PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/16

A1

(11) International Publication Number: WO 91/20043

(43) International Publication Date: 26 December 1991 (26.12.91)

(21) International Application Number: PCT/US91/04058

(22) International Filing Date: 10 June 1991 (10.06.91)

(30) Priority data:
536,198 11 June 1990 (11.06.90) US

(71) Applicant: SUPERCOMPUTER SYSTEMS LIMITED PARTNERSHIP [US/US]; 1414 W. Hamilton Avenue, Eau Claire, WI 54701 (US).

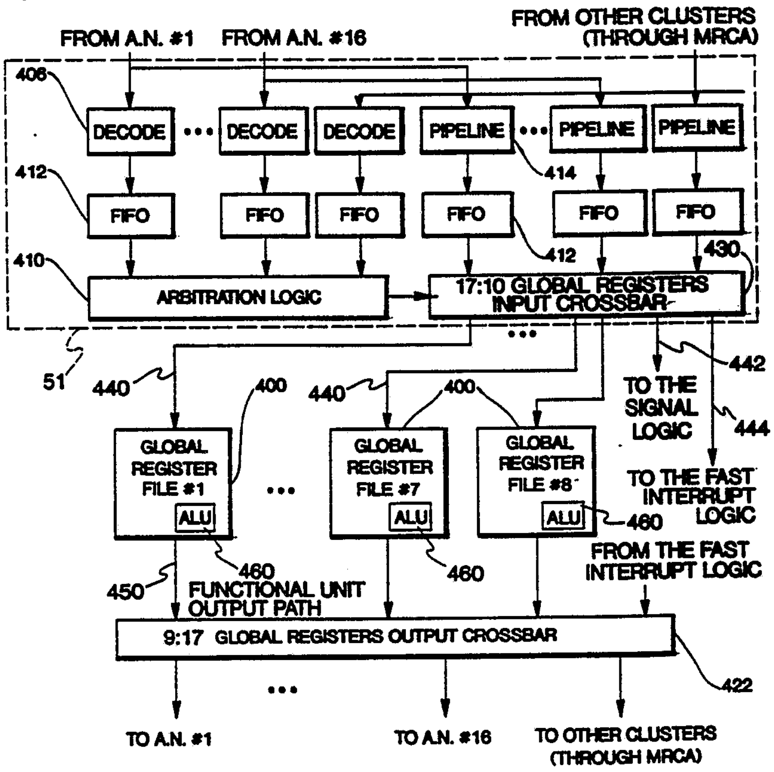
(72) Inventors: BEARD, Douglas, R. ; S10505 Lowes Creek Road, Eleva, WI 54738 (US). SPIX, George, A. ; 3309 Westover Lane, Eau Claire, WI 54701 (US). MILLER, Edward, C. ; 3383 Evergreen Lane, Eau Claire, WI 54701 (US). STROUT, Robert, E., III ; 948 Kern Court, Livermore, CA 94550 (US). SCHOOLER, Anthony, R. ; S7665 Homestead Road, Eau Claire, WI 54701 (US). SILBEY, Alexander, A. ; 2518 West Princeton Avenue, Eau Claire, WI 54703 (US). VANDERWARN, Brandon, D. ; East 2535 Kirk Drive, Eau Claire, WI 54701 (US). WILSON, Jimmie, R. ; 3716 Partridge Run, Eau Claire, WI 54701 (US). HESSEL, Richard, E. ; 3618 Altoona Avenue, Altoona, WI 54720 (US). PHELPS, Andrew, E. ; 6551 Hillview Road, Eau Claire, WI 54701 (US).

(74) Agents: PEDERSEN, Brad, D.; Dorsey & Whitney, 2200 First Bank Place East, Minneapolis, MN 55402 (US) et al.

(81) Designated States: AT (European patent), AU, BE (European patent), CA, CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).

Published
With international search report.

(54) Title: GLOBAL REGISTERS FOR A MULTIPROCESSOR SYSTEM



(57) Abstract

Global registers (16) for a multiprocessor system support multiple parallel access paths for simultaneous operations on separate sets of global registers (16), each set of global registers referred to as a global register file (400). An arbitration mechanism (51) associated with the global registers (16) is used for resolving multiple, simultaneous requests to a single global register file (400). An arithmetic and logical unit (ALU) (460) is also associated with each global register file (400) for allowing atomic arithmetic operations to be performed on the entire register value for any of the global registers in that global register file (400).

GLOBAL REGISTERS FOR A MULTIPROCESSOR SYSTEM

TECHNICAL FIELD

This invention relates generally to the field of registers and interconnection techniques for multiprocessor computer and electronic logic systems. More particularly, the present invention relates to a system of global registers for a multiprocessor system that provides for an efficient and distributed mechanism that is capable of providing an atomic resource allocation mechanism for interconnecting and coordinating the multiprocessors in such a system.

BACKGROUND ART

The use of global registers as part of the interconnection and control mechanisms for multiprocessor systems is well known in the prior art. Global registers are registers that are generally accessible to all requestors in a multiprocessor system. In an article by E. W. Dijkstra entitled "Co-operating Sequential Processes," in F. Genuys (ed.), Programming Languages (Academic Press, New York 1968), Dijkstra describes the use of global registers for a semaphore operation to control the operational flow of a multiprocessor system. The use of global registers as part of a semaphore operation is typically limited to minimally parallel supercomputers and hierarchical memory supercomputers. Massively parallel supercomputers, by their very architecture, do not have a use for a set of global registers as control and coordination of the processors is accomplished via a message passing scheme.

Most prior art global register systems utilize some form of hardware dependent interlock mechanism to accomplish the semaphore function. For example, in the architecture for the Cray X-MP supercomputer developed by Cray Research, Inc., that is the subject of U.S. Patent No. 4,363,942, a deadlock interrupt means is used to coordinate requests to the global registers by two high-speed processors. While this type of tightly-coupled, direct-connection method is an efficient means for coordinating two high speed processors, the hardware deadlock interrupt mechanism described in that patent is most effective when both the number of processors being coupled together and the number of global registers involved are relatively small.

In addition, most prior art global register systems have been implemented using a small set of global registers with relatively few access paths. Because minimally parallel supercomputers typically operate with a centralized operating system, many of the potential conflicts for global register usage are controlled by the centralized operating system which can limit the number of processors assigned to access a given global register. As a result, there has generally been no need in the prior art to provide for a large number of global registers capable of distributed and/or multithreaded processing on the contents of more than one global register at a time.

The design of global registers for supercomputers has been problematic in prior art multiprocessor systems, even with the limited design requirements of those architectures. In an effort to increase the processing speed and flexibility of multiprocessor computer processing systems, the previously filed parent application to the present invention entitled CLUSTER ARCHITECTURE FOR A HIGHLY PARALLEL SCALAR/VECTOR MULTIPROCESSOR SYSTEM, PCT Serial No.: PCT/US90/07655, provides a cluster architecture that allows a number of processors and external interface ports to make multiple and simultaneous requests to a common set of shared hardware resources. One of those shared hardware resources is a set of global registers. The problem of global register design is further compounded by several important design factors that are utilized in the design of this cluster architecture. First, the global registers must be capable of supporting many multiple requests to the same global register. Second, the global registers must operate in a distributed environment where there is no central scheduler and where portions of the distributed input/output are also allowed direct access to the global registers without processor intervention. Finally, the global registers must be capable of atomic arithmetic operations and atomic resource allocation operations in order to support the software routines for a multithreaded operating system that use shared-variable synchronization and anarchy-based scheduling to allocate work and coordinate access to common data structures used by the operating system.

The problem of global register design has generally been managed in prior art supercomputers by assigning a single, central scheduling processor to keep track of what resources were currently being used by which processor. In the distributed access architecture of the cluster architecture for a multiprocessor system, access to all shared resources, including global registers, is equal and democratic and there is no central scheduler. Consequently, a new design for global registers for a distributed access architecture multiprocessor system is needed.

SUMMARY OF THE INVENTION

The present invention provides for global registers for a multiprocessor system that will support multiple parallel access paths for simultaneous operations on separate sets of global registers, each set of global registers being referred to as a global register file. An arbitration mechanism associated with the global registers is used for resolving multiple, simultaneous requests to a single global register file. An arithmetic and logical unit (ALU) is also associated with each global register file for allowing atomic arithmetic operations to be performed on the entire register value for any of the global registers in that global register file.

The global registers of the present invention are a globally accessible resource that may be accessed from any processor or peripheral controller through an external interface port in the multiprocessor system. The global registers support a variety of synchronization primitives to allow the most efficient choice for synchronization primitive, depending upon the particular synchronization task at hand. One of the more notable synchronization primitives of the present invention is the Fetch and Conditional Add (FCA) instruction. The FCA instruction may be used by the software routines for a multithreaded operating system that uses shared-variable synchronization and anarchy-based scheduling to allocate work and coordinate access to common data structures used by the operating system.

In the preferred embodiment, the global registers are implemented as one part of an entire set of common shared hardware resources that are all available to each requestor in a distributed, democratic multiprocessor environment. The global registers are organized as eight global register files within each cluster of the preferred embodiment of the multiprocessor system. The organization of the global registers of the present invention into global register files allows simultaneous access to multiple global register files. In the preferred embodiment, there are 8192 global registers per cluster and 1024 global registers per global register file.

It is an objective of the present invention to provide a set of global registers that will support multiple parallel access paths for simultaneous operations on separate global register files.

Another objective of the present invention is to provide a set of global registers that allow atomic arithmetic operation to be performed on the entire register value for any of the global registers.

A further objective of the present invention is to provide a set of global registers that are capable of supporting a Fetch and Conditional Add (FCA) instruction.

These and other objectives of the present invention will become apparent with reference to the drawings, the detailed description of the preferred embodiment and the appended claims.

DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of the various interconnections among processors, external interface ports and the global registers in a single cluster of a multiprocessor system in the preferred embodiment of the present invention.

Figs. 2a and 2b are a block diagram of a four cluster implementation of the preferred embodiment of a multiprocessor system.

Fig. 3 is a block diagram showing the implementation of the global registers as part of the NRCA means of the preferred embodiment of the multiprocessor system.

Fig. 4 is a block diagram showing the arbitration logic and cross bar switch mechanisms for the various global register files of the present invention.

Fig. 5 is a is a more detailed block diagram of Fig. 4 showing the data and address pipelines for the global registers.

Fig. 6 is a schematic representation of the logical and physical address maps for the global registers.

Fig. 7 is a more detailed block diagram of Fig. 4 showing the address and data lines for a single global register file and the arithmetic logical unit associated with that global register file.

Fig. 8 is a schematic representation showing the global register addressing.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to Fig. 1, the architecture of a single multiprocessor cluster of the preferred embodiment of the multiprocessor system for use with the present invention will be described. The preferred cluster architecture for a highly parallel scalar/vector multiprocessor system is capable of supporting a plurality of high-speed processors 10 sharing a large set of shared resources 12 (e.g., main memory 14, global registers 16, and interrupt mechanisms 18). The processors 10 are capable of both vector and scalar parallel processing and are connected to the shared resources 12 through an arbitration node means 20. Also connected through the arbitration node means 20 are a plurality of external interface ports 22 and input/output concentrators (IOC) 24 which are further connected to a variety of external data sources 26. The external data sources 26 may include a secondary memory system (SMS) 28 linked to the input/output concentrator 24 via a high speed channel 30. The external data sources 26 may also include a variety of other peripheral devices and interfaces 32 linked to the input/output concentrator 24 via one or more standard channels 34. The peripheral devices and interfaces 32 may include disk storage systems, tape storage system, printers, external processors, and communication networks. Together, the processors 10, shared resources 12, arbitration node means 20 and external interface ports 22 comprise a single multiprocessor cluster 40 for a highly parallel multiprocessor system in accordance with the preferred embodiment of the present invention.

