associative caches and direct-mapped caches, and more particularly to addressing such caches with minimal increase in critical path delay.

Background Art

The use of caches for performance improvements in computing systems is well known and extensively used. See, for example, U.S. Patent 5,418,922 by L. Liu for "History Table for Set Prediction for Accessing a Set Associative Cache", and U.S. Patent 5,392,410 by L. Liu for "History Table for Prediction of Virtual Address Translation for Cache Access".

A cache is a high speed buffer which holds recently used memory data. Due to the locality of references nature for programs, most of the access of data may be accomplished in a cache, in which case slower accessing to bulk memory can be avoided.

In typical high performance processor designs, the cache access path forms a critical path. That is, the cycle time of the processor is affected by how fast cache accessing can be carried out.

A cache may logically be viewed as a table of data blocks or data lines in which each table entry covers a particular block or line of memory data. (Hereinafter the storage unit for a cache may be referred to as a line or a block.) The implementation of a cache is normally accomplished through

three major portions: directory, arrays and control. The directory contains the address identifiers for the cache line entries, plus other necessary status tags suitable for particular implementations. The arrays (sometimes called cache memory herein) store the actual data bits, with additional bits for parity checking or for error correction as required in particular implementations.

Cache control circuits provide necessary logic for the management of cache contents and accessing. Upon an access to the cache, the directory is accessed or "looked up" to identify the residence of the requested data line. A cache hit results if it is found in the cache, and a cache miss results otherwise. Upon a cache hit, the data may be accessed from the array if there is no prohibiting condition (e.g., protection violation.) Upon a cache miss, the data line is normally fetched from the bulk memory and inserted into the cache first, with the directory updated accordingly, in order to satisfy the access through the cache.

Since a cache only has capacity for a limited number of line entries and is relatively small compared with the bulk memory, replacement of existing line entries is often needed. The replacement of cache entries in a set associative cache is normally based on algorithms like the Least-Recently-Used (LRU) scheme. That is, when a cache line entry needs to be removed to make room for (be replaced by) a new line, the line entry that was least recently accessed will be selected.

In order to facilitate efficient implementations, a cache is normally structured as a 2-dimensional table. The number of columns is called the set-associativity, and each row is called a congruence class. (This row/column designation is traditional. However, the Liu patents, supra, rotate the cache 90 degrees and interchage the row and column designations.) For each data access, a congruence class is selected using certain address bits of the access, and the data may be accessed at one of the line entries in the selected congruence class if it hits there. It is usually too slow to have the cache directory searched first (with parallel address compares) to identify the set position (within the associated congruence class) and then to have the data accessed from the arrays at the found

location. Such sequential processing normally requires two successive machine cycles to perform, which degrades processor performance significantly.

Another aspect that complicates cache design is the commonly employed virtual addressing architecture in computer systems. In a virtual memory system (e.g., the IBM System/390™ architecture) each user process may have the view that it has its own virtual address space. Upon execution of programs the operating system may dynamically allocate real memory pages (e.g., 4 kilobytes per page) to more actively accessed virtual address pages. When a page accessed from a program does not have a real memory page allocated for it, an exception (page fault) condition will occur and trigger the operating system to properly allocate a real memory page frame. Page fault processing is normally associated with a very high performance overhead and often requires accessing data from slower backing devices like disks. However, due to the strong nature of program locality, a reasonable operating system can maintain a very low page fault rate during program executions. The operating system normally maintains the real page allocation information in architecturally specific software tables. Typically a 2-level translation table structure with segment and page tables is used for this purpose. Each program space has its own segment table, in which each entry points to a page table. At a page table each entry records the real page allocation information, plus some other status tags needed for particular architectures.

The operating system manages such translation tables according to its design algorithms. One consequence of the employment of virtual addressing is that the same virtual page address from different program address spaces may not be logically related and may be allocated at different real page frames in the storage. Furthermore, in architectures like that of the IBM System/390, the same real page frame may be accessed through different virtual addresses from different programs or processors.

With all these architectural requirements, most systems require a step called virtual address translation for processing storage accesses from processors. Virtual address translation translates

a virtual page address into a real page address. A page fault exception is triggered if the real page frame is not allocated, for which the operating system will update the translation information when allocation is complete and then allow the faulted program to resume execution.

In most modern systems, hardware facilities are used for speeding up the virtual address translation process. Typically a translation lookaside buffer (TLB) is employed for each processor. A TLB is a hardware directory table that records the translation information for actively accessed virtual pages. Due to the locality nature of program addressing, a relatively small TLB can capture the translation information for a great majority of storage accesses from a processor. Only upon a TLB miss condition, when the TLB cannot cover the particular storage access, will a slower translation, using the translation tables from memory, be activated. For efficiency in hardware implementation, a TLB is normally structured as a set-associative table like a cache directory. For a given virtual page address (including certain program space identifiers), the hardware uses certain address bits (and other information specific to a particular design) to derive a congruence class. Within the congruence class, the hardware performs a parallel search of the entries and identifies the results of translation.

In many processor designs, a storage access needs to go through TLB translation prior to the final resolution of a cache access. In a set associative design, the TLB look-up is carried out in parallel with the cache directory search, with the results merged for final late-select of the array data. This requirement of multiple directory searches for the resolution of cache accessing is a source of complexity for the optimization of many cache designs. When a storage processor issues a storage access request, the cache array cannot determine the exact location of data without knowing the results of conventional directory lookups, which produces signal delay. Such exact location for data access cannot be obtained easily. As a result of conventional late-select mechanisms, wasteful array I/O's must be used to retrieve multiple data units for the possible final selection of at most one data unit for the processor's use. Consequently, concurrent multiple, independent cache accesses in very high performance computers are difficult to support. See, for example, the design of the

IBM/3090™ 4-way set-associative cache with 128 congruence classes as described in Liu patent 5,418,922, which requires several comparators to resolve synonym conditions efficiently, and a long cache access path due to the waiting of late-select for directory read/compare results.

Cache designs that employ prediction methods to avoid these inefficiencies of the prior art have been proposed. These include a direct-map approach and a MRU-cache approach.

The direct-map design is one with a set-associativity of 1, with only one line entry at each cache congruence class. The direct-map design achieves ultimate simplicity and prediction accuracy by flattening the physical cache geometry to a 1-dimensional structure, but in so doing usually increases cache misses.

The MRU-cache approach manages cache line replacement and set selection on a per congruence class basis. Within each congruence class, there is a most-recently-used (MRU) entry and a least-recently-used (LRU) entry. Due to program locality, a cache access is most likely to hit to the MRU line entry. The LRU entry is the one chosen for replacement when a new line needs to be inserted into the congruence class. Thus, typically, the MRU handles set selection, and the LRU handles line replacement.

The slot MRU-cache approach logically views the MRU lines of the whole cache as a direct-map cache: whenever a congruence class is determined to be accessed, the data will be retrieved from the MRU line entry on a prediction basis. It is a real-address cache, in which a missed line will be inserted into the congruence class associated with the real address bits (after translation.)

For a given virtual address, the real address bits at the MRU entry of the associated TLB congruence class are read out and passed to the array control before the real address is verified as the actual translation. The predicted real address bits from the TLB are used by the array control to select the predicted congruence class. The MRU-cache approach requires that the predictions for

TLB and cache be based on the physical MRU entries, and hence loses accuracy and causes difficulties in implementation.

Liu U. S. Patent 5,418,922 describes set prediction and Liu U. S. Patent 5,392,410 describes a system using real address translation prediction for shortening the cache access critical path. A history table is used to predict virtual address translation information for a given access in a set-associative cache. When the prediction is correct, the performance is the same as if the location is known. In the case of a wrong prediction, the data access is aborted and reissued properly. However, Liu does not teach a mechanism for generating the address used to access the history table, and there is a need for a method to create that address with a minimum circuit delay.

In U. S. Patent 5,392,410, Liu also describes a hashing function, which occurs after computation of the address, to randomize the entries in the history table. As this introduces an extra delay in the critical path, there is a need for a method that performs the hashing function which avoids that delay.

It is an object of the invention to improve cache operation by completely overlapping real address prediction with effective address generation.

It is a further object of the invention to improve cache operation by performing a hashing operation to randomize entries in the MRU without causing delay in the critical path.

Summary of the Invention

In accordance with the invention, an apparatus and method is provided for accessing a cache array by multiplexing two most-recently-used (MRU) arrays which are addressed and accessed substantially in parallel with effective address generation, the outputs of which MRU arrays are

generated, one by assuming a carryin of zero, and the other by assuming a carryin of one to the least significant bit of the portion of the effective address used to access the MRU arrays.

In accordance with a further aspect of the invention, the hit rate in the MRU array is improved by hashing within an adder the adder's input operands with predetermined additional operand bits.

Other features and advantages of this invention will become apparent from the following detailed description of the presently preferred embodiment of the invention, taken in conjunction with the accompanying drawings.

Brief Description of the Drawings

Figure 1 is a block diagram illustrating a typical microprocessor architecture within which a preferred embodiment of the invention is implemented.

Figure 2 illustrates how Figure 2A through 2C relate, while the latter are block diagrams showing the implementation of a preferred embodiment of the invention within the microprocessor of Figure 1.

Figures 3-6 are block diagrams illustrating the system and L2 cache bus interfaces 101 and 103 of Figure 1, with Figure 3 generally illustrating the system data bus; Figure 4, the system bus controls; Figure 5, the L2 cache data bus; and Figure 6, the L2 cache controls.

Figure 7 is a block diagram showing a prior art design for providing full virtual address translation capability.

Figure 8 illustrates the manner in which Figures 8A and 8B relate, with the latter being a block diagram showing the invention as an improvement on the prior art design of Figure 7.

Figure 9 is an example syntax for a memory address.

Detailed Description of the Invention

Part 1

Referring to Figure 1, the microprocessor architecture within which a preferred embodiment of the invention is implemented will be described.

Microprocessor chip 100 is organized to interface system bus 102 and L2 cache 104, and includes the following functional units: fixed point unit (FXU) 106, floating point unit (FPU) 108, load store unit (LSU) 110 , instruction unit (IU) 112, instruction cache unit (ICU) 114, data cache unit (DCU) 116, L2 cache control unit 118, processor interface unit (PIU) 120, clock distribution and control 122, and address translation unit (ATU) 124. In a multiprocessor environment, several processors 100 and their associated L2 caches 104 may interface system bus 102 over buses equivalent to bus 101, and share access through system bus 102 to main memory (sometimes referred to as L3 memory) 126.

The various functional units of microprocessor 100 interface over data, address, and/or control I/O pins, lines and/or busses as will be described hereafter. When referring to a figure, "line" can refer to either a single signal line or a collection of signal lines (i.e., a bus). Those functional units most pertinent to the invention, and which will be described in greater detail hereafter, include the load/store unit (LSU) 110, the data cache unit (DCU) 116, the L2 cache control unit (CCU) 118, and the address translation unit (ATU) 124.

In broad overview, the functional units on chip 100 communicate as follows. Clock distribution and control 122 provides clocking signals to all functional units on microprocessor chip 100. System bus 102 interfaces to PIU 120 over bidirectional bus 101, and thence over buses 105 with CCU 118. L2 cache 104 communicates with CCU 118 over buses 103. CCU 118 communicates instructions with ICU 114 over buses 109, with DCU 116 over buses 111, and provides address information to ATU 124 and receives miss interface signals over buses 107. LSU 110 and IU 112 provide request interfaces to ATU 124 and receive translation state information over lines 129 and 131. ATU 124 provides translated addresses to ICU 114 over lines 115, and to DCU 116 over lines 113. ICU 114 interfaces to instruction unit 112 over bus 119. DCU 116 provides data to FXU 106, FPU 108 and LSU 110 over bus 121, and IU 112 provides instructions to FXU 106, FPU 108 and LSU 110 over bus 123. LSU 110 provides data to DCU 116 over bus 125. FPU 108 provides and receives data to DCU 116 over buses 127 to LSU 110, then across buses 125. Processor 100 accesses main memory through system bus 102.

Microprocessor Core 100

Referring to Figures 2A through 2C, and Figures 3-6, the core of microprocessor 100 will be described. Figure 2A generally corresponds to load/store unit (LSU) 110, Figure 2B to address translation unit (ATU) 124, and Figure 2C to data cache unit (DCU) 116. Figures 3-6 generally correspond to L2 cache control unit (CCU) 118 and processor interface unit (PIU) 120.

Dispatch block 300 directs instructions from instruction unit 112 to the DECODE stage buffers of the various execution units 106, 108, 110, including on bus 301 (which is that portion of buses 123 directed to LSU 110) to LSU pipeline buffer 302.

The function of load/store unit 110 is to generate effective addresses on 64 bit wide bus 313 for load and store instructions and to serve as a source and sink for GPR data. During writes to cache 400, registers 314 and 316 hold the data and address, respectively; the effective address is on

bus 313, and data select block 320 puts the data out on bus 323. During cache reads, data from cache 400 comes in on line 461, is latched in register 330, and from there sent on line 333 to general purpose registers 306 or to fixed point unit 106.

The output of LSU pipeline buffer 302 is fed on line 303 to the LSU decode and address generation block AGEN 304, which contains the general purpose registers 306 and address generation adders. The data output of decode block 304 is fed on lines 311 to data register 314 and thence on line 319 to data select block 320. The address output of AGEN 304 is fed on lines 313 to EXECUTE stage buffer 316, and on bus 309 to real address MRU 430. AGEN 304 output also includes control line 307, which it sets to indicate either real or virtual mode addressing to data cache control block 470.

The outputs of buffer 316 are fed on lines 317 to data select block 320 and to data cache address register 408, DIR address register 414 and register slot MRU address register 406. The output of register 408 is fed on line 409 to multiplexer 412. Data select block 320 contains the data to be stored to data cache 400 from load store unit 110, and this is fed thereto on store data output lines 323 via multiplexer 432, lines 433, align block 460, lines 461, register 456, lines 457, and line 427 via multiplexer 426. Data select block 320 also provides control signals to data cache controller 470 on lines 321. The other inputs to multiplexer 432 are (1) L2 corrected data 609 via multiplexer 426 and line 427, which is also fed to data cache 400, (2) bypass data to DC on line 621, and (3) unaligned data (aka store merging and correction) register 452 via lines 453 to line 427 via multiplexer 426. Multiplexer 432 output line 433 is also fed via align block 460 and line 461 to register 456 and thence via multiplexer 424 to L2 cache controller on line 425, along with the output of castout buffer 450 on line 451. Align block 460 is, in this embodiment, a barrel rotator or shifter which aligns D cache 400 data to quad word boundaries on reads, and from multiplexer 432 to quad word boundaries on stores.