The preferred embodiment of the multiprocessor clusters 40 overcomes the direct-connection interface problems of present shared-memory supercomputers by physically organizing the processors 10, shared resources 12, arbitration node means 20 and external interface ports 22 into one or more clusters 40. In the preferred embodiment shown in Fig. 2a and 2b, there are four clusters: 40a, 40b, 40c and 40d. Each of the clusters 40a, 40b, 40c and 40d physically has its own set of processors 10a, 10b, 10c and 10d, shared resources 12a, 12b, 12c and 12d, and external interface ports 22a, 22b, 22c and 22d that are associated with that cluster. The clusters 40a, 40b, 40c and 40d are interconnected through a remote cluster adapter 42 that is a logical part of each arbitration nodes means 20a, 20b, 20c and 20d. Although the clusters 40a, 40b, 40c and 40d are physically separated, the logical organization of the clusters and the physical interconnection through the remote cluster adapter 42 enables the desired symmetrical access to all of the shared resources 12a, 12b, 12c and 12d across all of the clusters 40a, 40b, 40c and 40d.

In the present invention, any and all processors 10 and external interface ports 22 may simultaneously access the same or different global registers 16 in any given clock cycle. The global registers 16 are physically and logically organized into global register files. References to global registers within a given global register file are serialized over a number of clock cycles and take place at the rate of one operation every clock cycle. Simultaneous references to registers in separate global register files take place in the same clock cycle. Global register logic resolves any access contention within a global register file by serially granting access to each requestor so that only one operation is performed at a time. References to a single global register within a global register file are processed in the order in which they arrive. The preferred embodiment provides addressing for a contiguous block of 32,768 global registers located among the four clusters 40. There are 8192 global registers per cluster 40. The global registers are organized within each cluster 40 as eight global register files so that accesses to different global register files can occur simultaneously.

Referring now to Fig. 3, the preferred embodiment of the global registers 16 is described. In the preferred embodiment, the global registers are associated with the logic for the NRCA means 46 in the remote cluster adapter 42. While the physical location of the global register 16 is preferably in the remote cluster adapter 42 for the preferred multiprocessor system, it will be understood that the location and global registers 16 can be accomplished by a variety of designs, depending upon the architecture and layout of the multiprocessor system that is using them.

There are sixteen NRCA ports 47 in the arbitration node means 20 (one per arbitration node 44) that provide an access path to the global registers 16 from the thirty-two processors 10 and thirty-two external interface ports 22 in a cluster 40. Each NRCA port 47 is shared by two processors 10 and two external interface ports 22 and is accessed over the path 52. A similar port 49 services inter-cluster requests for the global registers 16 in the cluster 40 as received by the MRCA means 48 and accessed over the path 56. It will be recognized that access time to global registers 16 will, in general, be slightly faster than to main memory 14 when requests remain within the same cluster 40. Also, there is no interference between in-cluster memory traffic and global register traffic because requests are communicated over different paths.

In the NRCA means 46, a cross bar/arbitration means 51 and an remote cluster crossbar 53 receive requests from the sixteen arbitration nodes 44 and the MRCA means 48. Access to the NRCA means 46 via paths 52 and 56 are routed through the cross bar/arbitration means 51 to direct the access to and from the appropriate logic in the NRCA means 46 for the global register 16 and the interrupt mechanism 18 comprised of signal logic 31 and fast interrupt logic 33. For the global registers 16, an arbitration decision requires address information to select the target register and control information to determine the operation to be performed as described in greater detail hereinafter. This information is transmitted to the NRCA means 46 along with the data. The address and control can be for data to be sent to global registers 16 or to signal logic 31 or fast interrupt logic 33.

An important feature of the global registers 16 of the present invention is their ability to perform a read-modify-write operation in a single uninterruptable operation. This feature is used to provide atomic resource allocation mechanisms that are used by the operating system and input/output system for creating a multiprocessor system that has integrated support for distributed and multithreaded operations throughout the multiprocessor system. Several versions of such an atomic resource allocation mechanism are supported. The atomic global register operations are as follows:

Test And Set (TAS) - Data supplied by the originator of the request is logically ORed with data in the register, and the result is placed in the selected register. Contents of the register prior to modification are returned to the originator of the request.

Set (SET) - Data supplied by the originator of the request is logically ORed with data in the register, and the result is placed in the register.

Clear (CLR) - Selected bits in the selected global register are cleared set in resposne to data supplied by the originator of the request.

Fetch And Add (FAA) - Data supplied by the originator of the request is arithmetically added to the value in the register, and the result is placed in the register. Register contents prior to the addition are returned to the originator of the request.

Fetch and Conditional Add (FCA) - Data supplied by the originator of the request is arithmetically added to the value in the register, and the result is placed in the register if the result of the add is greater than or equal to zero. If the result of the add is less than zero, the register contents are not changed. Register contents prior to the addition are returned to the originator of the request.

Add (ADD) - Data supplied by the originator of the request is arithmetically added to the value in the register, and the result is placed in the register.

Swap (SWAP) - Data supplied by the originator of the request is written into the selected register. Contents of the register prior to modification are returned to the originator of the request.

Read (READ) - Contents of the register are returned to the originator of the request.

Write (WRITE) - Data supplied by the originator of the request is written into the selected register.

A more detailed description of each of these instructions is set forth in Appendix A which is attached hereto and incorporated by reference.

Synchronization via a semaphore-like operation using the global registers 16 is accomplished by the Test and Set (TAS) instruction and a software convention to make a specific global register 16 contain semaphore information. The TAS instruction causes a number of bits to be set in a global register 16. However, before the data is modified, the contents of the global register 16 are sent back to the issuing processor 10. The processor 10 then checks to see if these bits are different than the bits originally sent. If they are different, the processor 10 has acquired the semaphore because only one register at a time can change any data in a global register 16. If the bits are the same, the software may loop back to retry the TAS operation.

Besides the obvious rapid synchronization capability required to support parallel processing, additional functionality has been designed into the global registers 16 and the overall architecture. At compilation, each process determines how many processors 10 it can use for various portions of the code. This value can be placed in its active global register set. Any free processor is, by definition, in the operating system and can search for potential work simply by changing the GMASK and GOFFSET control registers as described in further detail in connection with Fig. 8 and scanning an active process's processor request number.

Processors, when added to a process, decrement the processor request number. The operating system can easily add processors to a process, or pull processors from a process, based on need and usage. The fetch and conditionally add (FCA) instruction ensures that no more processors than necessary are added to a process. This instruction also facilitates the parallel loop handling capabilities of multiple processors.

Referring now to Fig. 4, the cross bar/arbitration means 51 is described in greater detail. The flow begins with data from one of the arbitration nodes 44 which has been buffered by the NRCA means 46. As each request is received at the NRCA input registers 510 (Fig. 5), decode logic 406 decodes the request to be presented to a global register arbitration network 410. If simultaneous requests come in for multiple global registers 16 in the same global register file 400, these requests are handled in a pipelined manner by the FIFO's 412, pipelines 414 and the global register arbitration network 410. Priority is assigned by a FIFO (first in, first out) scheme supplemented with a multiple request toggling priority scheme. The global register arbitration network 410 uses this type of arbitration logic, or its equivalent, to prioritize simultaneous requests to the same global register file 400. When priority is determined by the arbitration network 410, a 17x10 crossbar switch means 430 matches the request in the FIFO 412 with the appropriate global register file 400. A plurality of NRCA input registers 510 (Fig. 5) provide seventeen paths into the global registers input crossbar 430. There are eight paths 440 out of the global registers input crossbar 430 to the global register files 400, one path

442 to the signal logic 31, and one path 444 to the fast interrupt logic 33. After the global register file operation is completed, global register output cross bar 422 routes any output from the operation back to the requesting port.

In the preferred embodiment shown in Fig. 4, each global register file 400 has 1024 general purpose, 64-bit registers. Each global register file 400 also contains a separate Arithmetic and Logical Unit (ALU) operation unit 460, permitting eight separate global register operations in a single clock cycle per cluster. The global register files 400 are interleaved eight ways such that referencing consecutive locations accesses a different file with each reference. In this embodiment, the global registers are implemented using a very fast 1024x64-bit RAM.

As shown in Fig. 5, address and command information travel through a pipeline 520 that is separate from the data pipeline 530. The address and command information is decoded and used to direct data and certain of the address bits to their destination. Because the results of the arbitration decisions are used to direct data to this destination, the data and arbitration results must arrive at the input crossbar 430 in the same clock cycle. Staging registers 560 are added to the data pipeline 530 to adjust the data delay to match the control delay through the address pipeline 520.

As shown in Fig. 6, the arbitration is based on a decode of address bit 13 (the SETN select bit), the three address least significant bit (the global register file select bits), and a four-bit operation code (not shown). If the operation code specifies a signal operation, the address and data information are always sent to the signal logic output port 442. If address bit 13 is set to one, the address, data, and command information are sent to the fast interrupt logic output port 444. Otherwise, the address, control, and data are sent to the global register file output port selected by the three address LSB using one of the paths 440.

The other ten address bits of the logical address (bits 12-3) shown at path 540 in Fig. 5 are not used in the arbitration process. They accompany the data and are used in the functional units to select which register in the file 400 will be modified. The command bits on path 540 are duplicated and carried through the data pipeline as well for use at the destination.

Simultaneous requests from different sources for the same global register file 400 (or for the signal logic 31 or the fast interrupt logic 33) are resolved by the arbitration logic 410 by granting one of the requestors access and delaying any other requests to later cycles. The arbitration address pipeline registers 520 hold any requests that cannot be immediately serviced in the Address Pipeline FIFO 570. In any single Data Pipeline FIFO 580, the data are submitted serially. Similarly, requests in the Address Pipeline FIFO 570 are handled serially. For example, data B entered later cannot pass data A entered before it. Although data A may be waiting for a busy global register, and data B may be waiting for an available global register, data B can not be processed until data A is finished. Data stays in order within a single queue; no data under Address Control can slip ahead of the data order in Data Address Control.

Ten arbitrations can be handled simultaneously by the arbitration logic 410. If data cannot go, signals 512 and 514 are sent to FIFOs 570 and 580, respectively, instructing them to hold the request at their respective outputs. The FIFOs 570 and 580 then wait for their arbitration decision. Signals (not shown) are sent back to each requestor from the arbitration logic 410 indicating that a request has been removed from the FIFOs 570 and 580. The source uses this signal to determine when the FIFOs 570 and 580 are full. The source stops sending requests when the FIFOs 570 and 580 are full so that no requests are lost. Once an arbitration decision is made, a multiplex select signal 590 is generated that steers the input cross bar 460. This automatically unloads the FIFOs 570 and 580 and sends data to the global register files 400 or the signal logic 31 or the fast interrupt logic 33.

The input crossbar 460 is implemented as ten, 17:1 multiplexors. There is one multiplexor for each of the eight output paths 440, and output paths 442 and 444. The multiplexors are controlled by multiplex select signals 590 from the arbitration logic. The arbitration logic 410 also sends a signal to alert the NRCA means 46 (Fig. 3) that data will be returning to the source via the functional unit output path 450 (Fig. 4). Once the request is granted access, data will return to the NRCA means 46 in a fixed number of cycles. The NRCA logic relies on this fixed interval to determine when to receive the data from the global registers 16 and return it to the processor 10. Data is returned through a 9:17 Global Registers Output Crossbar 422 (signal logic does not return data).