An effective address from instruction unit 112 on line 367 (a portion of lines 131) is latched in register 364 and fed on line 365 to ITLB 358 and to the compare and address select block 356 at ISLB 354. Line 313 from AGEN 304 is latched in register 384, and fed on line 385 to DTLB array 378 and compare and address select block 374 at DSLB 376. In this preferred embodiment, DTLB 378 may be a standard design, such as that described by Liu, supra. Whereas the Liu TLB design is 32 bits wide, in this preferred embodiment a 64 bit wide TLB 378 is used.

Data select 320 output on line 325 is fed to PUTAWAY stage buffer 330, which also receives data on lines 461 from data cache 400 (via lines 401 and align block 460) for LSU 110, and FPU 108 results on line 327 which is a portion of bus 127. The output of PUTAWAY stage buffer 330 is fed on lines 333 to a floating point register in FPU 108, special purpose registers 334 (among which are the timers), and general purpose registers 306. Special purpose registers 334 output line 335 is fed back to data select block 320 which allows the processor to read them. Line 333 carries the data for FPU 108 when doing a fetch from cache 400.

The selected output of instruction segment look aside buffer (ISLB) 354 is fed on lines 355 to comparator 362, along with the virtual address output of ITLB 358 on lines 359. ITLB real address output lines 361 are input to IC controls 350 (which includes instruction directory 352) and status information on line 361 is fed to ATU controls 370. The output of comparator 362 is fed on lines 363 to IC controls 350 and to ATU controls 370. The output of DSLB 376 is fed on lines 377 to comparator 382, along with the output of DTLB 378 on lines 379. The output of comparator 382 is fed on lines 383 to ATU controls 370 and DC control 470. DTLB 378 status output 381 is fed to ATU controls 370 and DC control 470. ATU controls 370 outputs include control lines 369 to data cache controller 470, L2 address 371 and ATU write data 373. IC controls 350 output is L2 address line 351. Real address from DTLB 378 is fed on lines 381 to DC address register 408 and directory address register 414.

Predicted real address MRU 430 output signals on line 431, representing the predicted read address bits 50:51, are latched in registers 410 and 416. The output of data cache address register 410 on line 411 is multiplexed with bits 50:51 of the output of register 408 in multiplexer 412, and its output is fed on address lines 413 to data cache 400. The remaining bits of DC address register 408 are fed straight through on line 413 to data cache 400. Similarly, the output of register 416 is fed on lines 417 to multiplexer 436, where it is multiplexed with bits 50:51 of the output of register 414 on line 415, and the result fed on lines 437 to directory array 440. The output of register 414 on line 415 is also fed to address register 408.

The function of real address MRU 430 is to provide predicted real address bits 50:51 to data cache 400 and directory array 440.

During the fetch stage, data cache 400 output 401 is fed to unaligned data register 452 and align block 460, and thence on line 461 to registers 456 and 330. Line 401 contains the data to be read from data cache 400 by the load store unit 110, snoop data to the L2 cache controller 118, merge data for partial stores to the data cache 400, and castout data to castout buffer 450. Slot MRU 402 output line 403 controls the selection of one of four sets of data to load on bus 401 through a multiplexer (not shown) on the output of data cache 400.

The output of castout buffer 450 is multiplexed in multiplexer 424 with the output of register 452 on lines 453 and line 457 from DC putaway register 456, the output appearing on lines 425 to the L2 cache controller. The output of register 452 along with DC putaway register 456 and L2 corrected data on line 609 is also fed to data cache input multiplexer 426, the output of which appears on lines 427 to data cache 400 and multiplexer 432. The output of register 406 is fed on line 407 to slot MRU 402. Slot MRU 402 output 403 is fed to data cache 400 where it controls a data multiplexer which selects the appropriate cache set (as taught by Liu, supra.)

Data cache (DC) control 470 receives inputs from directory array 440 on lines 441 (signifying a directory array hit or miss), from AGEN 304 on lines 307, data select and execute cycle control block 320 on lines 321, ATU controls 370 on lines 369, and comparator 382 on lines 383. Its output is fed to L2 address line 471, and includes a signal signifying a miss condition. Miss information is also sent to ATU controls 370 and PA controls (not shown) in LSU 110.

The function of data cache control 470 is to control the data flow multiplexing into and out of data cache 400 and send results to the load/store unit 110, address translation unit 124, and L2 cache control unit 118, and also to control writing of data into data cache 400.

Data directory 440 contains address tags to indicate if the contents of the real address are present in cache 400, and the status of the cache lines, whether modified, shared, or invalid. It also contains an LRU pointer for each congruence class, indicating which cache 400 line should be replaced.

Address translation unit (ATU) control 370 handles translations from effective addresses to virtual addresses to real addresses. It receives as inputs L2 corrected data on line 353, and provides TLB reload data output on lines 375 to instruction translation lookaside buffer (ITLB) 358 and data translation lookaside buffer (DTLB) 378, ISLB 354, and DSLB 376. With respect to look aside tables 354, 358, 376, 378, if a miss condition is detected, ATU sequencer 370 requests data (address and length) to L2 cache on bus 371 (Figure 6). When L2 responds on bus 353 (Figure 5), ATU examines the data to select data for look aside buffer 378, 376, 354, 358, as the case may be, or signals a translation exception back to the instruction unit. ATU controls 370 tracks segment and page table updates and sends them to L2 controls on line 371. Line 381 provides the real address to the data cache directory for comparison, the MRU 430 supplies some of the bits.

The effective address is compared in ISLB 354 comparator 356 with the virtual address. If these match, then a valid effective to virtual address translation exists in buffer 354, which transmits the virtual address on line 355 to compare block 362.

ITLB 358 is accessed by an effective address on line 365 from register 364 for doing virtual to real address translation. The address input to ITLB 358 is a portion of the effective address from IU 112 on lines 367. Comparator 362 compares virtual addresses on lines 355 and 359, and signals the result on line 363. Associated with each virtual address in ITLB array 358 is a real address. The signal on line 363 indicates whether or not the address on line 361 is valid.

DTLB 378 is accessed by an address from register 384. Comparator 382 compares data on lines 379 and 377, and signals the result on line 383. The signal on line 383 indicates whether or not the address on line 379 is valid.

System Bus Interface 120

Referring to Figures 3 though 6, the system bus interface 120 and L2 cache control unit 118 of Figure 1 will be further described.

Correspondence between the high level block diagram of Figure 1, and the more detailed illustration of the preferred embodiment in Figures 3 to 6, is as follows. Bus 101 of Figure 1 corresponds in Figures 3-6 to system controls lines 559 at driver/receiver 556, system address lines 569 at driver/receiver 564, system data hi bus 513 at driver/receiver 512, and system data low bus 517 at driver receiver 516. Bus 103 to L2 cache 104 of Figure 1 corresponds to L2 cache address lines 691 out of driver 690, L2 tag address line 693 out of driver 692, L2 tag data lines 697 at driver/receiver 694, and L2 cache data bus 645 at driver/receiver 644. ICU bus 109 of Figure 1 corresponds (from ICU) to IC request lines 351, and (to ICU) DOIC register 606 output lines 607,

and bypass to IC multiplexer 616 on lines 617. DCU bus 111 of Figure 1 corresponds (from DCU) to DC request lines 471 and data cache write data bus 425, and (to DCU) to bypass to DC multiplexer 620 on lines 621 and data cache data out (DODC) register 608 output line 609. Address translation unit (ATU) input/output bus 107 of Figure 1 corresponds to ATU request lines 371, ATU write data bus 373, and multiplexer 612 output lines 353.

Referring to Figures 4 and 6, requests to L2 cache control 118 are latched in address/command register 650 from ATU request lines 371, IC request lines 351, DC request lines 471, and on lines 567 from address in register 566, which latches system bus addresses on lines 565 from receiver 564. These address/command signals are latched as required in registers 650, 652 and 654 connected via lines 651 and 653. The outputs of the third register 654 is fed to controls block 660 on line 655. The output of first stage register 650 is fed on lines 651 to register 652, driver 690 to provide L2 cache address signal 691, driver 692 to provide L2 tag address signal 695, ECC checking circuit 684, address comparator 664, controls block 660, cache controller (CC) snoop address register 670, processor address registers CBPADR 674 and CBMADR 676, and address multiplexer 680. ECC 684 output is fed on lines 685 to driver 694 to provide L2 tag data on lines 697. CBPADR address register 674 contains the address to the system bus in the event of a cache miss, the output of which is fed to multiplexer 680 on line 675. CBMADR address register 676 contains the snoop address portion, and its output is fed to multiplexer 680 on line 677. Receiver 694 output from L2 tag data lines 697 is fed on lines 695 to L2 tag in register (L2TAGIN) 688 and thence on lines 689 to error correction code (ECC) block 686. The output of ECC block 686 is fed on lines 687 to comparator 664, address registers 670, 674 and 676. The output of comparator 664 is fed on line 665 to controls block 660. CCS address register 670 output line 671 represents the data cache snoop address. The output of address out multiplexer 680 is fed on lines 681 to address out register 560, and thence on line 561 to the system address bus 569 through driver 564. The output of controls block 660 is fed on lines 663 to arbitration and control block 552, and on lines 661 to address/command register 658. Arbitration and control block 552 receives control data from receiver 556 via lines 557, and provides output on lines 555 to controls block 660, and in the event of an L2

cache miss request control signals are sent on line 553 through driver 556 to system controls bus 559. Another output of controls block appears on lines 661 to address/command register 658, the output of which appears on line 659 to multiplexer 672. Multiplexer 672 also receives input from lines 653 and 655, and provides its output on lines 673 back to register 650.

Referring to Figure 5, ECC block 632, DOIC register 606, DODC register 608, L2PDO register 636, multiplexer 616 and multiplexer 620 each receive inputs from data input register 624 on bus 625. The output of ECC block 632 is fed on line 633 to L2 data out register 638, and thence to driver 644 on line 639. The output of L2PDO register 636 is fed on line 637 to inpage buffer 646, the output of which is fed on line 647 to L2PDI register 642 and ECC circuit 632. The output of L2PDI register 642 is fed on line 643 to DOIC register 606, DODC register 608, CCDI register 624, and to bypass multiplexers 620 and 616. The output of multiplexers 620 and 616 represent bypass data, and are fed on lines 621 and 617 to the DC and IC, respectively. Data cache write data line 425 is fed to CMCD register 628 and CCDI register 624. The output of CMCD register 628 is fed on lines 629 to L2PDO register 636, and castout buffers 602.

Referring to Figures 3 and 5, L2 cache data in from bus 645 is received at receivers 644, fed on line 649 to L2 data in register 640 and thence on lines 641 to ECC circuitry 634 and bypass multiplexers 616 and 620. From ECC circuitry 634, L2 cache data in is fed on lines 635 to cache controller data in register (CCDI) 624, DOIC register 606 and DODC register 608. DODC register 608 output 609 is fed to data cache unit 116 (Figure 1), DC bypass multiplexer 620, ATU multiplexer 612, and castout buffers 602. The output of DOIC register 606 is fed on lines 607 to instruction cache unit 114 (Figure 1), ATU multiplexer 612, and castout buffers 602. Castout buffers 602 output on lines 603 is fed to data high output register 502 and multiplexer 520, the output of which is fed on lines 521 to data output registers 502 and 504.

In operation, registers 624 and 636 form a pipeline buffer to inpage buffer 646 and register 642. Inpage buffer 646 caches a line from the system bus. Line 641 from L2 data in register 640

to bypass multiplexers 616, 620 allows the saving of a cycle on cache misses when error correction is not required. DOIC register 606 provides corrected data to instruction cache unit 114, and DODC register 608 provides corrected data to data cache unit 116. Either register may supply data to the ATU 124.

The normal path for routing L2 cache data is through register 640, ECC 634, and DOIC register 606 and DODC register 608.

Processor Interface Unit 120

Referring now to Figure 3, a more detailed description of processor interface unit 120 of Figure 1, and associated circuitry, will be provided. Figure 3 represents the data flow portion of PIU 120 and system bus 102.

System bus 102 data high bus 513 and data low bus 517 communicate through driver/receivers 512 and 516, respectively with data high output register 502 on lines 503, data high in register 506 on lines 515, data low out register 504 on lines 505, and data low input register 508 on lines 519. Each of busses 513, 517 is capable of handling eight bytes of data, providing a 16 byte data bus. If the system is operating on only eight bytes, only one set of the input/output registers (such as 504, 508) is used.

System data input registers 506, 508 outputs on lines 507 and 509, respectively, are fed to multiplexer 524 and thence, along with register 506 on lines 507, on lines 525 to cache control data in (CCDI) register 624 (Figure 5), which is the main data input register of the cache controller. Data input register 624 output is fed on bus 625 to multiplexer 520.

Load/Store Unit (LSU) 110

Load/store unit (LSU) 110 functions to decode fixed point and floating point loads and store and cache management operations, and to send effective addresses and storage commands to the data cache unit (DCU) 116. LSU 110 also handles most move-to and move-from special purpose register (SPR) 334 instructions. In addition to functioning as a load/store unit, LSU 110 also controls instruction execution sequencing after instructions have been dispatched, through detection of most instruction execution interlocks, and the generation of resulting pipeline hold signals.

LSU 110 provides a six port register file 306, made up of four 32x18 register array macros, arranged as a 32x72 array with two write ports and four read ports. This array implements the 64-bit general purpose registers (GPRs) 306. GPR array 306 also provides operands for fixed point unit (FXU) 106 decode stage (not shown) as well as for LSU 110. FXU 106 decodes its own instructions and generates requests to LSU 110 for the necessary operands, as well as providing on line 327 a result operand and address when appropriate. LSU 110 accesses GPRs 306 for registers needed to generate effective addresses (EA), and for data for store instructions. Data operands received from data cache 116 on line 461, and updated effective addresses are written back to the GPRs by LSU 110. Lines 327 contain the FPU results, and are fed to register 330.

In handling floating point loads and stores, LSU 110 generates effective addresses using operands from GPR 306, and accesses the necessary floating point register (FPR) operands from the floating point unit (FPU) 108.

Instructions dispatched to LSU 110 are latched in its DECODE cycle instruction register 302 at the end of the I-fetch cycle. The basic LSU 110 pipe is three stages: DECODE 302/304, EXECUTE 316/320, and PUTAWAY 330. During the DECODE cycle corresponding to 302/304, the instructions are decoded, and operands are fetched from the GPR 306 array. Addressing

operands are gated to a 64-bit address generation (AGEN) adder, and a 64-bit effective address is calculated. The effective address (EA) is sent on lines 313 to the address translation unit (ATU) 124 and to data cache unit (DCU) 116 and latched at the end of the DECODE cycle in pipeline buffer 316 which holds the effective address during the EXECUTE cycle.

During the EXECUTE cycle, the operand for store operations is passed to the DCU on line 323, where it is aligned in block 460 and saved in register 456 for PUTAWAY in D-cache 400. At the end of the EXECUTE cycle, if a load type instruction is being executed, the data operand returns on line 461 to LSU 110 from the DCU, and is saved in pipeline buffer 330 for PUTAWAY.

During PUTAWAY cycle 330, as is represented by lines 333, up to two 8-byte or one 16-byte operand may be written to GPR 306. Floating point loads are limited to one 8-byte operand per cycle. GPRs 306 are not written until late in the PUTAWAY cycle 330. This requires that operands being written to these arrays be bypassed around the arrays to reduce pipeline interlocks. Delaying the write to GPR 306 also allows sign extension for algebraic load operations to be performed, helping to balance path delays between EXECUTE 316/320 and PUTAWAY 330 cycles for these instructions.

Fixed Point Unit (FXU) 106

Fixed point unit (FXU) 106 executes the fixed point instructions, not including storage access instructions. FXU 106 includes a 64-bit adder, a 64-bit logical unit, a 64-bit rotate-merge unit, and a 64-bit carry save adder which supports two-bit-per-cycle product formation during multiply instructions.

During division, quotient formation occurs one bit per cycle, through repeated subtraction of the divisor from the shifted dividend.

Floating Point Unit (FPU) 108

Floating point unit (FPU) 108 executes the floating point instructions, but not the storage access instructions. In one exemplary embodiment, FPU 108 includes a 5-port 32x72-bit register array, a 32-bit status-control register, a 3-bit overlap scan booth encoder unit, 2-bit quotient generation unit, a 106-bit carry save adder, a 106-bit increment-full adder, an operand alignment shifter unit, a normalizer unit, and a rounder unit.

Address Translation Unit (ATU) 124

Referring, primarily, to Figure 2B, address translation unit (ATU) 124 translates the data effective address (EA) from load/store unit (LSU) 110 and the instruction effective address from instruction unit 112 into real addresses used by the data and instruction caches to access their L1 caches and used by the L2 cache control unit 118 to access the L2 cache 104.

Microprocessor 100 implements segment lookaside buffers (SLB) 354, 376 and translation lookaside buffers (TLB) 358, 378, which function as caches for segment and page table entries. When a required entry is not found in a look aside buffer, ATU 124 initiates a fetch to L2 cache control 118 to access segment and page table entries from memory 126 or L2 cache 104.

ATU 124 reports any translation data storage interrupts (DSI) to the load/store unit 110 and any translation instruction interrupts to the instruction unit 112. Reference, change and tag change bits are all updated by store requests to cache control 118 from ATU 124.

Microprocessor 100 provides a 4-entry SLB 354 for instruction address translation and an 8-entry SLB 376 for data address translation. SLBs 354, 376 contain the most-recently translated segments in a fully associative arrangement. The ESID (Effective Segment ID) portion of the

effective data or instruction address is compared 356, 374 simultaneously to all entries in the respective SLB 354, 376 ESIDs when segment translation is enabled.

ATU 124 includes separate instruction and data TLBs 358, 378, respectively, to hold the results of virtual to real address translations. With virtual to real translation active, the VSID from the matching SLB 354, 376 is compared in comparators 362, 382 against the VSID stored in the TLB 358, 378. If a compare is found, the real page number (RPN) stored in the matching TLB 358, 378 entry is used to form the real address. Replacement is managed independently in each TLB 358, 378 by an LRU bit for each of the 256 pairs of entries.

L1 Data Cache Unit (DCU) 116

In a preferred embodiment, L1 data cache unit (DCU) 400 has the following attributes: 64 KB size, 64 byte line size, 4-way set-associative, 2 subline-modified bits per line, MRU slot selection, 40-bit real address, 16-byte dataflow to/from processor, store-in design, and multi-processor support. The term "cache line" refers to a 64-byte block of data in the cache which corresponds to a single cache directory entry. Slot MRU 402 provides selection of one of four sets of cache data during an execute cycle. Real address MRU 430 supplies bits 50:51 to cache 400 and cache directory 440. Error correction ECC (not shown) is provided on cache 400 and cache directory 440. Write-thru mode is implemented.

The data cache 116 array 400, representing a collection of sub arrays, is based on a 1024x78 1R1W "virtual" 2-port array macro. It provides for a read operation followed by a write operation within a processor cycle. Read data remains valid on the array outputs until the next read operation begins even if there is an intervening write operation. Eight of these arrays are used to form a 64KB cache 400. Two arrays are used per slot to form a 16-byte dataflow in and out of the array, represented by lines 401. Data parity is stored in the array. The last bit stored in the array is odd address parity across bits 50:59 of the address used to access the data cache.

Two arrays are needed to implement data cache directory 440. The directory implements a 28-bit real page number (RPN) along with five ECC check bits. A valid and two subline modified status bits are maintained, and three check bits are stored with them. The RPN and status fields are replicated four times to represent the four sets accessed at a particular directory array address. A 3-bit LRU is shared between two directory arrays to indicate the least recently used slot.

Slot MRU 402 logically appears as a 1024x4 array where each entry is associated with a cache line in data cache 400. Bits 48:51 of the 48:57 used to access the logical array 400 are effective address bits. MRU 402 bits are updated whenever an incorrect slot guess or a cache miss occurs.

Real address (RA) MRU 430 is used to generate a prediction of real address bits 50 and 51 for addressing both cache 400 and cache directories 440. As is represented by line 309, array 430 is read as part of the AGEN stage 304 of the pipeline. If a load/store unit pipeline EXECUTE stage or latched PUTAWAY stage hold is present, then the array 430 output is not used. Real mode is also used to determine if the array 430 (Figure 2C) output is used. Real mode determination occurs in AGEN 304 which sets control line 307 to either the real or virtual mode addressing. If real mode is determined, then load/store effective address (LSEA) 317 bits 50:51 are used by register 408 to access cache 400 and register 414 to access cache directory 440 instead of RA MRU array 430 output.

Real address (RA) MRU array 430 is updated from DC address register 408 via lines 281 whenever a wrong prediction occurs with respect to translated address bits 50:51. Also, data cache address register 408 and data cache directory address register 414 are updated with proper values of address bits 50:51 via line 381 for reaccessing the cache 400 and cache directory 440 arrays. Multiplexor 412 is then switched under control of data cache control block 470 so that address register 408 is used to access cache array 400. A similar function occurs with multiplexer 436 so that register 414 is used to access the directory array 440. The LSU 110 pipeline is stalled for one

cycle to allow cache 400 and directory 440 to be reaccessed in parallel in the same cycle. Data is then returned to LSU 110 via line 461 in the following cycle.

Instruction Cache Unit (ICU) 114

Instruction cache unit (ICU) 114 contains the physical arrays, address compares, and error checking circuitry to provide a 64KB 4-way associative instruction cache with single-bit error detection and recovery. The single-cycle cache access provides up to four instructions from a selected 128-byte cache line. Instruction cache unit 114 provides instructions to other functional units, including branch prediction.

L2 Cache Control Unit 118

The functions of the L2 cache control unit 118 are to provide processor 100 with access to a private L2 cache 104, plus access to memory 126 through system bus 102 which also supports memory coherence control for multiprocessor operations. L2 cache 104 is implemented as external static RAMs, with one set of SRAMs for the directory and another set for the data.

CCU 118 accepts commands from four sources: data cache unit 116, instruction cache unit 114, address translation unit 124, and system bus 102 via processor interface unit (PIU) 120. To handle these commands, CCU 118 uses the buffer structure shown in Figure 6. External and internal commands are prioritized by CCU controls 660 and placed into ADR/CMD buffer 650. ADR/CMD buffer 650 output 651 is then used to access an L2 directory (not shown) via interface lines 693 driven by driver circuits 692 to determine the hit/miss status. Additionally, appropriate address bits from bus 651 are concurrently used to access an L1 status array (not shown) in controls 660 to determine if a data cache snoop needs to be done. Finally, ADR/CMD buffer 650 is used to control updating status and tag information in the L2 directory as required, a process well understood in the art.

The four L2 hit/miss states are:

1) Modified - This line is different from memory and no other coherent cache has a copy of this line.

2) Exclusive - This line is the same as memory and no other coherent cache has a copy of this line.

3) Shared - This line is is the same as memory and other caches may have a copy of this line.

4) Invalid - This cache and this processor's data cache do not have a copy of this line.

Data can be in the data cache only if it is also in the L2 cache.

Commands only stay in ADR/CMD buffer 650 for three cycles, at which time the command moves to ADR/CMD buffer 652 or ADR/CMD buffer 658. A processor command is moved into the ADR/CMD buffer 652 when said command is in ADR/CMD buffer 650 and the resources it needs, such as the data flow, are not available. The command will stay in ADR/CMD buffer 652 until the resource becomes available.

Commands are moved to the ADR/CMD buffer 658 from ADR/CMD buffer 650 by way of controls block 660 when a system bus snoop command needs to use the data path. The command will stay in ADR/CMD buffer 658 until the data path is available. Commands that need to issue address commands on the system bus are placed in ADR/CMD buffer 654. The command will stay in ADR/CMD buffer 654, being retried if necessary, until a successful address status and response is received from system bus 102. If data movement is required the command is then turned over to the CCU data flow logic.

Feedback from ADR/CMD buffer 658 to ADR/CMD buffer 650 is necessary for two separate functional operations. The first feedback case is for processor read commands that encountered a shared address response from system bus 102. When the processor read command is first in the ADR/CMD buffer 650 the L2 directory is marked exclusive, assuming that this L2 will have the only copy of the data. If another device indicates that it also has a copy of this data, by a shared address response on system bus 102, then the L2 directory must be changed from exclusive to shared.

The second feedback operation is used for processor write operations that must wait for a successful system bus 102 address status and response before the data can be written. For processor stores or data-cache-block-zero (dcbz) instructions that hit shared in the L2 directory, the processor must make sure that it holds the line in the exclusive state before it updates the data. Before the processor can get ownership of the shared line it may lose the line to another device, so the feedback path is provided to reinitiate the directory access.

L2 Cache Control Unit Snoop Operations

Snoop commands from system bus 102 come in through processor interface unit 120 and are presented to ADR/CMD buffer 650 via bus 567. At the same time a shift register (not shown) is started. The shift register is used to 'time out' the snoop command. Snoop commands require a response within a fixed time, but the command may be delayed before being brought into ADR/CMD buffer 650 because of other higher priority commands. If the shift register 'times out', an address retry response will be issued to the system bus 102.

When a snoop command is accepted into ADR/CMD buffer 650 the L2 directory and L1 status array are checked. If the command hits in the L2 directory and the L1 status array, then a snoop command is issued to the data cache. If data must be moved to complete the snoop command, it will be first moved out of the L2 cache into the castout buffer 602. Then if the data cache has a

modified copy of the data, its copy of the data will be moved to the castout buffer 602 and subsequently via bus 603 to system bus 102.

The memory management policy is such that segment and page translation table entries may not be accessed directly from the L1 data cache by the ATU 124. Consequently, another type of snoop operation is done for ATU commands. When an ATU command comes in, the data cache is snooped using the L1 status array. If the data cache has modified data, the ATU command is stopped until the data is moved from the data cache to the L2 data RAMs.

Processor Interface Unit (PIU) / Bus Interface Unit (BIU) 120

Referring to Figures 1 and 3, processor interface unit (PIU) 120 controls and monitors all communications with the main system bus 102. The main functions of PIU 120 are:

1) Transport commands, address, and data between CCU 118 and system bus 102.

2) Prune out incoming command-address transfers that do not require the attention of CCU 118.

3) Compensate for clock domain differences between the processor 100 units and system bus 102.

4) Maintain and monitor system checkstop logic for processor run-time diagnostics (PRD).

System bus interface, or processor interface unit (PIU) 120, in general, receives commands from L2 cache controller (CCU) 118 on lines 663, transforms them in block 552 to the system bus clock domain and presents them on lines 559 to bus 102. It then monitors status and response information received on lines 559 for the command and informs CCU 118 on lines 555. As

commands arrive from the bus on lines 559, PIU 120 categorizes them into one of three categories: master operations, snoop operations and other operations. Master operations are those originated by CCU 118 on the same chip 100 as PIU 120. These operations need to be monitored for status and response, updating CCU 118 as this information arrives. Snoop operations are those that are originated by other bus units and require the attention of CCU 118. PIU 120 will pass these operations on to CCU 118 indicating a snoop and continue to monitor status and response. Other operations are those originated by other units that do not require the attention of the CCU 118. For these operations, PIU 120 only monitors status and response without informing CCU 118.

Clock Distribution and Control 122

Clock distribution and control 122 contains the logic for gating, shaping, and distributing the internal clocks as well as the off chip cache and directory clocks.

During normal system operation, all clocks are derived from and synchronized to a single oscillator input by a phase locked loop (PLL) circuit which provide a 'zero delay' clock tree relative to the input oscillator and also a frequency multiplier function. Microprocessor 100 uses this function to run the internal processor logic at a faster rate than the system bus 102 interface logic, which runs at the same rate as the oscillator input. A second on-chip 100 PLL is employed to derive the clocks for the off-chip L2 cache 104. This PLL uses the frequency multiplied output of the first PLL as its input. An off-chip feedback path, constructed to match the path to the cache chips, results in a low skew delay relative to the processor clock domain and allows for synchronous communication between processor 100 and cache 104.