Referring now to Fig. 7, the operation of a single global register file 400 will be described. Data associated with a requested operation enters the global register file 400 through the data to global register pipe 610. Data travels through to global register pipe 610 in four steps. Each of the steps requires a single clock cycle. The four steps in the data to global register pipe 610 are as follows:

1. Load data from the arbitration input crossbar 460 in Fig. 5 into the data pipe input register 627.
2. Perform error detection and load the data into the detection output register 628.
3. Perform error correction and load data into the correction output register 629.
4. Move the data to the data pipeline output register 626.

The register address information associated with a requested operation enters through the address pipe 609. Addresses pass through the pipe in two steps. Each step requires a single clock cycle. The two steps in the address pipe 609 are as follows:

1. Load data from the arbitration input crossbar 460 into the address pipe input register 630.
2. Move the address to the register file read address register 624.

The address information is used to fetch data from the register file 623. The fetched data is modified by combining it with data from the global register pipe 610 in the ALU 460. The modified data is then written back into the register file 623. If the specified operation requires that data be returned to the requestor, the data first fetched from the file 623 is delivered to the NRCA logic via the functional unit output register 631.

The ALU 460 consists of a primary adder 602, a wrap adder 603, and a logical unit 604. These three elements can take two operands from three sources. The primary adder takes one operand from either the file output latch 619 or the ALU output latch 621 via latch 620 and the second operand from the data pipeline output register 626. The wrap adder takes one operand from the ALU output latch 621 via latch 620 and the other from the data pipeline output register 626. The logical unit takes one operand from either the file output latch 619 or the ALU output latch 621 via latch 620 and the second operand from the data pipeline output register 626.

Five clock cycles are required to read a register in the file 623, operate on the data in the ALU 460, and return data to the register. Each of the steps requires a single clock cycle. The five steps are listed:

1. Read the register file 623, load data into the file output latch 619.
2. Move data into the ALU input latch 620.
3. Perform the requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
4. Move data into the file input latch 622
5. Store data back into the selected register in the file 623.

The address delay unit 632 delays the read address used in step 1 by four cycles so that it will be available to use when the modified data is written back to the file 623 in step 5. The address delay unit 632 is loaded from the register file read address register 624 at the end of step 1.

This sequence is followed whenever requests for access to the same register are received no faster than once in five cycles. A second operation on the same register initiated after step 5 above will follow the same sequence.

If requests for access to the same register in the file 623 are received on consecutive clock cycles, a different sequence is followed to ensure that the second operation takes place using the results of the first operation, even though the results of the first operation have not been written back to file 623 at the time the second operation takes place. This sequence of operations for consecutive accesses to the same register is as follows:

1. Read the register file 623, load data into the file output latch 619.
2. Move data into the ALU input latch 620.
3. Perform the first requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
4. Perform the second requested operation, using the wrap adder 603 or the logical unit 604, and load data into the ALU output latch 621.
5. Move data into the file input latch 622.
6. Store data back into the selected register in the file 623.

In the preceding sequence, if the logical unit 604 is used, data is taken from the ALU input latch 620 in step 3 but is taken from the ALU output latch 621 in step 4. Selection is made by the logical unit input mux 625. If an adder is required in the second operation, the wrap adder 603 is used in step 4 because it takes data from the ALU output latch 621. This method ensures that data resulting from the first operation is used in the second operation.

If two requests to the same register are received in a three cycle period separated by a single cycle, the following sequence is used:

1. Read the register file 623, load data into the file output latch 619.
2. Move data into the ALU input latch 620.
3. Perform the first requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
4. Move data to the ALU input latch 620 via path 606.
5. Perform the second requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
6. Move data into the file input latch 622.
7. Store data back into the selected register in the file 623.

In the preceding sequence, if the logical unit 604 is used, data is taken from the ALU input latch 620 in both steps 3 and 5. This method ensures that data resulting from the first operation is used in the second operation.

If two requests to the same register are received in a four cycle period separated by two cycles, the following sequence is used:

1. Read the register file 623, load data into the file output latch 619.
2. Move data into the ALU input latch 620.
3. Perform the first requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
4. Load data into the file output latch 619 via path 607.
5. Move data to the ALU input latch 620.
6. Perform the second requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.

7. Move data into the file input latch 622.
8. Store data back into the selected register in the file 623

In the preceding sequence, if the logical unit 604 is used, data is taken from the ALU input latch 620 in both steps 3 and 6. This method ensures that data resulting from the first operation is used in the second operation.

If two requests to the same register are received in a five cycle period separated by a three cycles, the following sequence is used:

1. Read the register file 623, load data into the file output latch 619.
2. Move data into the ALU input latch 620.
3. Perform the first requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
4. Move data to the file input latch 622.
5. Load data into the file output latch 619 via path 608.
6. Move data to the ALU input latch 620.
7. Perform the second requested operation, using the primary adder 602 or the logical unit 604, and load data into the ALU output latch 621.
8. Move data into the file input latch 622.
9. Store data back into the selected register in the file 623.

In the preceding sequence, if the logical unit 604 is used, data is taken from the ALU input latch 620 in both steps 3 and 7. This method ensures that data resulting from the first operation is used in the second operation.

Data returning to the requestor follows one of four paths to the output, depending on the location of the requested data word in the operation pipeline at the time the request is made. Normally, the requested data word will be located in the register file 623. The sequence of steps to move the contents of the selected register is shown below. Each step requires a single clock cycle:

1. Read the register file 623, and load data into the file output latch 619.
2. Move data into the primary exit register 633 via path 613.

3. Compute and append ECC syndrome bits, and load data and syndrome bits into the primary ECC output register 634.
4. Move data and syndrome bits to the functional unit output register 631 via path 617.
5. Return data and syndrome bits to the NRCA through the global register output cross bar 422 and, through further steps, to the requestor.

If the requested data word has been modified by an operation in the immediately preceding cycle, the following steps are used:

1. Move the previous contents of the referenced register into the ALU input latch 620.
2. Perform the previously requested operation, and move the data into the ALU output latch 621. This step computes the data that has been requested by the current fetch operation.
3. Move data into the alternate exit register 635 via path 638.
4. Compute and append ECC syndrome bits, and load data and syndrome bits into the functional unit output register 631 via path 639.
5. Return data and syndrome bits to the NRCA through the global register output cross bar 422 and, through further steps, to the requestor.

If the requested data word has been modified by an operation two cycles earlier, the following steps are used:

1. Perform the previously requested operation, move the data into the ALU output latch 621. This step computes the data that has been requested by the current fetch operation.
2. Move data into the alternate exit register 635 via path 638.
3. Compute and append ECC syndrome bits, and load data and syndrome bits into the first staging register 636.
4. Move data and syndrome bits to the functional unit output register 631 via path 640.
5. Return data and syndrome bits to the NRCA through the global register output cross bar 422 and, through further steps, to the requestor.

If the requested data word has been modified by an operation three cycles earlier, the following steps are used:

1. Move data from the ALU output latch 621 into the alternate exit register 635 via path 638.
2. Compute and append ECC syndrome bits, and load data and syndrome bits into the first staging register 636.
3. Move data and syndrome bits into the second staging register 637.
4. Move data and syndrome bits to the functional unit output register 631 via path 616.
5. Return data and syndrome bits to the NRCA through the global register output cross bar 422 and, through further steps, to the requestor.

If the requested data word has been modified by an operation four cycles earlier, the following steps are used:

1. Move data from the previously requested operation from the file input latch 622 into the file output latch 619.
2. Move data into the primary exit register 633 via path 613.
3. Compute and append ECC syndrome bits, and load data and syndrome bits into the primary ECC output register 634.
4. Move data and syndrome bits to the functional unit output register 631 via path 617.
5. Return data and syndrome bits to the NRCA through the global register output cross bar 422 and, through further steps, to the requestor.

If the requested data was modified more that four cycles earlier, the normal fetch sequence is used.

The embodiment ensures that any global register operation is completed before another request can be initiated on the same register, giving the appearance that the operation has completed in a single cycle even through multiple cycles are actually required. The pipelined organization allows a new operation to be initiated in the functional unit every cycle, regardless of prior activity. This pipelining, in combination with multiple, parallel paths to multiple functional units, results in the best possible throughput, and hence, the most efficient means for supporting synchronization variables among multiple parallel processes.

Referring now to Fig. 6, the method for accessing the global registers 16 is illustrated. Two methods are shown. The logical address map 710 is used by the processor 10. The physical address map 720 is used by the IOC 24.

Fig. 8 illustrates the global register calculation in the processor 10. The present invention uses a relative addressing scheme for the global registers 16 to eliminate the need for explicit coding of global register addresses in the user's program. Global register address calculations are based on the contents of three processor control registers: GOFFSET 810, GMASK 820 and GBASE 830. Setting GMASK 820 to all ones permits the user to access all of the available global registers 16. GOFFSET 810 and GMASK 820 are protected registers that can be written only by the operating system. Together they define a segment of the collection of global registers 16 that the processor 10 or IOC 24 can address. The three least-significant bits of GOFFSET 810 are assumed to be zero when the address calculation is performed, and the three least-significant bits of GMASK 820 are assumed to be ones.

GBASE 830 is a user-accessible 15-bit register. The value contained in the instruction j field 850 is added to GBASE 830 to form the user address. The j field 850 is considered to be unsigned, and any carry out is ignored. The sum of GBASE 830 and the instruction j field 850 is logically ANDed with the contents of GMASK 820, placing a limit on the maximum displacement into the register set that the user can address. The result of the mask operation is added to the contents of GOFFSET 810. Any carry out is ignored. It should be noted that the two most significant bits of the resulting 15-bit sum are used to select which cluster 40 is accessed. A carry that propagates into the upper two bits as a result of either of the add operations will change the cluster select bits. Note that GOFFSET 810 is a 16-bit register. The 16th bit is used to select the SETN registers associated with the fast interrupt logic 33 and must be zero when accessing the global registers 16.

The address generated by this method allows access to the set of global registers 16 that the operating system assigns any particular processor. All processors 10 could be assigned to one particular set or to different sets of global registers 16, depending on the application and availability of processors. It will be understood that logic in the processor means 10 rearranges the logical address 710 into the physical address 720 used at the NRCA means 46, as shown in the mapping in Fig. 7. It should be noted that address values which specify a binary one in bit position 13 of 720 will address the SETN registers, rather than the global registers 16.

The IOC 24 can also perform global register operations. The operating system reserves for itself any number of global register sets that will be used for parameter passing, interrupt handling, synchronization and input/output control. In the preferred embodiment, the peripherals 32 attached to the various IOCs 24 contain part of the operating system software and are able to access all of the global registers 16 in all clusters 40.