Part 2

Data Cache Address Generation

The cache accessing mechanism of the present invention is an improvement upon the mechanism set forth in Figure 7 (corresponding to Figure 6 in the aforementioned Liu U.S. Patent 5,392,410.) In the system of Figure 7, prediction of real address translation is used for shortening the cache access critical path.

In Figure 7, the reference numerals correspond to those of Liu Figure 6, which Liu describes at column 9 line 46 through column 11 line 24. Refer to Liu '410 for the detailed description of the operation.

Translation lookaside buffer (TLB) 210 provides for fast translations of virtual addresses to real addresses. Cache directory (DIR) 220 contains, for each line entry of data cache 230, the real address identification and other needed status tags. Cache array (ARR) 230 is the cache array or cache memory that holds the actual cache data bits, plus other bits necessary for parity checking or for error correction. In this design, a 4-way set-associative cache 230 and a 2-way set-associative TLB 210 are provided. TLBLAT 241 is a 1-dimensional history table, which does not necessarily have the same number of entries as the number of congruence classes in TLB 210.

Processor instruction/execution unit (IE) request address bus 250 accesses storage via 32-bit logical (virtual or real) addresses. Cache 230 is a real address based cache. That is, when a physical line is inserted into the cache, the associated congruence class is determined solely by the real address bits of the line. Comparator 261 detects the TLB hit or miss conditions and verifies TLBLAT prediction.

History table TLBLAT 241 may be used to predict virtual address translation information. TLBLAT 241 is a 1-dimensional history table, which does not necessarily correspond either dimensionally or in the number of congruence classes with TLB 210. In the minimum, each entry in TLBLAT 241 contains the bits in the real page address that are required to select the real address congruence class.

Upon a cache fetch access from the instruction and execution unit (IE) with a virtual address A 250, the following three steps first occur in parallel:

1. Proper bits of A 250 are sent via path 271 to TLB 210 to read out the two entries of TLB 210 which are in the TLB congruence class defined by these bits. The output 281 is sent to compare unit CMP 261.

2. Proper bits of A 250 are sent via path 277 to the control of TLBLAT 241 to select the 3 bits in TLBLAT[A] needed for prediction of real address congruence class for the cache. The 3 bit TLBLAT[A] output is sent to the controls of CMP 261, DIR 220 and ARR 230.

3. Proper bits of address 250 are sent to controls of DIR 220 and ARR 230 on paths 272 and 273. These address bits (within a page boundary), concatenated with the 3 output bits from TLBLAT 241, determine the prediction of congruence class of the cache. When this congruence class information is ready, the following is carried out: a) the four DIR 220 entries at the predicted congruence class are read out via path 282 to CMP 261, and b) four doublewords are read out of ARR 230 and sent via path 283 to late select unit 262.

Referring now to Figures 8A and 8B, primarily in connection with Figures 2B and 2C, in accordance with a preferred embodiment of the invention, consider a microprocessor 100 with a 64-bit internal architecture and big-endian bit numbering.

As is illustrated in Figure 9, according to the big-endian bit numbering protocol, bit 0 is the most significant bit of an operand, and is located on the left. Storage addresses, generated by adding two 64-bit operands together, are called effective addresses (EA) prior to undergoing address translation, which converts them to real addresses (RA). Bits are numbered 0:63 (that is, from 0 to 63, inclusive) from most significant to least significant.

Adder input operands R1 and R2, 700 and 710 are fed on lines 705 and 715, respectively to adder 740, and bits 32:34 are fed on lines 701, 711, and input operands bits 44:51 are fed on lines 703, 713, to adders 720 and 722, respectively. The output of adder 740 is fed on line 251 to registers 250, 414 and 408, and on line 741 to multiplexer 736. Adder 720 assumes a carry in value of C=1, and adder 722 assumes a carry in value of C=0. Adder 720 output is fed on line 721 to TLBLAT (also called real address MRU) 730, and adder 722 output is fed on line 723 to TLBLAT 732, the outputs of which are fed on lines 731 and 733, respectively, to multiplexer 736. The output of multiplexer 736 appears on two-bit wide bus 737 to registers 410 and 416. The output of register 271 is fed to two-way associative TLB 210, the output of which appears on line 281, which is fed to comparators 764 and 766, and back to TLBLATs 730 and 732. The output of registers 414 and 416 are fed on lines 415 and 417 to directory array 220, the output of which appears at comparator 766 on line 282. The outputs of registers 410 and 408 are fed on lines 411 and 409, respectively, to data array 230, the output of which is latched from line 283 at register 262. The output of comparator 764 is TLBLAT update command line 765 and TLBLAT update bits 294. The output of comparator 766 is cache miss control line 294. Register 262 latches data on line 292. TLBLAT update line 765 active designates a TLBLAT miss. Real address MRU 430 comprises TLBLATs 730, 732, and MRU adder output line 309 corresponds to lines 721, 723 and carry line 741.

In operation, referring further to Figures 8A and 8B in connection with Figures 2B and 2C, in LSU AGEN block 304, where address generation takes place, the addressing operands 700, 710 are fetched from general purpose registers (GPRs) 306 and added together in 64-bit adder 740, the resulting EA 251 being latched in a pipelining buffer 316, DC address register 408, and directory

address register 414 at the end of the DECODE stage 304. (Data cache access is done off of register 408.) Cache array 400 itself is accessed at the beginning of the next cycle, called EXECUTE. All cache array 400 address bits must be valid at the beginning of the cache array access. The cycle time of microprocessor 100 is limited by the critical path to create the effective address in address generation (AGEN) stage 304 and access the cache in EXECUTE stage 320. Addition of a TLBLAT to allow generation of some cache address bits would not be possible using the method of accessing it with the output of the 64-bit adder, unless cycle time was increased.

Real address MRU 430 in the preferred embodiment has a logical size of 256 addresses and is two bits wide. This enables MRU 430 to supply two RA bits for accessing data cache 400 in the next cycle, and a history of up to 256 translations at one time. Effective address bits 52:63 from stage 304 on line 313, provided to data cache 400 by way of register 408 and multiplexer 412 on lines 409, 413 are translation invariant, and are used to directly access data cache 400 as RA 52:63. EA bits 44:51, provided on line 309 from the output of address generation stage 304, are used to access MRU 430, which then predicts (based on history) and provides to data cache 400 by way of register 410 and multiplexer 412, a value for RA 50:51.

Referring to Figure 9, the byte address includes bits 50:51, which are translated (virtual to real) in TLB 210, and line address bits 52:57 which, together with bits 58:63, are translation invariant.

Referring further to Figures 8A and 8B, in accordance with the present invention, given a fixed access time for real address MRU array 430, the only other option (other than limiting cache size, as previously discussed) for avoiding an increase in the critical path delay is to reduce the time it takes to generate EA bits 44:51. As with most carry-lookahead adders, the output of the more significant sum bits are dependent on the carry out of less significant bits. To solve the access time problem, a special adder circuit 720, 722 is provided in address generation stage 304 which has two sum outputs 721, 723 for bits 44:51. The first output 723 assumes the carry-in to bit 51 is '0', and

the second output 721 assumes the carry-in is '1'. MRU array 430 is duplicated, and is shown in Figures 8A and 8B as TLBLAT 1 730 and TLBLAT 0 732. That is, there are two MRU arrays 730 and 732, and each one is accessed with one of the adder sum outputs 721, 723. The MRU array outputs 731, 733, are multiplexed together at multiplexer 736, based on the carry-in to bit 51 on line 741 from LSU address generation stage 304 adder 740. Multiplexer output 737 corresponds to lines 431 in Figure 2C. This design allows the MRU array 430 access to be performed in parallel with the generation of the carry 741, instead of waiting for the carry to be generated before commencing the MRU array 430 access. In one specific embodiment, the carry line 741 is supplied by 64-bit adder 740, eliminating the need for a separate carry-generating circuit. Operands 700, 710 are from general purpose registers for address generation (AGEN). R1 and R2 fields 44:51 are added, while 32:34 are XOR hash bits.

In accordance with another feature of the invention, the special adder 720, 722 provides for hashing of some additional operand 700, 710 bits with the adder's input operands 703, 713 to improve the hit rate in the MRU table 430. By hashing input operand bits 32:34 701 and 711 with bits 44:46, the miss rate of MRU table 430 is halved. In this preferred embodiment, hashing is performed as follows:

R1(32:34) xor R2(32:34) xor HSUM(44:46)

where R1 is operand 1 700, R2 is operand 2 710, and HSUM is the halfsum term in the MRU adder 720, 722. Use of this hashing method avoids increasing the critical path through MRU adder 720, 722, since it does not involve the time-critical carry path inside adders 720 and 722.

The advantages of the method of the preferred embodiments of this invention include improving cache operation by substantially completely overlapping MRU lookup with effective address generation and by performing a hashing operation to randomize entries in the MRU without causing delay in the critical path.

It will be appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention.

In particular, it is within the scope of the invention to provide a memory device, such as a transmission medium, magnetic or optical tape or disc, or the like, for storing signals for controlling the operation of a computer according to the method of the invention and/or to structure its components in accordance with the system of the invention.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. Method for accessing a cache array, comprising the steps of:

generating the outputs of two most-recently-used (MRU) arrays, one by assuming a carryin of zero, and the other by assuming a carryin of one to the least significant bit of a portion of an effective address used to access the MRU arrays; and

addressing and accessing said MRU arrays substantially in parallel with effective address generation;

whereby the addresses for accessing said MRU arrays are generated without having to wait for the carry bit into said portion of said effective address corresponding to said address used to access said MRU arrays, said carry resulting from generation of said effective address.

2. The method of claim 1, further comprising the step of hashing within an adder operable for addressing and accessing said MRU arrays selected portions of operands used for generating said effective address, said hashing operation being performed in the halfsum stage of said adder; thereby improving the probability of finding correct address bits in said MRU array.

3. Method for generating an address to access a cache array, comprising the steps of:

generating an effective address by combining a first operand and a second operand in a first adder;

combining in a second adder, first selected bits from each of said first and second operands with a preselected carryin bit of value 1 to form a first address;

combining in a third adder said first selected bits with a preselected carryin bit of value 0 to form a second address;

accessing a first real address MRU at a location addressed by said first address to generate a third address;

accessing a second real address MRU at a location addressed by said second address to generate a fourth address; and

selecting said third address or said fourth address based upon a calculated carryin bit generated from combining said first operand and said second operand to generate said effective address for accessing said cache array.

4. The method of claim 3 wherein said calculated carryin bit is generated by said first adder.

5. The method of claim 3 wherein each said real address MRU is two bits wide and contains a plurality of addresses.

6. The method of claim 3 comprising the further step of combining translation invariant bits from said effective address with the selected third or fourth address to access said cache array.

7. The method of claim 3 wherein said first real address MRU and said second real address MRU are duplicates of each other.

8. The method of claim 3 wherein said selecting step is performed by feeding said third address and said fourth address to a multiplexer gated by said calculated carryin bit.

9. The method of claim 3 wherein said generating step is performed substantially in parallel with said combining and accessing steps.

10. The method of claim 3, including the further step of hashing second selected bits from said first operand and said second operand with a portion of said first selected bits in forming said first address and said second address.

11. The method of claim 10 wherein said hashing step further comprises the steps of:

exclusive ORing said second selected bits from said first operand with said second selected bits from said second operand; and

exclusive ORing the result with the half sum of said first selected bits.

12. The method of claim 11 wherein said half sum is generated in said second adder in forming said first address and in said third adder in forming said second address.

13. The method of claim 11 wherein each said first operand (R1) and said second operand (R2) include 64 bits (0:63), and wherein said hashing step includes calculating the value

R1(32:34) XOR R2(32:34) XOR HSUM(44:46).

14. Apparatus for generating an address for accessing a history table, comprising:

a first adder for generating an effective address by combining a first operand and a second operand;

a second adder for combining first selected bits from each of said first and second operands with a preselected carryin bit of value 1 to form a first address;

a third adder for combining said first selected bits with a preselected carryin bit of value 0 to form a second address;

a first real address MRU for generating a third address from a location addressed by said first address;

a second real address MRU for generating a fourth address from a location addressed by said second address; and

multiplexer means for selecting said third address or said fourth address gated by a calculated carryin bit generated from combining said first operand and second operand to generate a fifth address for accessing said cache array.

15. The apparatus of claim 14 wherein said calculated carryin bit is generated by said first adder.

16. The apparatus of claim 14 wherein each said real address MRU is two bits wide and contains a plurality of more than two address.

17. The apparatus of claim 14 further comprising means for combining translation invariant bits with the selected third or fourth address to access said cache array.

18. The apparatus of claim 14 wherein said first real address MRU and said second real address MRU are duplicates of each other.

19. The apparatus of claim 14 further comprising multiplexer means gated by said calculated carryin bit for gating said third address or said fourth address to form said fifth address.

20. The apparatus claim 14 wherein said first, second and third adders operate substantially in parallel.

21. The apparatus of claim 14, wherein said second and said third adders are each operable for hashing second selected bits from said first operand and said second operand with a portion of said first selected bits in forming said first address and said second address.

22. The apparatus claim 21 wherein said second and third adders are each operable for exclusive ORing said second selected bits from said first operand with said second selected bits from said second operand; and
exclusive ORing the result with the half sum of said first selected bits.

23. The apparatus of claim 22 wherein said half sum is generated in said second adder in forming said first address and in said third adder in forming said second address.

24. The apparatus of claim 22 wherein each said first operand (R1) and said second operand (R2) includes 64 bits (0:63), and wherein said first and second adders are each operable for calculating the value

R1(32:34) XOR R2(32:34) XOR HSUM(44:46)

in forming said first and second addresses.

25. A program storage device readable by a machine, tangibly embodying a program of instructions executable by a machine to perform method steps for generating an address to access a cache array, said method steps comprising:

generating an effective address by combining a first operand and a second operand in a first adder;

combining in a second adder, first selected bits from each of said first and second operands with a preselected carryin bit of value 1 to form a first address;

combining in a third adder said first selected bits with a preselected carryin bit of value 0 to form a second address;

accessing a first real address MRU at a location addressed by said first address to generate a third address;

accessing a second real address MRU at a location addressed by said second address to generate a fourth address; and

selecting said third address or said fourth address based upon a calculated carryin bit generated from combining said first operand and second operand to generate a fifth address for accessing said cache array.