Appendix A

Group 1: Global Registers

Address of g register is calculated as:
GOFFSET + (GMASK & (GBASE + j))

10	addg	*sk*	g[*n*]	Move (sk) + g into g
11	addg	*q*	g[*n*]	Move q + g into g
12	set	*sk*	g[*n*]	Move (sk) I g to g
13	set	*q*	g[*n*]	Move q I g to g
14	clear	*sk*	g[*n*]	Move ~ (sk) & g to g
15	clear	*q*	g[*n*]	Move ~ q & g to g
16	move	*sk*	g[*n*]	Move (sk) to g
17	move	*q*	g[*n*]	Move q to g
18	faa	*sk*	g[*n*],*si*	Move g to si; move (sk)+g to g
19	faa	*q*	g[*n*],*si*	Move g to si; move q+g to g
1a	tas	*sk*	g[*n*],*si*	Move g to si; move (sk) I g to g
1b	tas	*q*	g[*n*],*si*	Move g to si; move q I g to g
1c	fca	*sk*	g[*n*],*si*	Move g to si; add sk to g if the sum would be positive
1d	fca	*q*	g[*n*],*si*	Move g to si; add sk to g if the sum would be positive
1e	swap	*sk*	g[*n*],*si*	Move g to si; move sk to g
1f	move	g[*n*]	*si*	Move g to si

ADDG s g [n] 10 xx nn kk

Add to global register

Assembly syntax	**addg** *sk* g[*n*] Where *n* is an unsigned 8-bit number.
Hold issue conditions	Sk reserved. Scalar memory write port unavailable.
Function	Add (sk) to the global register and leave the result in the global register. The global register modified is register GOFFSET+(GMASK & (GBASE+j)). Any carry out of the sum is ignored.
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

ADDG q g[n] **11 xx nn qq**

Add to global register

Assembly syntax	**addg** *q* g[*n*] Where *q* is a signed 8-bit literal and *n* is an unsigned 8-bit number.
Hold issue conditions	Scalar memory write port unavailable.
Function	Add q to the global register and leave the result in the global register. The global register modified is register

	GOFFSET+(GMASK & (GBASE+j)). Any carry out of the sum is ignored.
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

SET s g[n] **12 xx nn kk**

Set bits in global register

Assembly syntax	set *sk* g[*n*] Where *n* is an unsigned 8-bit number.
Hold issue conditions	Sk reserved. Scalar memory write port unavailable.
Function	"Or" the contents of sk with the global register and leave the result in the global register. The global register modified is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

SET q g[n] **13 xx nn qq**

Set literal bits in global register

Assembly syntax	**set** *q* g[*n*] Where *q* is a signed 8-bit literal and *n* an unsigned 8-bit number.
Hold issue conditions	Scalar memory write port unavailable.
Function	"Or" the contents of sk with the global register and leave the result in the global register. The global register modified is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

CLEAR s g[n] 14 xx nn kk

Clear bits in global register

Assembly syntax	**clear** *sk* g[*n*] Where *n* is an unsigned 8-bit number.
Hold issue conditions	Sk reserved. Scalar memory write port unavailable.
Function	"And" the complement of the contents of sk with the global register and leave the result in the global register. That is, each bit set in sk causes that bit to be cleared in the global register. The global register modified is register GOFFSET+(GMASK & (GBASE+j)).

Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

CLEAR q g[n] **15 xx nn qq**

Clear literal bits in global register

Assembly syntax	clear *q* g[*n*] Where *q* is a signed 8-bit literal and where *n* is an unsigned 8-bit number.
Hold issue conditions	Scalar memory write port unavailable.
Function	"And" the complement of the contents of sk with the global register and leave the result in the global register. That is, each bit set in the literal causes the corresponding bit of the global register to be cleared. The global register modified is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

MOVE s g[n] **16 xx nn kk**

Move S to Global Register

Assembly syntax	**move** *sk* g[*n*] Where *n* is an unsigned 8-bit number.
Hold issue conditions	Sk reserved. Scalar memory write port unavailable.
Function	Move the contents of register sk in to the global register GOFFSET+(GMASK & (GBASE+j)).
Time to completio	Another instruction may issue which reads or modifies the same global register: One cycle.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

MOVE q g[n] **17 xx nn qq**

Move literal to Global Register

Assembly syntax	**move** *q* g[*n*] Where *q* is a signed 8-bit literal and where *n* is an unsigned 8-bit number.
Hold issue conditions	Sk reserved. Scalar memory write port unavailable.
Function	Move the literal q into the global register GOFFSET+(GMASK & (GBASE+j)).
Time to completio	Another instruction may issue which reads or modifies the same global register: One cycle.

Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

FAA s g[n],s **18 ii nn kk**

Fetch And Add S with global register

Assembly syntax	faa *sk* g[*n*],*si* Where *n* is an unsigned 8-bit number.
Hold issue conditions	Si or sk reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. Add the contents of register sk with the global register and leave the result in the global register. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

FAA q g[n],s **19 ii nn qq**

Fetch And Add literal to global register

Assembly syntax	faa *q* g[*n*],*si* Where *q* is a signed 8-bit literal and where *n* is an unsigned 8-bit number.

Hold issue conditions	Si reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. Add the literal to the global register and leave the result in the global register. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

TAS s g[n],s **1a ii nn kk**

Test And Set S with global register

Assembly syntax	**tas** *sk* g[*n*],*si* Where *n* is an unsigned 8-bit number.
Hold issue conditions	Si or sk reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. "Or" the contents of sk with the global register and leave the result in the global register. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.

Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

TAS q g[n],s 1b ii nn qq

Test And Set literal with global register

Assembly syntax	tas *q* g[*n*],*si* Where *q* is a signed 8-bit literal and where *n* is an unsigned 8-bit number.
Hold issue conditions	Si reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. "Or" the contents of sk with the global register and leave the result in the global register. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

FCA s g[n],s 1c ii nn kk

Fetch and Conditionally Add S with global register.

Assembly syntax	fca *sk* g[*n*],*si* Where *n* is an unsigned 8-bit number.
Hold issue	Si or sk reserved.

conditions	Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. "Or" the contents of sk with the global register and leave the result in the global register. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

FCA q g[n],s **1d ii nn qq**

Fetch and Conditionally Add literal to global register.

Assembly syntax	fca *q* g[*n*],*si* Where *q* is a signed 8-bit literal and where *n* is an unsigned 8-bit number.
Hold issue conditions	Si reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register to si. Add the literal to the global register. Place the sum in the global register only if the sum is ≥ 0. The global register used is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.

Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

SWAP g[n]s 1e ii nn kk

Swap contents of global and scalar register

Assembly syntax	**swap** *sk* g[*n*],*si* Where *n* is an unsigned 8-bit number.
Hold issue conditions	Si or sk reserved. Scalar memory read or write port unavailable.
Function	Move the contents of the addressed global register into si. The global register addressed is register GOFFSET+(GMASK & (GBASE+j)). Move contents of sk into the addressed global register.
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

MOVE g[n]s 1f ii nn xx

Move global register to s

Assembly syntax	**move** g[*n*],*si* Where *n* is an unsigned 8-bit number.
Hold issue conditions	Si reserved. Scalar memory read port unavailable.

Function	Move the contents of the addressed global register into si. The global register read is register GOFFSET+(GMASK & (GBASE+j)).
Time to completion	TBD cycles.
Exceptions	None.
Comments	The carry out of GBASE+j and out of GOFFSET+ masked (GBASE+j) is ignored; that is, the g register number is taken to be modulo the number of g registers in a four-cluster system.

Although the description of the preferred embodiment has been presented, it is contemplated that various changes could be made without deviating from the spirit of the present invention. Accordingly, it is intended that the scope of the present invention be dictated by the appended claims rather than by the description of the preferred embodiment.

We claim:

CLAIMS

1. A set of global registers for a multiprocessor system comprising:

a plurality of global register files, each global register file containing one or more register means for storing and manipulating data and addresses for a unique one of the set of global registers;

a plurality of access path means operably connected to the global register files for transferring requests comprised of address, command and data to and from the global register files from one or more requestors in the multiprocessor system;

a plurality of switch means operably connected the global register files and the access path means for routing the requests from the access path means to the selected global register file and register means within the global register file in response to the command and address contained in the request; and

arithmetic and logical unit means operably connected to each global register file for performing arithmetic and logical operations on the data stored in the register means associated with that global register file.

2. The set of global registers of claim 1 wherein the access path means further comprises arbitration means for determining the priority of the requests to the global registers.

3. The set of global registers of claim 1 where the arithmetic and logical unit means is capable of performing atomic resource allocation operations in the form of a single uninterrupted read-modify-write operation.

4. The set of global registers of claim 1 wherein the arithmetic and logical unit means is capable of performing a fetch and conditional add instruction.

5. The set of global registers of claim 1 wherein the requestors are comprised of a plurality of processors and a plurality of external interface ports.

6. The set of global registers of claim 1 wherein the multiprocessor system is a highly parallel multiprocessor systems organized as a plurality of clusters, each cluster including a plurality of requestors comprised of a

plurality of processors and a plurality of external interface ports which are operably connected to a unique subset of the set of global registers and wherein each subset of global registers may be accessed both by the requestors within and without the cluster associated that subset of global registers.

7. The set of global registers of claim 1 wherein the address is comprised of a base address, a mask address, an offset address and an instruction field value.

8. The set of global registers of claim 7 wherein an operating system for the multiprocessor system assigns global registers to a process by specifying the offset address and mask address for the process.

9. The set of global registers of claim 1 wherein the access path means is comprised of an address pipe means for transferring the address and command and a data pipe means for transferring the data.

10. The set of global registers of claim 1 wherein the global registers are capable of pipelining multiple parallel requests on consecutive clock cycles of the multiprocessor system.

11. A set of global registers for a multiprocessor system, the multiprocessor system having one or more requestors including a plurality of processors and a plurality of external interface ports, the set of global registers comprising:

a plurality of global register files, each global register file containing one or more register means for storing and manipulating data and addresses for a unique on of the set of global registers, the address being comprised of a base address, a mask address, an offset address and an instruction field value such that an operating system for the multiprocessor system assigns global registers to a process by specifying the offset address and mask address for the process;

a plurality of access path means operably connected to the global register files for transferring requests comprised of address, command and data to and from the global register files from the requestors in the multiprocessor system, the access path means including:

arbitration means for determining the priority of the requests to the global registers;

address pipe means for transferring the address and command; and

data pipe means for transferring the data;

a plurality of switch means operably connected the global register files and the access path means for routing the requests from the access path means to the selected global register file and register means within the global register file in response to the command and address contained in the request; and

arithmetic and logical unit means operably connected to each global register file for performing arithmetic and logical operations, including atomic resource allocation operations in the form of a single uninterrupted read-modify-write operation, on the data stored in the register means associated with that global register file,

such that the sets of global registers are capable of pipelining multiple parallel requests on consecutive clock cycles of the multiprocessor system.

12. The set of global registers of claim 11 wherein the multiprocessor system is a highly parallel multiprocessor systems organized as a plurality of clusters, each cluster including a plurality of requestors which are operably connected to a unique subset of the set of global registers and wherein each subset of global registers may be accessed both by the requestors within and without the cluster associated that subset of global registers.

13. A highly parallel computer processing system, comprising:

C multiprocessor clusters operably connected to one another, wherein C is an integer between 2 and 256, inclusive, each multiprocessor cluster comprising:

shared resource means for storing and retrieving data and control information,

P processor means for performing computer processing of data and control information, wherein P is an integer between 2 and 256, inclusive;

Q external interface means for transferring data and control information between the shared resource means and one or more external data sources, wherein Q is an integer between 2 and 256, inclusive;

Z arbitration node means operably connected to the processor means, the external interface means, and the shared resource means for symmetrically interconnecting the processor means and the external interface means with the shared resource means, wherein Z is an integer between 1 and 128, inclusive, and the ratio of P to Z is greater than or equal to 2; and

remote cluster adapter means operably connected to remote cluster adapter means in all other of the multiprocessor clusters for allowing the arbitration node means of the multiprocessor cluster to access the shared resource means of all other of the multiprocessor clusters and for allowing all other of the multiprocessor clusters to access the shared resource means of the multiprocessor cluster,

the shared resource means including a unique set of global registers which may be directly accessed by the processor means and the external interface means of the multiprocessor cluster and which may be accessed by the processor means and the external interface means of all other of the multiprocessor clusters through the remote cluster adapter means.