26. An article of manufacture comprising:

a computer usuable medium having computer readable program code means embodied therein for generating an address to access a cache array, the computer readable program means in said article of manufacture comprising:

computer readable program code means for causing a computer to effect generating an effective address by combining a first operand and a second operand in a first adder;

computer readable program code means for causing a computer to effect combining in a second adder, first selected bits from each of said first and second operands with a preselected carryin bit of value 1 to form a first address;

computer readable program code means for causing a computer to effect combining in a third adder said first selected bits with a preselected carryin bit of value 0 to form a second address;

computer readable program code means for causing a computer to effect accessing a first real address MRU at a location addressed by said first address to generate a third address;

computer readable program code means for causing a computer to effect accessing a second real address MRU at a location addressed by said second address to generate a fourth address; and

computer readable program code means for causing a computer to effect selecting said third address or said fourth address based upon a calculated carryin bit generated from combining said first operand and second operand to generate a fifth address for accessing said cache array.

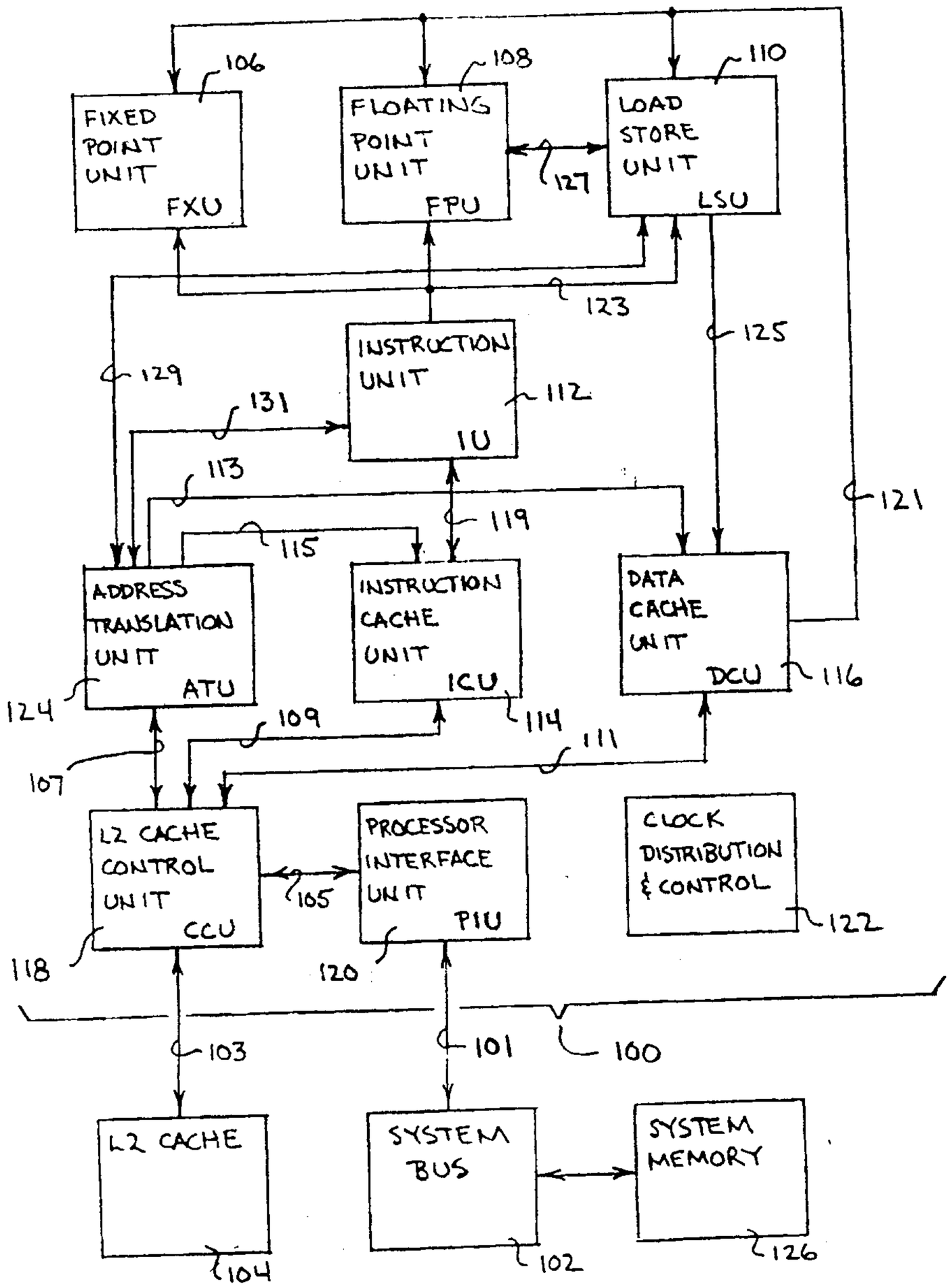
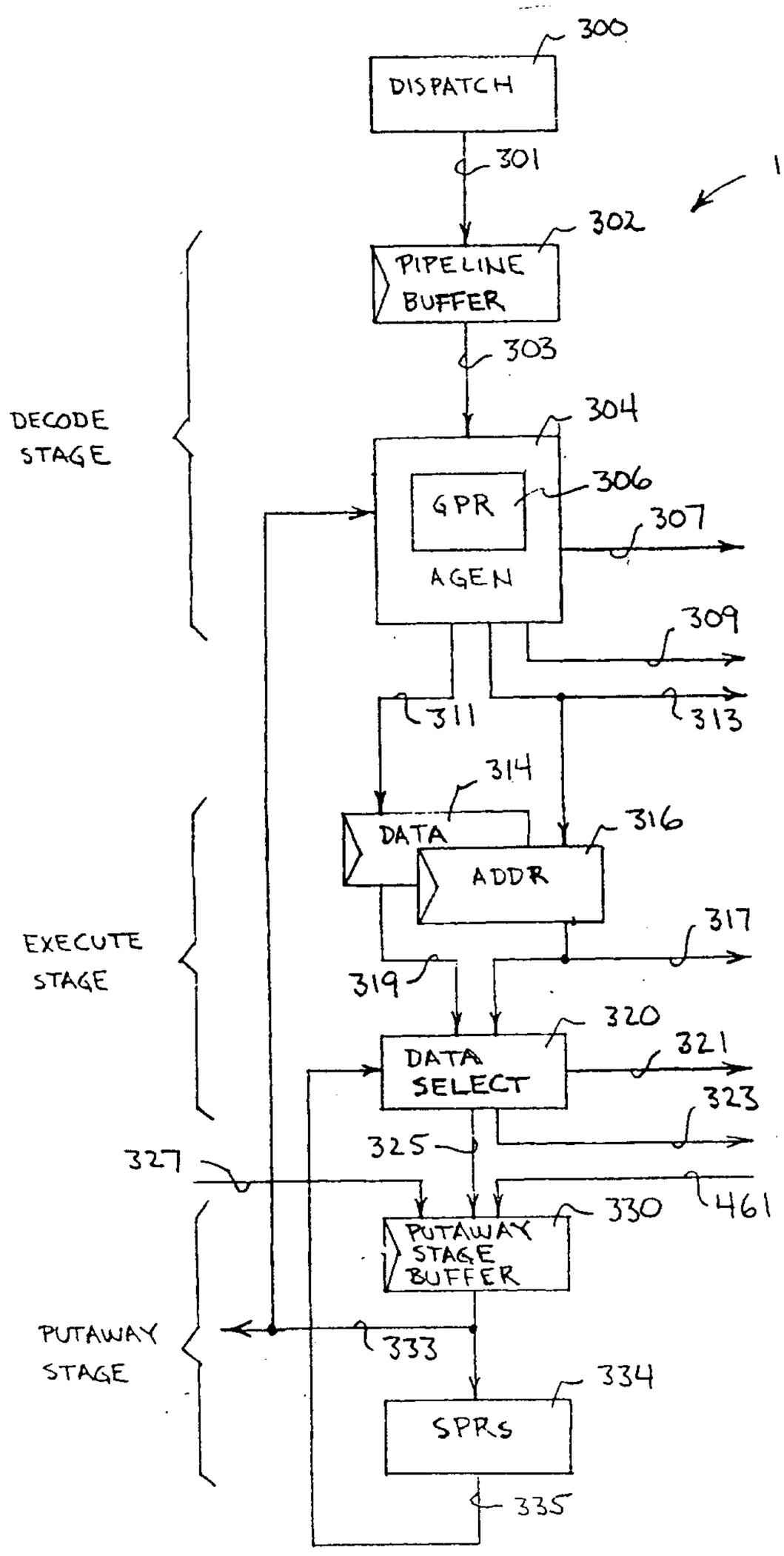

FIXED POINT UNIT
FXU
106
FLOATING POINT UNIT
FPU
108
LOAD STORE UNIT
LSU
110
127
123
125
129
131
INSTRUCTION UNIT
IU
112
113
119
121
115
ADDRESS TRANSLATION UNIT
ATU
124
INSTRUCTION CACHE UNIT
ICU
114
DATA CACHE UNIT
DCU
116
109
111
107
L2 CACHE CONTROL UNIT
CCU
118
105
PROCESSOR INTERFACE UNIT
PIU
120
CLOCK DISTRIBUTION & CONTROL
122
103
101
100
L2 CACHE
104
SYSTEM BUS
102
SYSTEM MEMORY
126

FIGURE 1



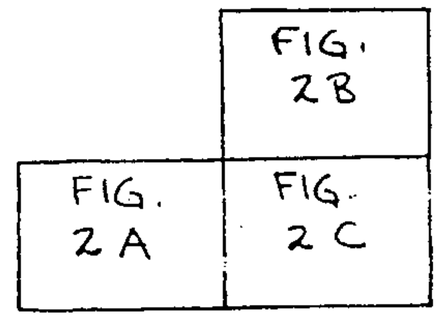


FIG. 2

FIGURE 2A

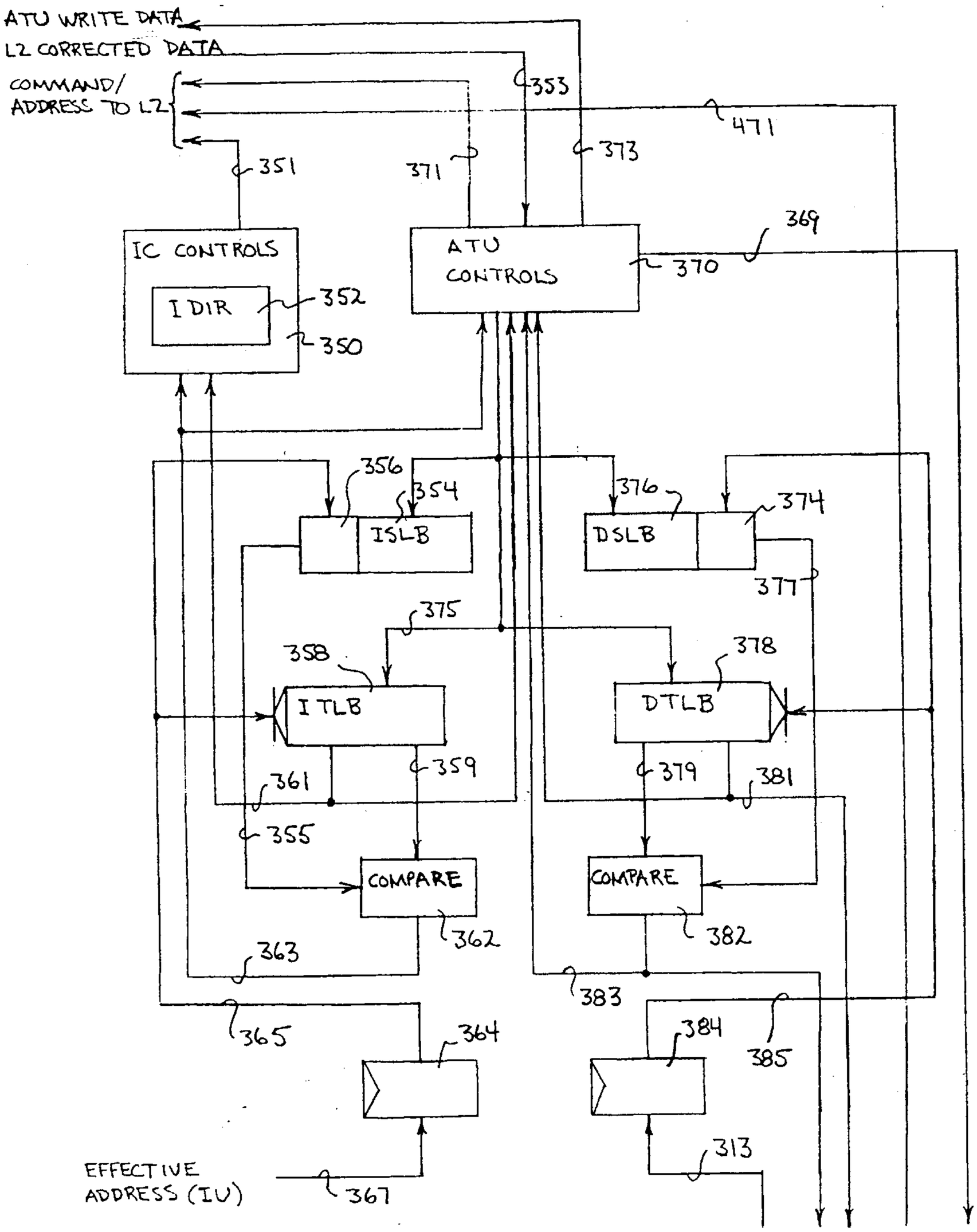

ATU WRITE DATA
L2 CORRECTED DATA
COMMAND/
ADDRESS TO L2
353
471
351
371
373
IC CONTROLS
I DIR
352
350
ATU
CONTROLS
369
370
356
354
376
374
ISLB
DSLB
377
375
358
378
I TLB
DTLB
359
379
381
361
355
COMPARE
COMPARE
362
382
363
383
365
364
384
385
313
EFFECTIVE
ADDRESS (IU)
367