14. The highly parallel computer processing system of claim 13 wherein each of the set of global registers comprises:

a plurality of global register files, each global register file containing one or more register means for storing and manipulating data and addresses for a unique one of the global registers of the set of global registers;

a plurality of access path means operably connected to the global register files for transferring requests comprised of address, command and data to and from the global register files from the processor means and the external interface means;

a plurality of switch means operably connected the global register files and the access path means for routing the requests from the access path means to the selected global register file and register means within the global register file in response to the command and address contained in the request; and arithmetic and logical unit means operably connected to each global register file for performing arithmetic and logical operations on the data stored in the register means associated with that global register file.

15. A multiprocessor cluster for a highly parallel computer processing system, the multiprocessor cluster adapted for connection to other similar multiprocessor clusters in the highly parallel computer processing system, the multiprocessor cluster comprising:

shared resource means for storing and retrieving data and control information;

P processor means for performing computer processing of data and control information, wherein P is an integer between 2 and 256, inclusive;

Q external interface means for transferring data and control information between the shared resource means and one or more external data sources, wherein Q is an integer between 2 and 256, inclusive; and

Z arbitration node means operably connected to the processor means, the external interface means, and the shared resource means for symmetrically interfacing the processor means and the external interface means with the shared resource means, wherein Z is an integer between 2 and 128, inclusive, and the ratio of P to Z is greater than or equal to 2,

the shared resource means including a unique set of global registers which may be directly accessed by the processor means and the external interface means of the multiprocessor cluster and which may be accessed by all other of the multiprocessor clusters.

16. The multiprocessor cluster for a highly parallel computer processing system of claim 15 wherein each of the set of global registers comprises:

a plurality of global register files, each global register file containing one or more register means for storing and manipulating data and addresses for a unique one of the global registers of the set of global registers;

a plurality of access path means operably connected to the global register files for transferring requests comprised of address, command and data to and from the global register files from the processor means and the external interface means;

a plurality of switch means operably connected the global register files and the access path means for routing the requests from the access path means to the selected global register file and register means within the global register file in response to the command and address contained in the request; and

arithmetic and logical unit means operably connected to each global register file for performing arithmetic and logical operations on the data stored in the register means associated with that global register file.

Fig. 1

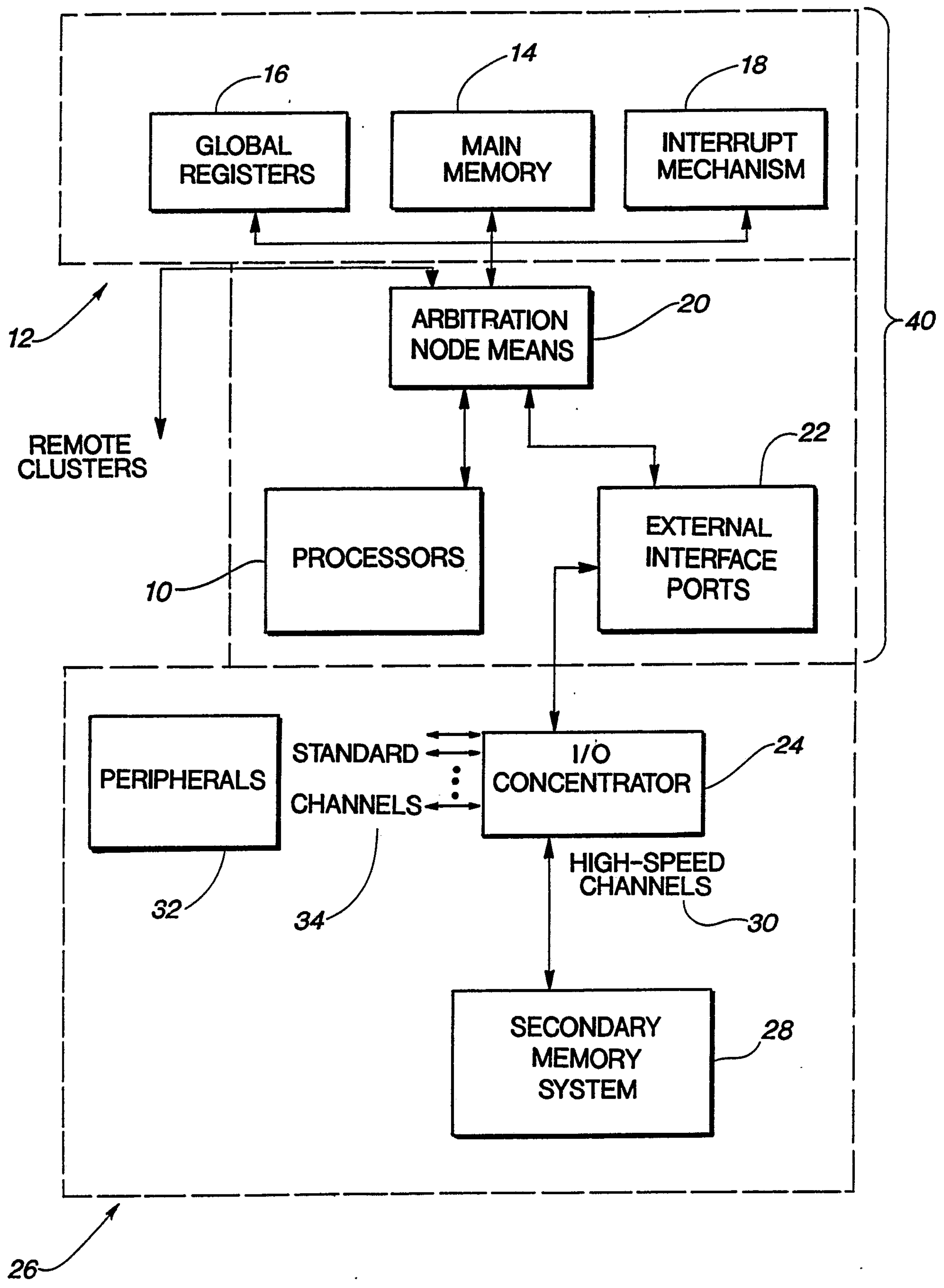

GLOBAL REGISTERS
16
MAIN MEMORY
14
INTERRUPT MECHANISM
18
ARBITRATION NODE MEANS
20
REMOTE CLUSTERS
12
PROCESSORS
10
EXTERNAL INTERFACE PORTS
22
40
PERIPHERALS
32
STANDARD
CHANNELS
34
I/O CONCENTRATOR
24
HIGH-SPEED CHANNELS
30
SECONDARY MEMORY SYSTEM
28
26

Fig. 2a

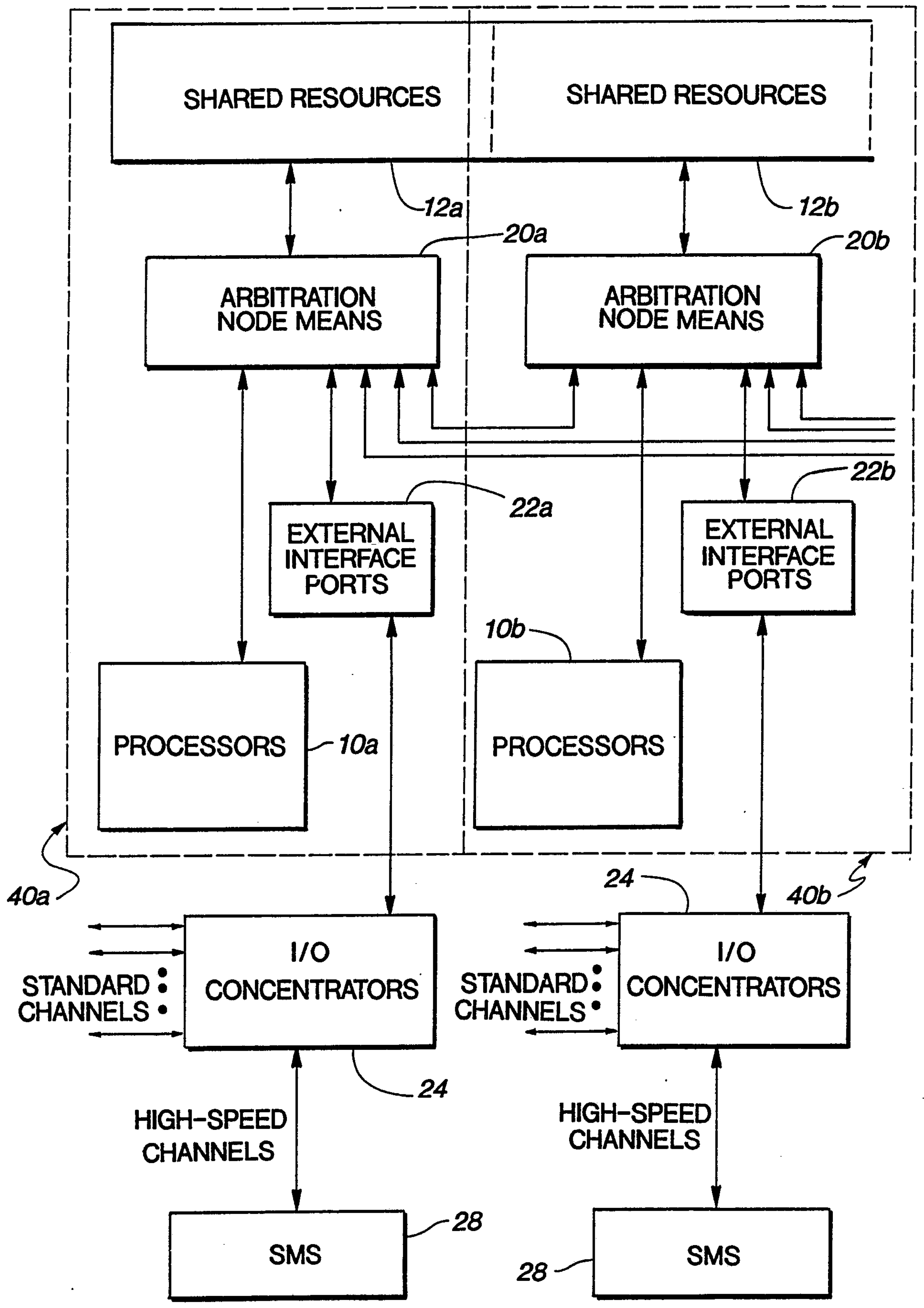
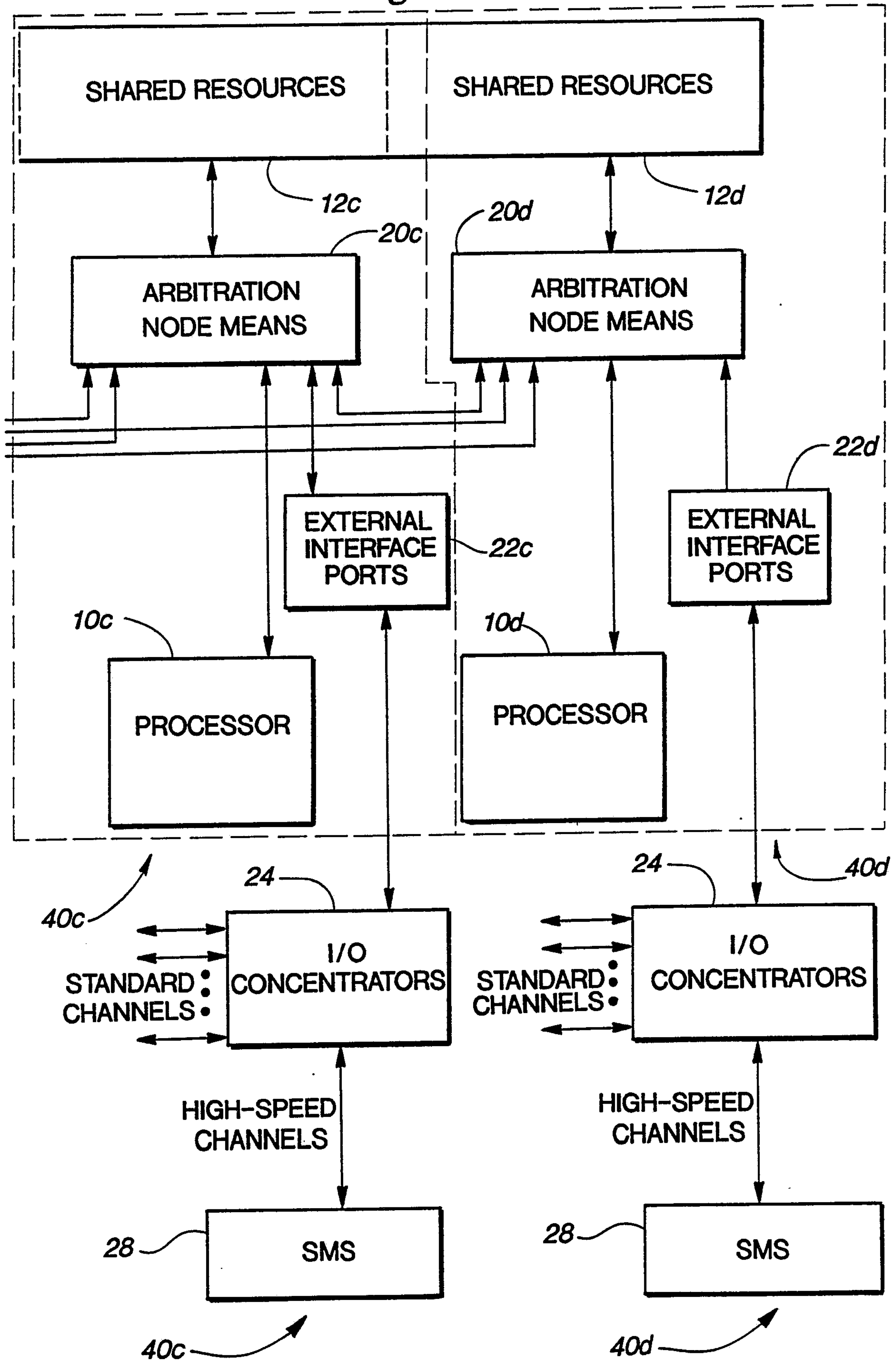