FIGURE 2B

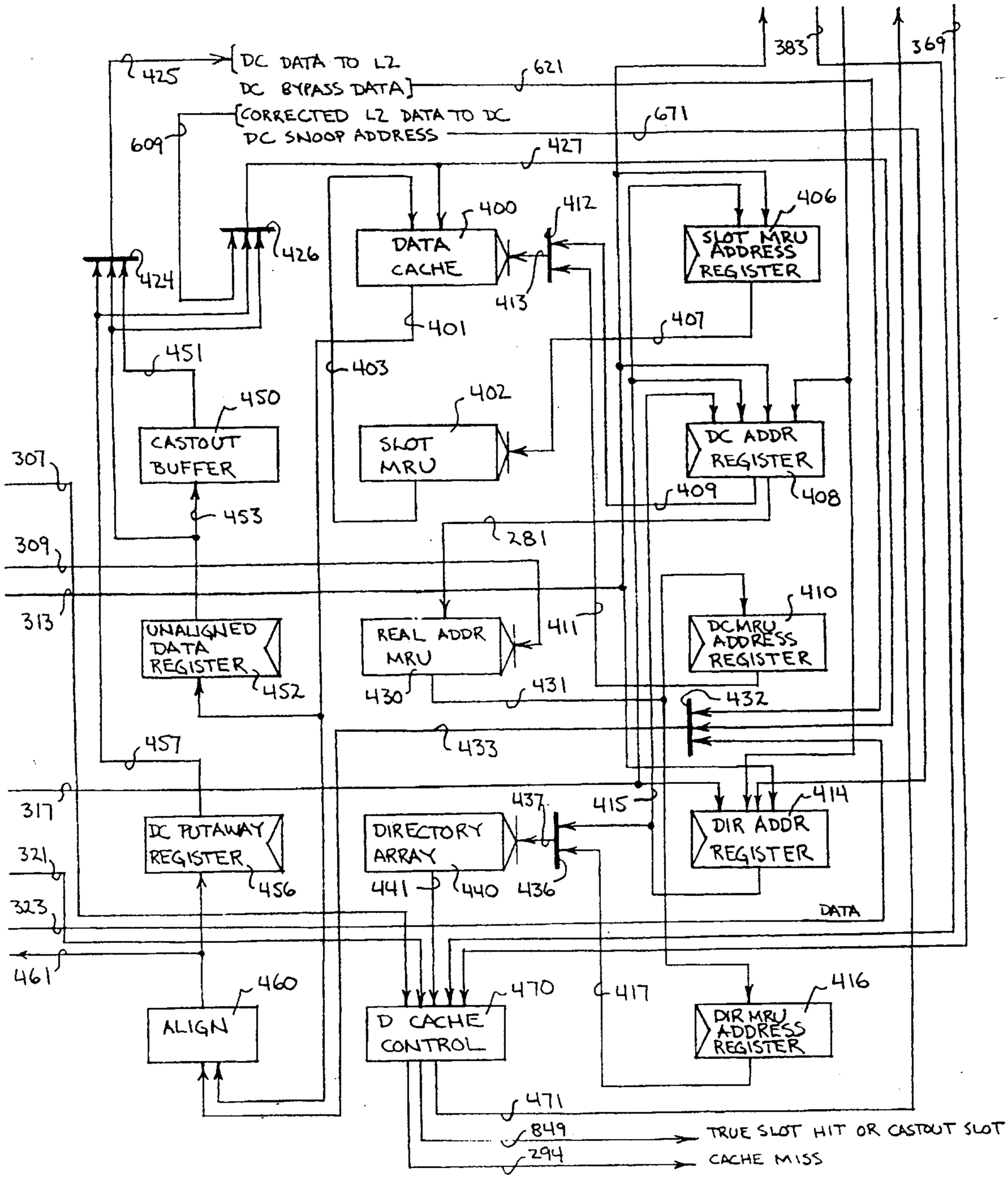

425
DC DATA TO LZ
DC BYPASS DATA
621
383
369
609
CORRECTED LZ DATA TO DC
DC SNOOP ADDRESS
671
427
400
412
406
DATA CACHE
SLOT MRU ADDRESS REGISTER
426
424
413
401
407
451
403
450
402
CASTOUT BUFFER
SLOT MRU
DC ADDR REGISTER
307
453
409
408
309
281
313
410
UNALIGNED DATA REGISTER
REAL ADDR MRU
411
DC MRU ADDRESS REGISTER
452
430
431
432
457
433
317
437
415
414
IC PUTAWAY REGISTER
DIRECTORY ARRAY
DIR ADDR REGISTER
321
456
441
440
436
323
DATA
461
460
470
417
416
ALIGN
D CACHE CONTROL
DIR MRU ADDRESS REGISTER
471
849
TRUE SLOT HIT OR CASTOUT SLOT
294
CACHE MISS

FIGURE 2C

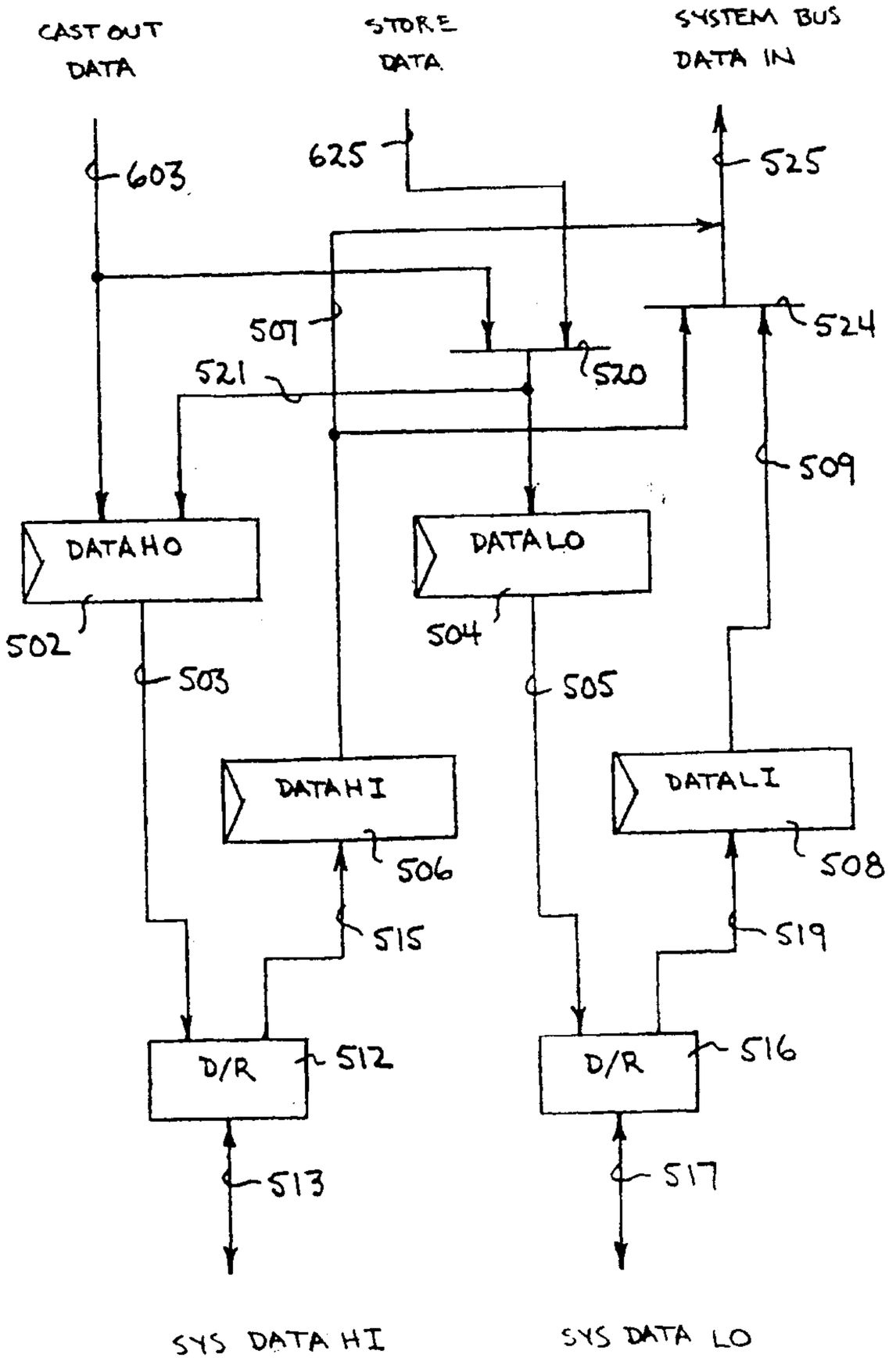

CAST OUT DATA
STORE DATA
SYSTEM BUS DATA IN
603
625
525
507
520
524
521
509
DATAHO
DATALO
502
504
503
505
DATAHI
DATALI
506
508
515
519
D/R
512
D/R
516
513
517
SYS DATA HI
SYS DATA LO

FIGURE 3

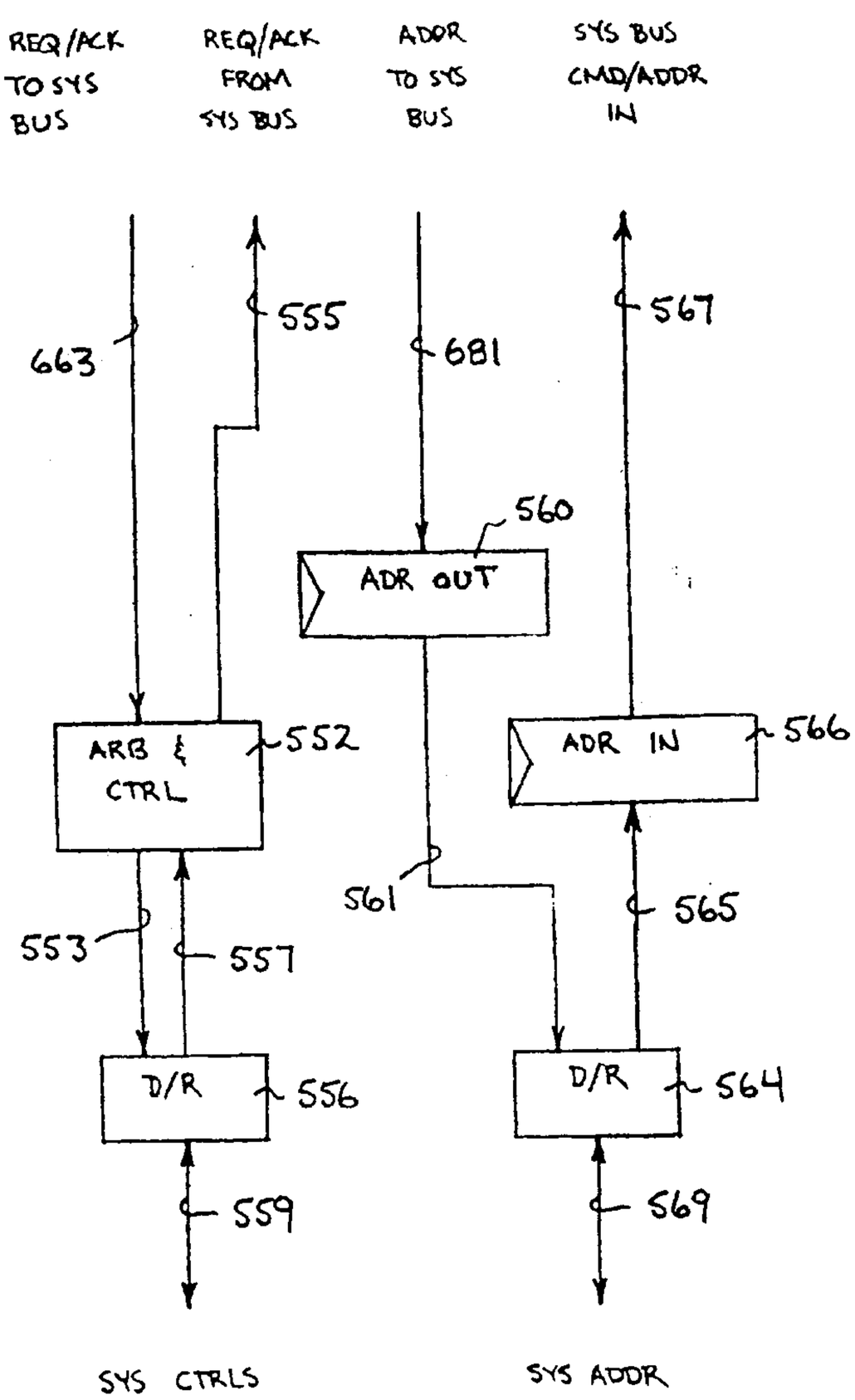

REQ/ACK TO SYS BUS
REQ/ACK FROM SYS BUS
ADDR TO SYS BUS
SYS BUS CMD/ADDR IN
663
555
681
567
560
ADR OUT
ARB & CTRL
552
ADR IN
566
561
565
553
557
D/R
556
D/R
564
559
569
SYS CTRLS
SYS ADDR