SHARED RESOURCES
SHARED RESOURCES
12a
12b
20a
20b
ARBITRATION NODE MEANS
ARBITRATION NODE MEANS
22a
22b
EXTERNAL INTERFACE PORTS
EXTERNAL INTERFACE PORTS
10b
PROCESSORS
10a
PROCESSORS
40a
24
40b
I/O CONCENTRATORS
I/O CONCENTRATORS
STANDARD CHANNELS
STANDARD CHANNELS
24
HIGH-SPEED CHANNELS
HIGH-SPEED CHANNELS
SMS
28
28
SMS

Fig. 2b


SHARED RESOURCES
SHARED RESOURCES
12c
12d
20d
20c
ARBITRATION NODE MEANS
ARBITRATION NODE MEANS
22d
EXTERNAL INTERFACE PORTS
22c
EXTERNAL INTERFACE PORTS
10c
10d
PROCESSOR
PROCESSOR
40c
24
24
40d
I/O CONCENTRATORS
STANDARD CHANNELS
STANDARD CHANNELS
I/O CONCENTRATORS
HIGH-SPEED CHANNELS
HIGH-SPEED CHANNELS
28
SMS
28
SMS
40c
40d

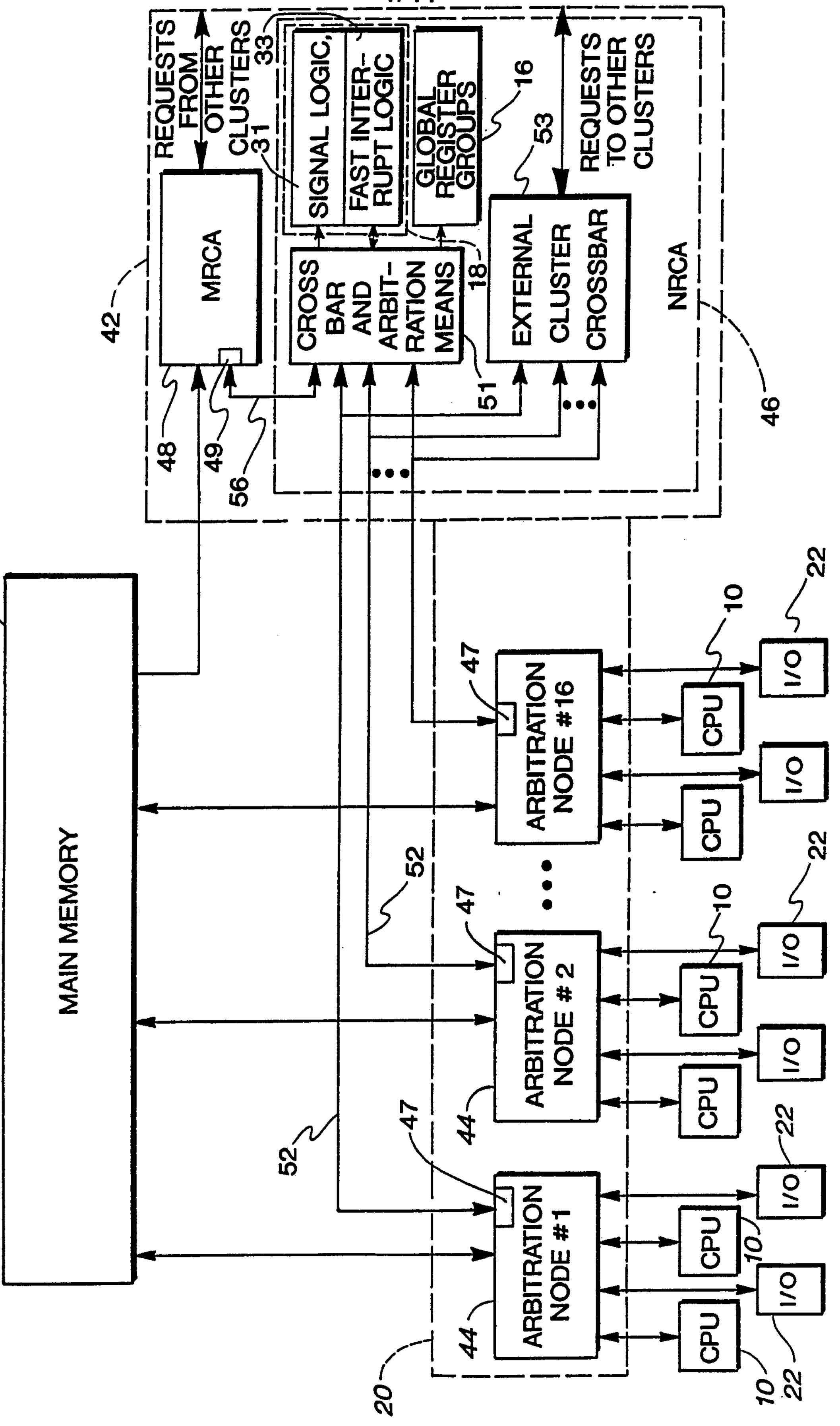

MAIN MEMORY
14
MRCA
42
48
49
56
REQUESTS FROM OTHER CLUSTERS
31
33
SIGNAL LOGIC,
FAST INTER-RUPT LOGIC
GLOBAL REGISTER GROUPS
16
18
CROSS BAR AND ARBIT-RATION MEANS
51
EXTERNAL CLUSTER CROSSBAR
53
REQUESTS TO OTHER CLUSTERS
NRCA
46
ARBITRATION NODE #16
ARBITRATION NODE # 2
ARBITRATION NODE #1
47
44
52
20
CPU
I/O
10
22