FIGURE 4

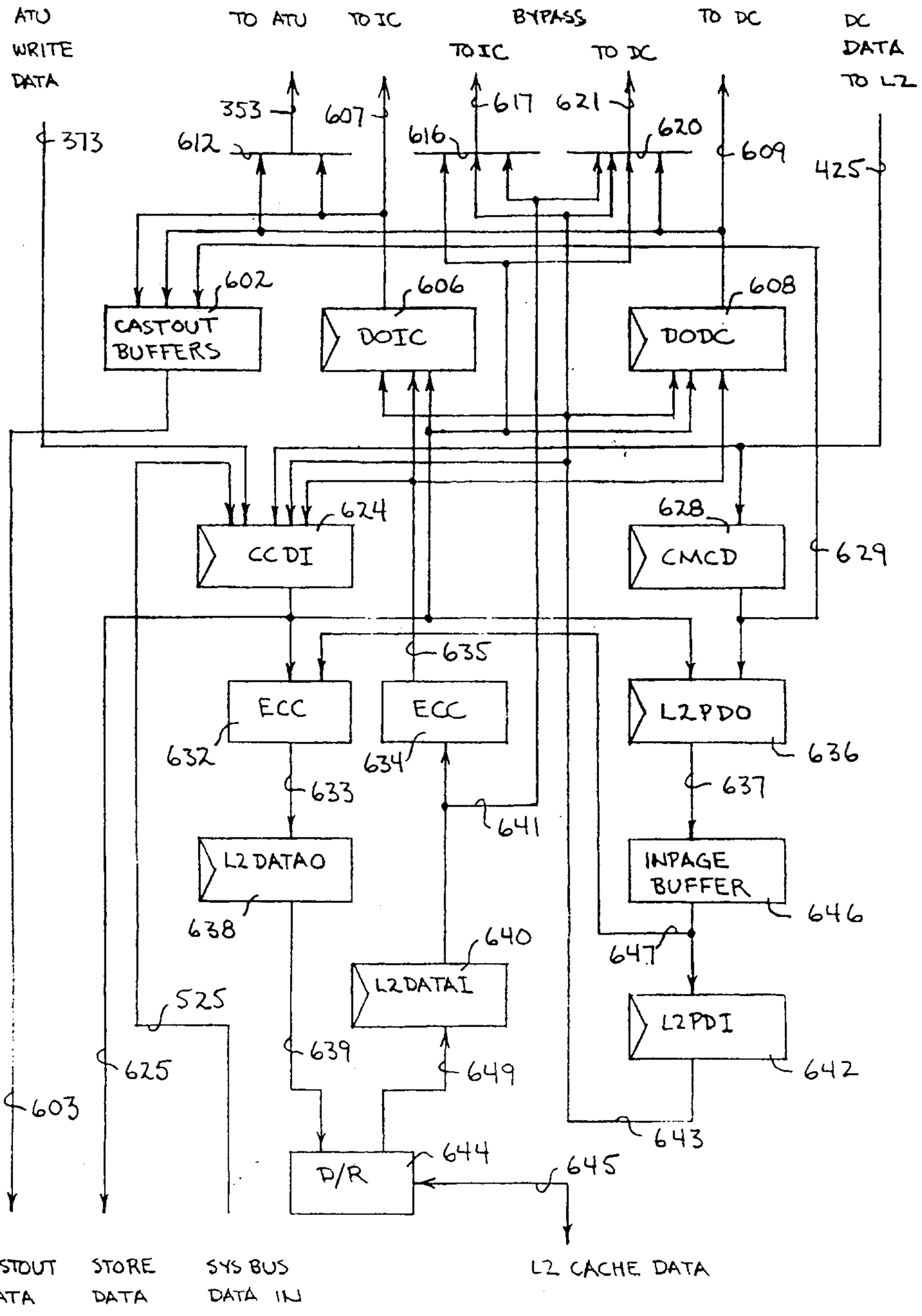

ATU WRITE DATA
TO ATU
TO IC
BYPASS
TO DC
TO IC
TO DC
DC DATA TO L2
373
353
607
612
617
621
616
620
609
425
602
606
608
CASTOUT BUFFERS
DOIC
DODC
624
628
CCDI
CMCD
629
635
ECC
ECC
L2PDO
632
634
636
633
637
641
L2DATAO
INPAGE BUFFER
638
646
640
647
L2DATAI
L2PDI
525
639
649
642
603
625
643
644
645
D/R
CASTOUT DATA
STORE DATA
SYS BUS DATA IN
L2 CACHE DATA

FIGURE 5

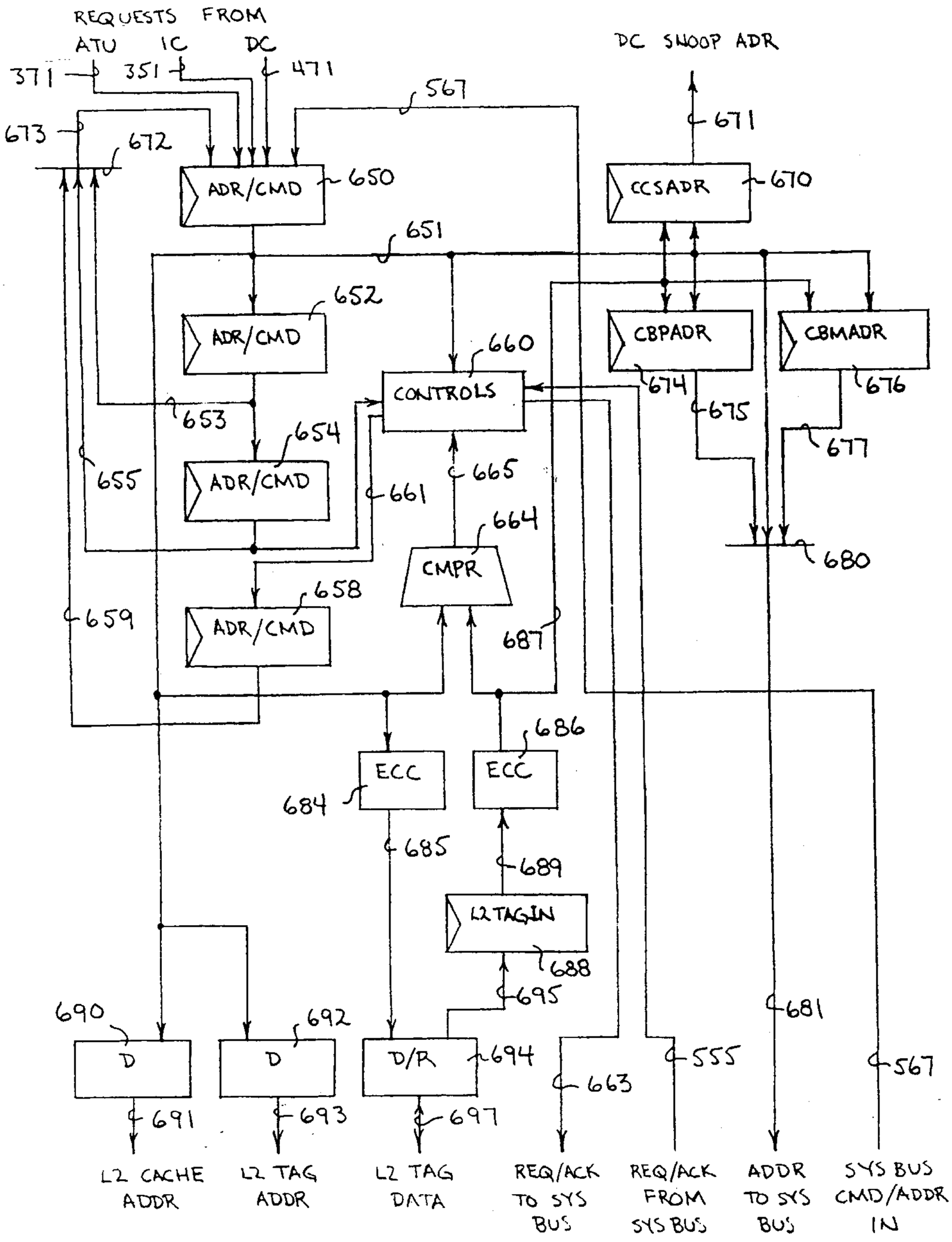

REQUESTS FROM
ATU
IC
DC
371
351
471
567
673
672
ADR/CMD
650
651
652
ADR/CMD
660
CONTROLS
653
654
655
ADR/CMD
661
665
664
CMPR
658
659
ADR/CMD
687
686
ECC
ECC
684
685
689
L2TAGIN
688
695
690
692
D
D
D/R
694
691
693
697
663
555
681
567
DC SNOOP ADR
671
CCSADR
670
CBPADR
CBMADR
674
675
676
677
680
L2 CACHE ADDR BUS
L2 TAG ADDR SYS BUS
L2 TAG DATA BUS
REQ/ACK TO SYS
REQ/ACK FROM IN
ADDR TO SYS
SYS BUS CMD/ADDR

FIGURE 6

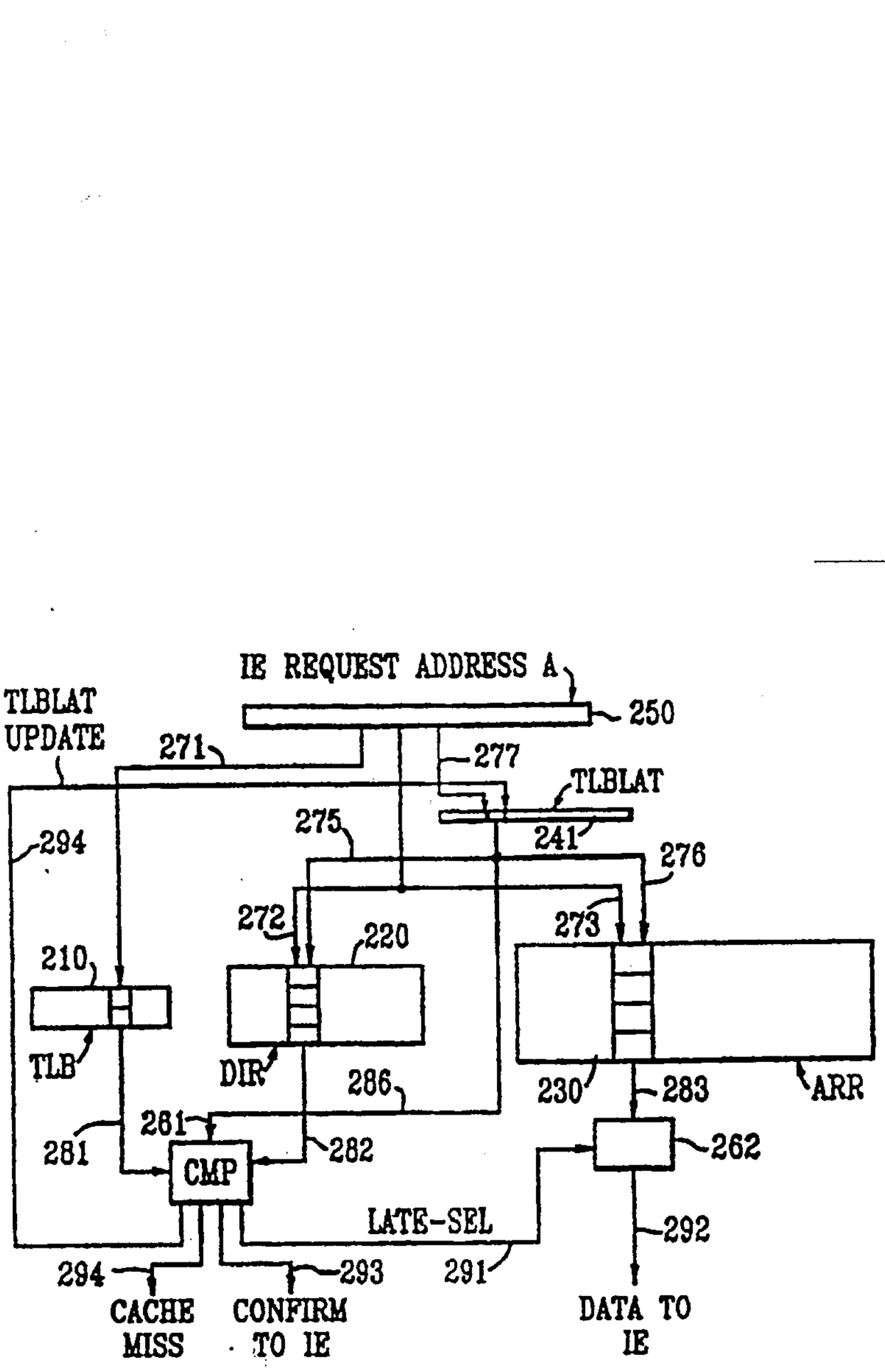

IE REQUEST ADDRESS A
250
TLBLAT UPDATE
271
277
TLBLAT
241
275
276
294
272
220
273
210
TLB
DIR
286
230
283
ARR
281
261
282
CMP
262
LATE-SEL
292
294
293
291
CACHE MISS
CONFIRM TO IE
DATA TO IE

FIGURE 7

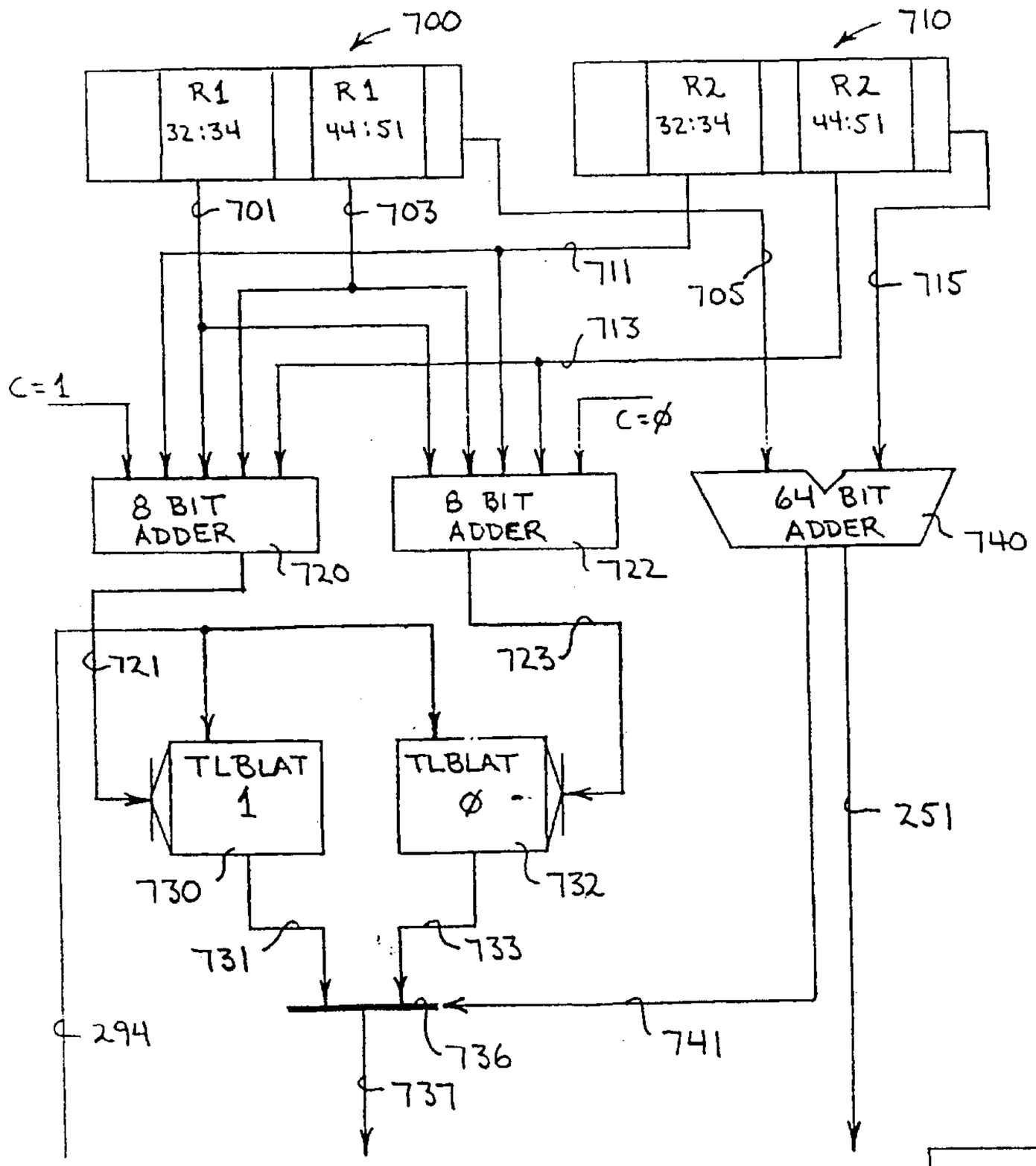


FIGURE 8 A

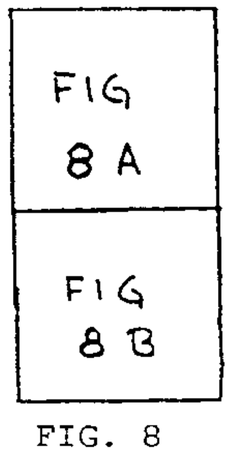


FIG. 8

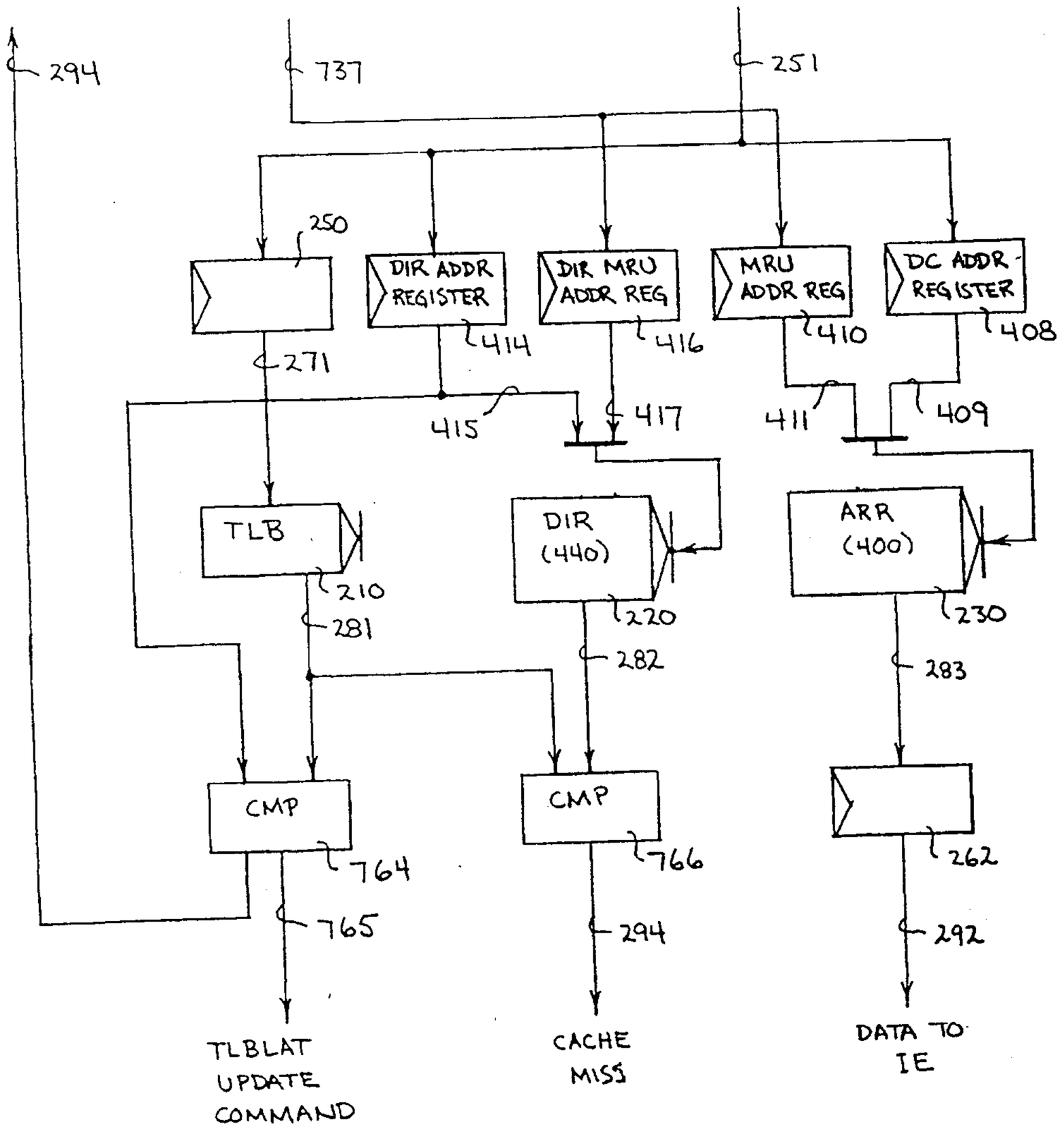

294
737
251
250
DIR ADDR REGISTER
DIR MRU ADDR REG
MRU ADDR REG
DC ADDR REGISTER
414
416
410
408
271
415
417
411
409
TLB
DIR (440)
ARR (400)
210
220
230
281
282
283
CMP
CMP
764
766
262
765
294
292
TLBLAT UPDATE COMMAND
CACHE MISS
DATA TO IE

FIGURE 8B

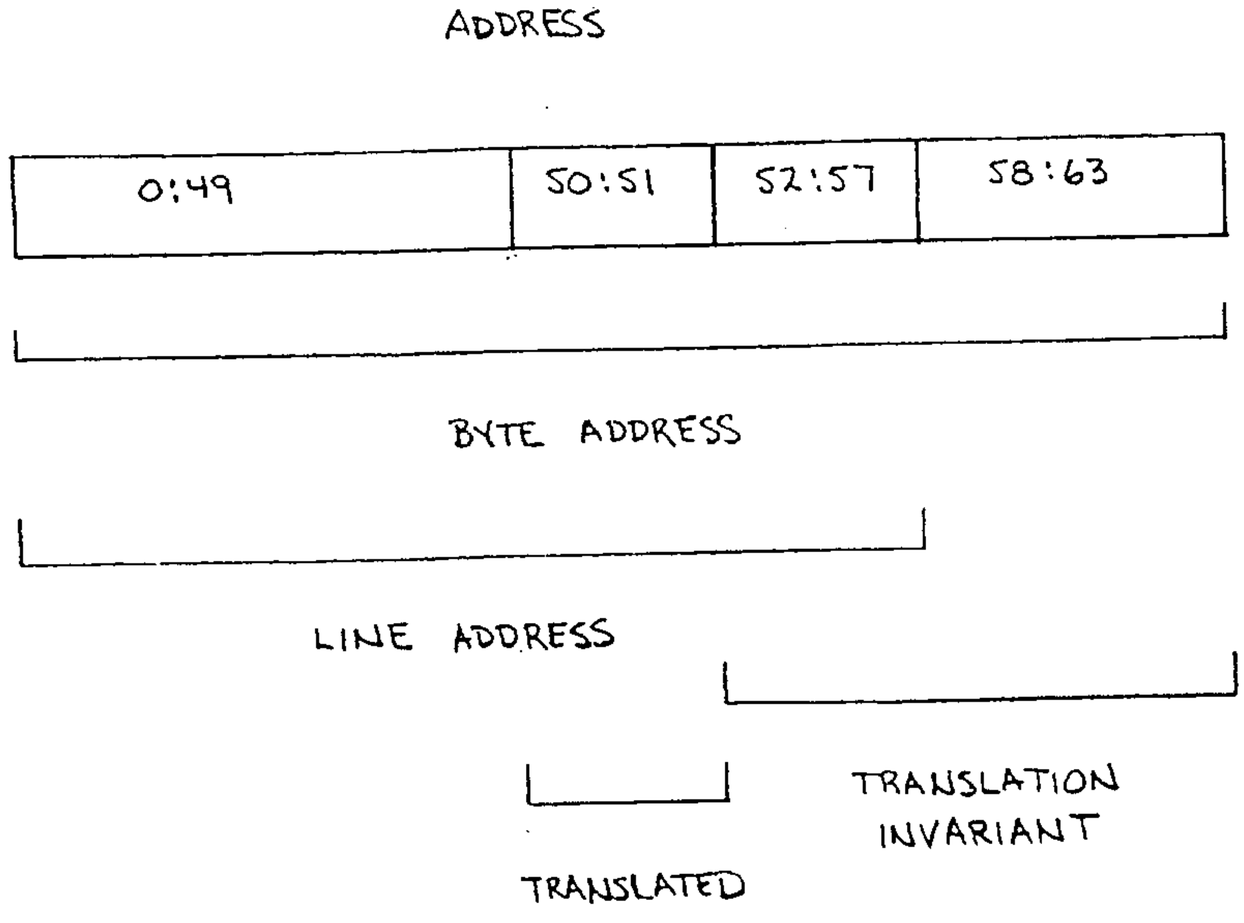
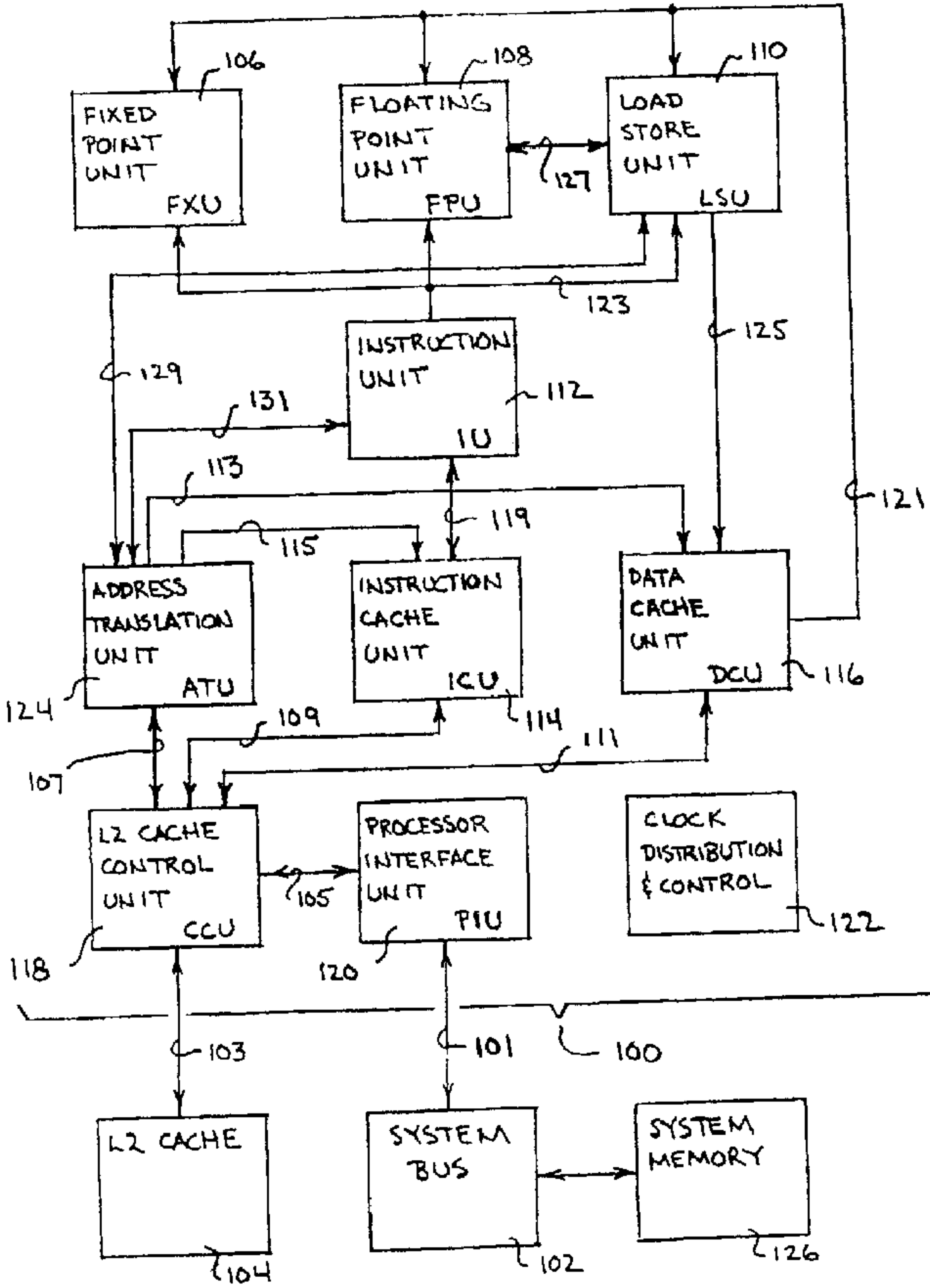

ADDRESS
0:49
50:51
52:57
58:63
BYTE ADDRESS
LINE ADDRESS
TRANSLATION INVARIANT
TRANSLATED

FIGURE 9


106
108
110
FIXED POINT UNIT
FXU
FLOATING POINT UNIT
FPU
LOAD STORE UNIT
LSU
127
123
125
129
131
INSTRUCTION UNIT
IU
112
113
119
115
121
ADDRESS TRANSLATION UNIT
ATU
INSTRUCTION CACHE UNIT
ICU
DATA CACHE UNIT
DCU
116
124
109
114
111
107
L2 CACHE CONTROL UNIT
CCU
PROCESSOR INTERFACE UNIT
PIU
CLOCK DISTRIBUTION & CONTROL
122
105
118
120
103
101
100
L2 CACHE
SYSTEM BUS
SYSTEM MEMORY
104
102
126