Fig. 3

Fig. 4

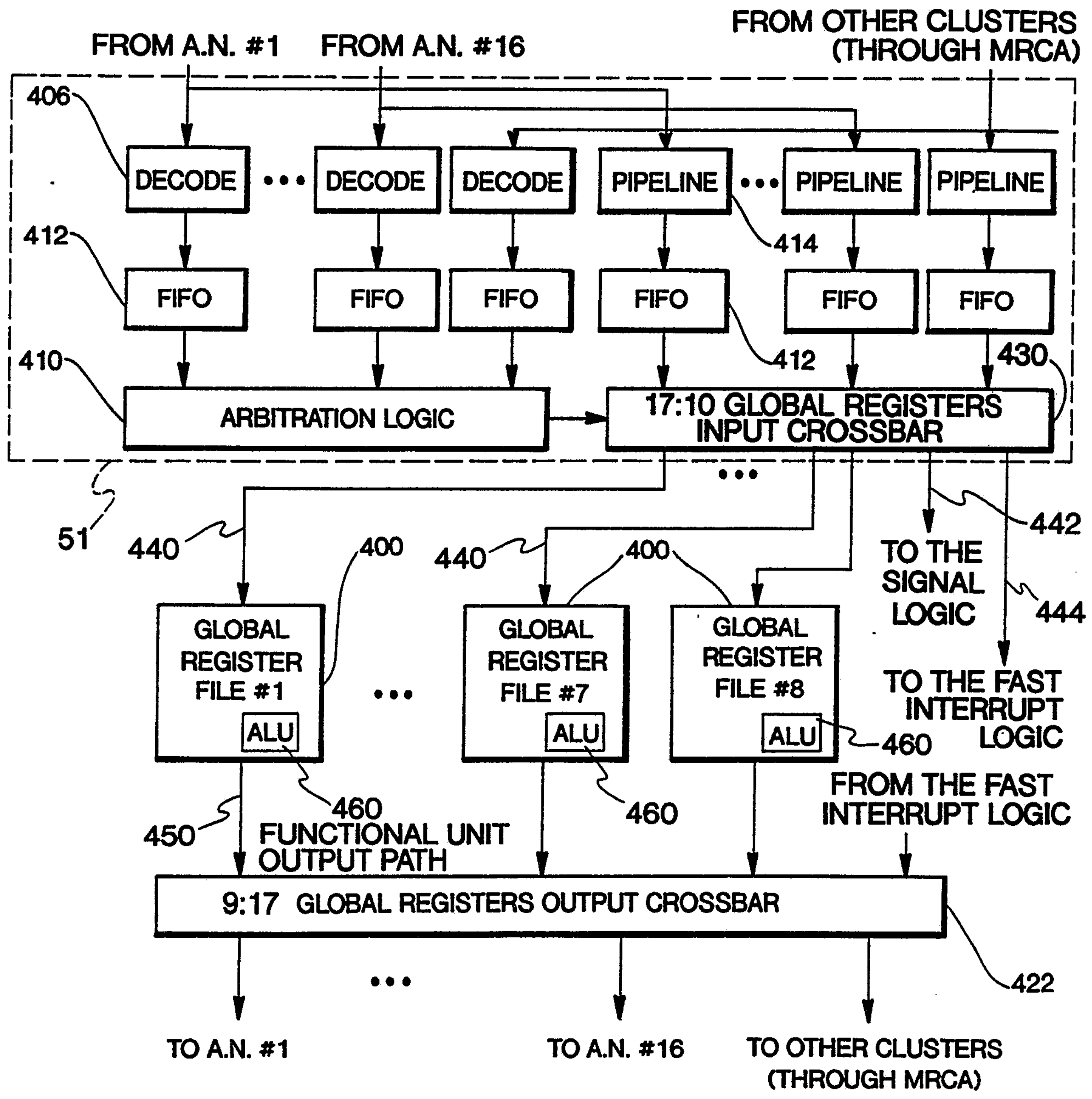
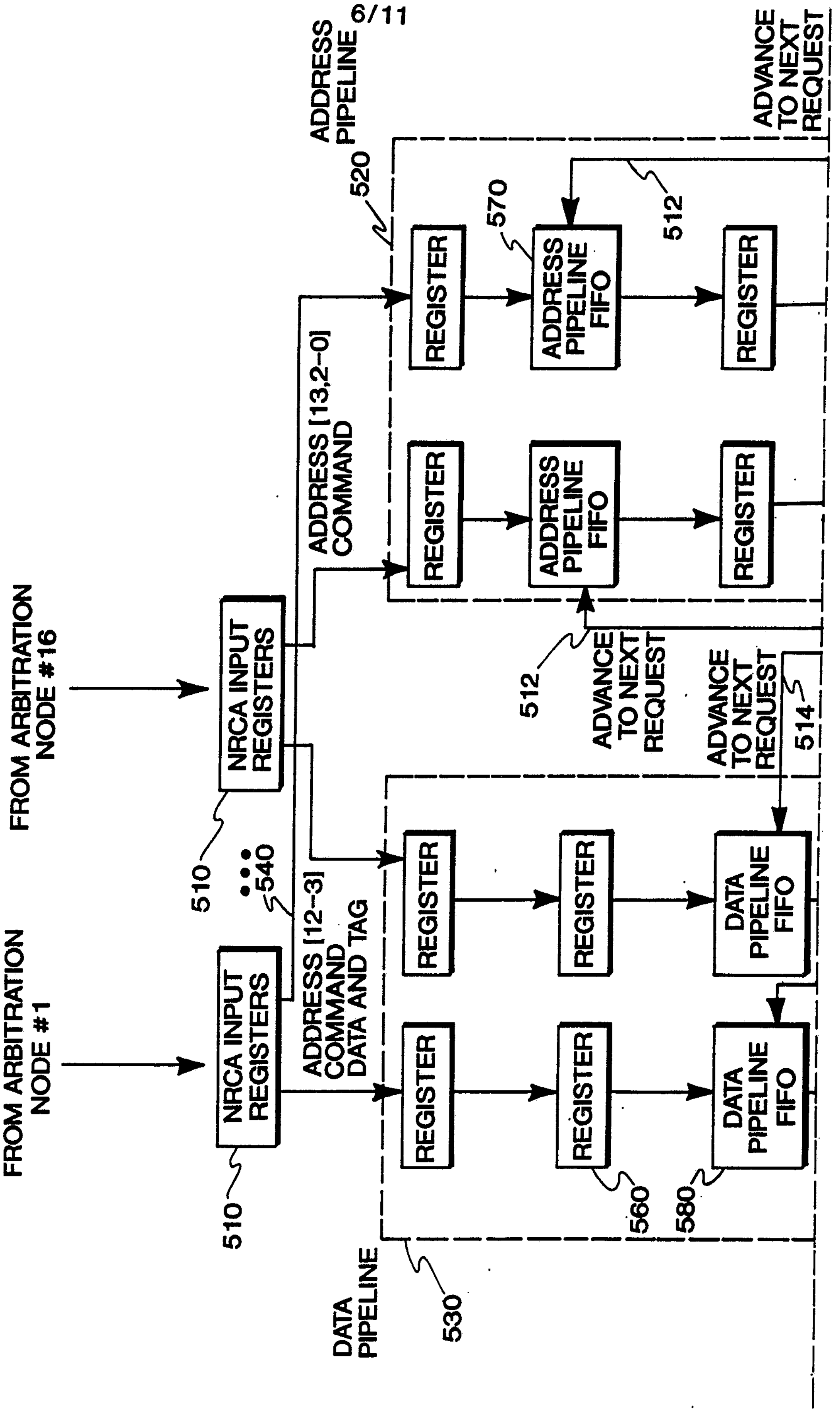

FROM A.N. #1
FROM A.N. #16
FROM OTHER CLUSTERS (THROUGH MRCA)
406
DECODE
DECODE
DECODE
PIPELINE
PIPELINE
PIPELINE
414
412
FIFO
FIFO
FIFO
FIFO
FIFO
FIFO
412
410
ARBITRATION LOGIC
17:10 GLOBAL REGISTERS INPUT CROSSBAR
430
51
440
400
440
400
442
444
TO THE SIGNAL LOGIC
GLOBAL REGISTER FILE #1
GLOBAL REGISTER FILE #7
GLOBAL REGISTER FILE #8
ALU
ALU
ALU
460
460
460
TO THE FAST INTERRUPT LOGIC
450
FUNCTIONAL UNIT OUTPUT PATH
FROM THE FAST INTERRUPT LOGIC
9:17 GLOBAL REGISTERS OUTPUT CROSSBAR
422
TO A.N. #1
TO A.N. #16
TO OTHER CLUSTERS (THROUGH MRCA)

Fig. 5a


FROM ARBITRATION NODE #1
FROM ARBITRATION NODE #16
NRCA INPUT REGISTERS
NRCA INPUT REGISTERS
510
510
540
ADDRESS [12–3] COMMAND DATA AND TAG
ADDRESS [13,2–0] COMMAND
ADDRESS PIPELINE
520
570
REGISTER
ADDRESS PIPELINE FIFO
512
ADVANCE TO NEXT REQUEST
514
DATA PIPELINE
530
560
580
DATA PIPELINE FIFO

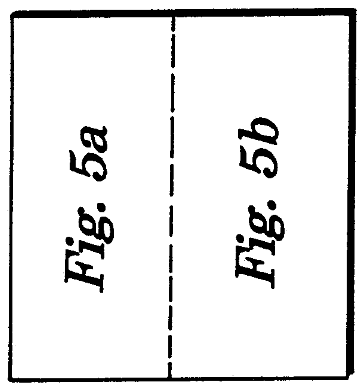


Fig. 5a
Fig. 5b

Fig. 5

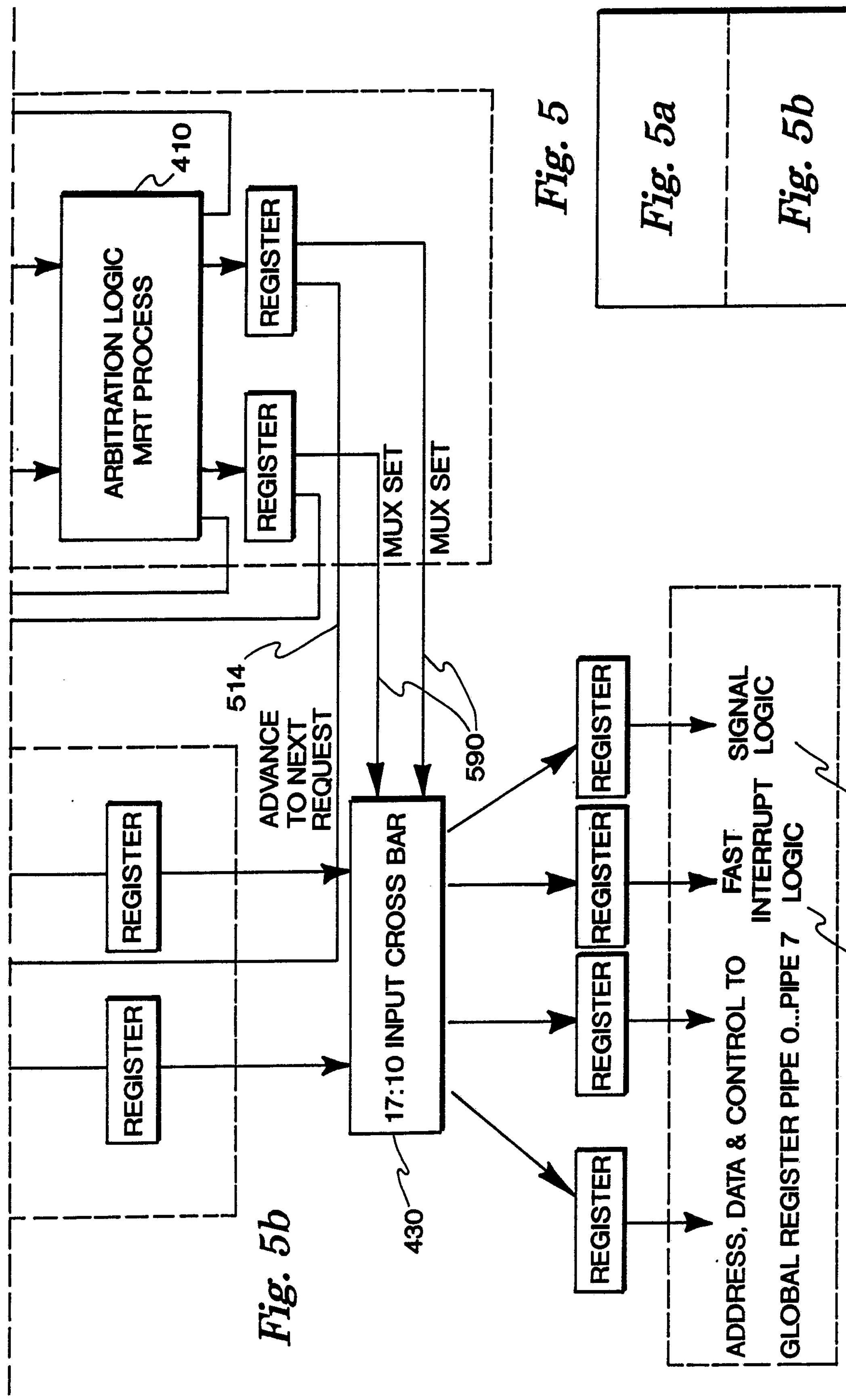

410
ARBITRATION LOGIC
MRT PROCESS
REGISTER
REGISTER
MUX SET
MUX SET
514
ADVANCE TO NEXT REQUEST
590
REGISTER
REGISTER
17:10 INPUT CROSS BAR
430
REGISTER
REGISTER
REGISTER
REGISTER
SIGNAL LOGIC
FAST INTERRUPT LOGIC
ADDRESS, DATA & CONTROL TO GLOBAL REGISTER PIPE 0...PIPE 7
31
33

Fig. 5b

Fig. 6

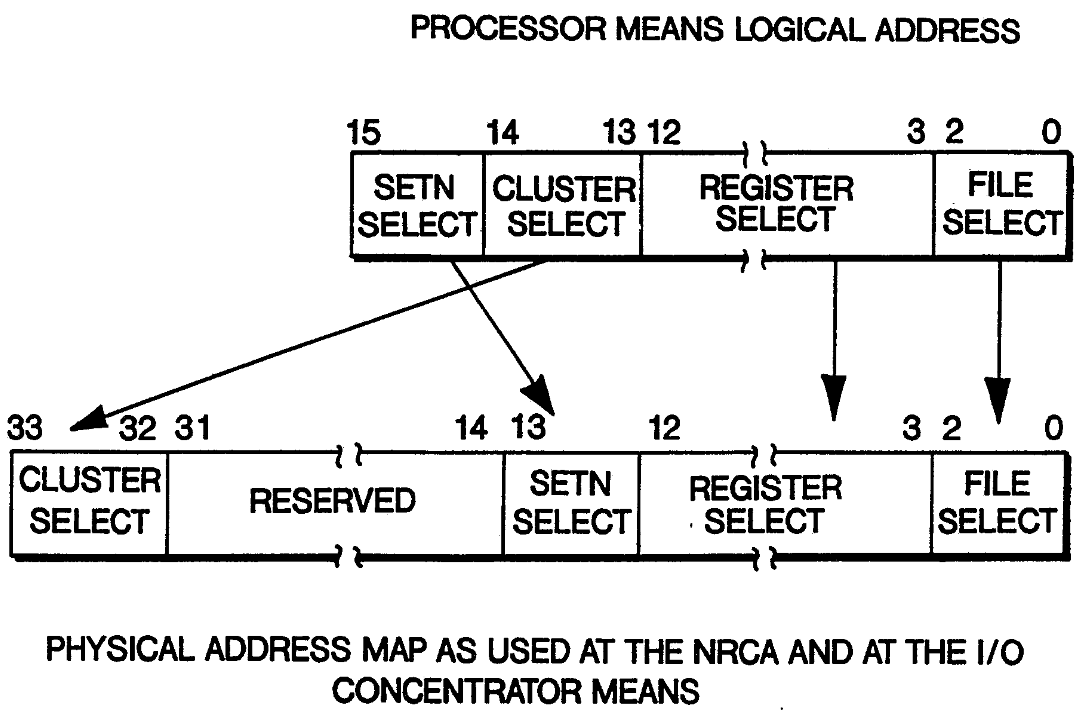

PROCESSOR MEANS LOGICAL ADDRESS
15
14
13
12
3
2
0
SETN SELECT
CLUSTER SELECT
REGISTER SELECT
FILE SELECT
33
32
31
14
13
12
3
2
0
CLUSTER SELECT
RESERVED
SETN SELECT
REGISTER SELECT
FILE SELECT
PHYSICAL ADDRESS MAP AS USED AT THE NRCA AND AT THE I/O CONCENTRATOR MEANS

Fig. 7a

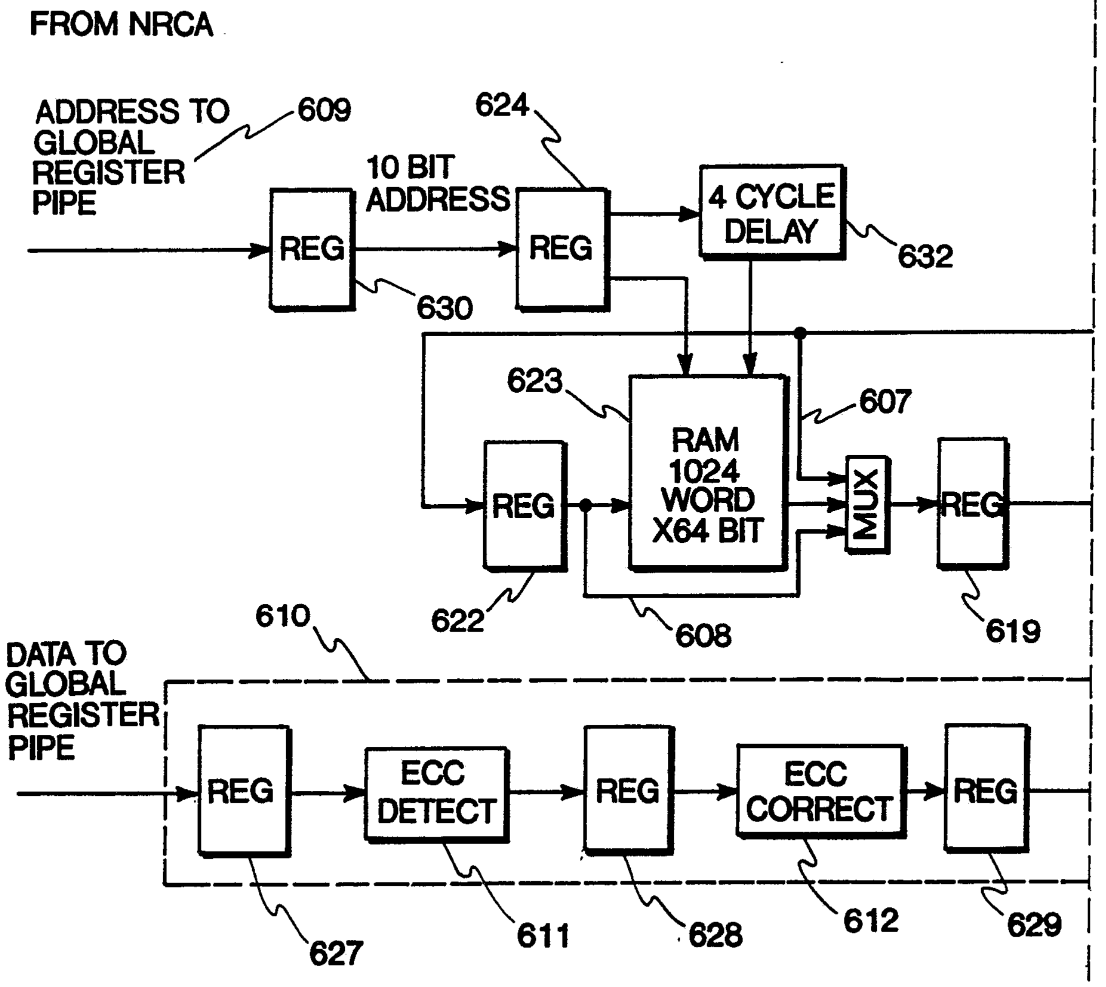

FROM NRCA
ADDRESS TO GLOBAL REGISTER PIPE
609
10 BIT ADDRESS
624
REG
630
REG
4 CYCLE DELAY
632
623
RAM 1024 WORD X64 BIT
607
REG
MUX
REG
622
608
619
610
DATA TO GLOBAL REGISTER PIPE
REG
ECC DETECT
REG
ECC CORRECT
REG
627
611
628
612
629

Fig. 7

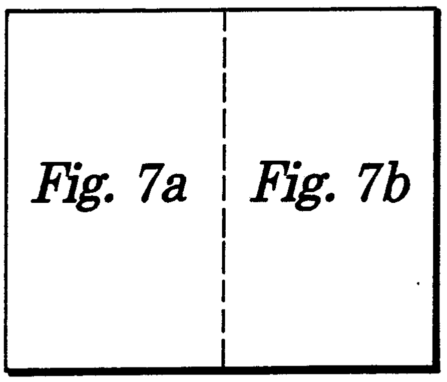


Fig. 7a
Fig. 7b

Fig. 7b

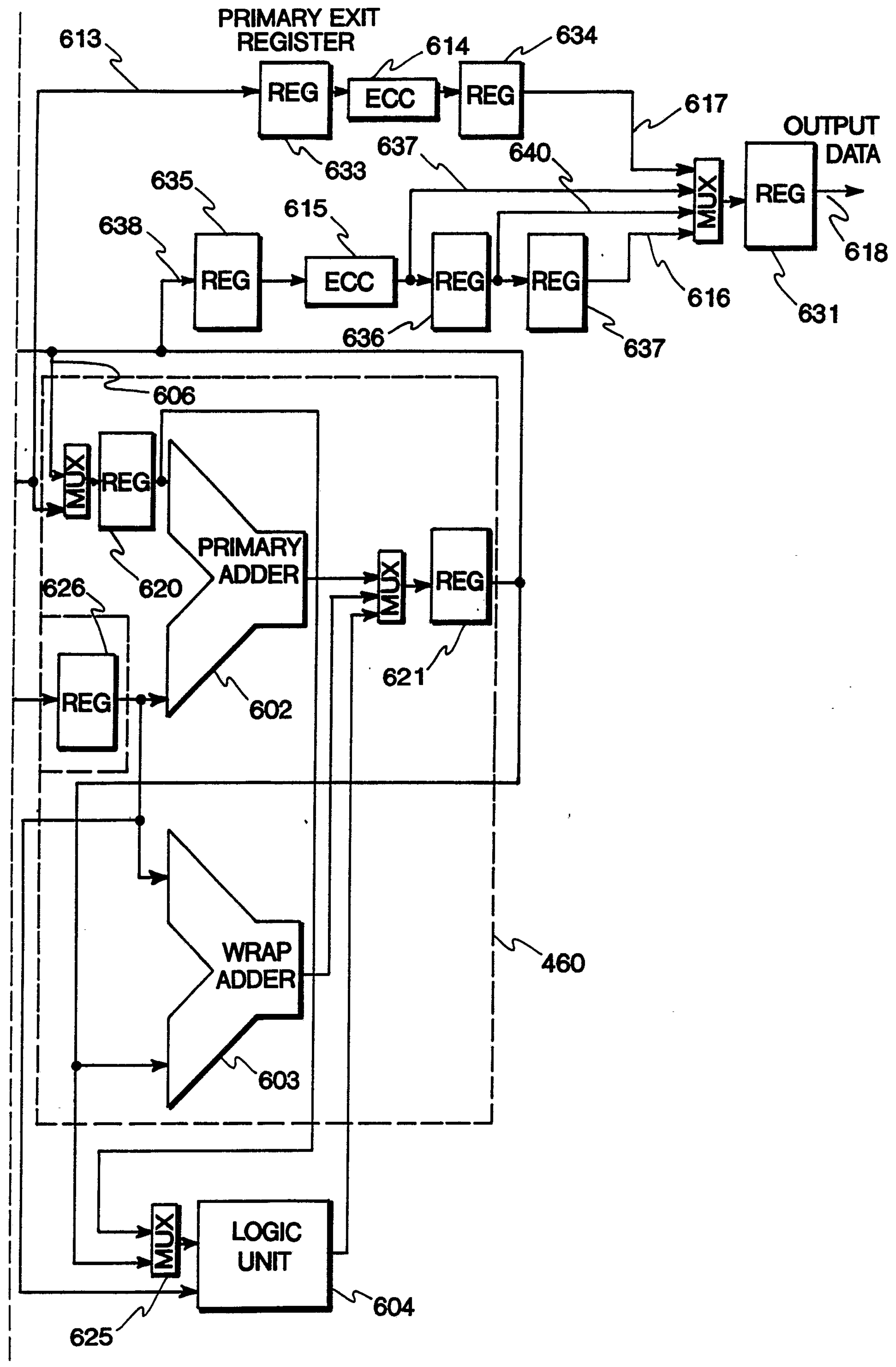
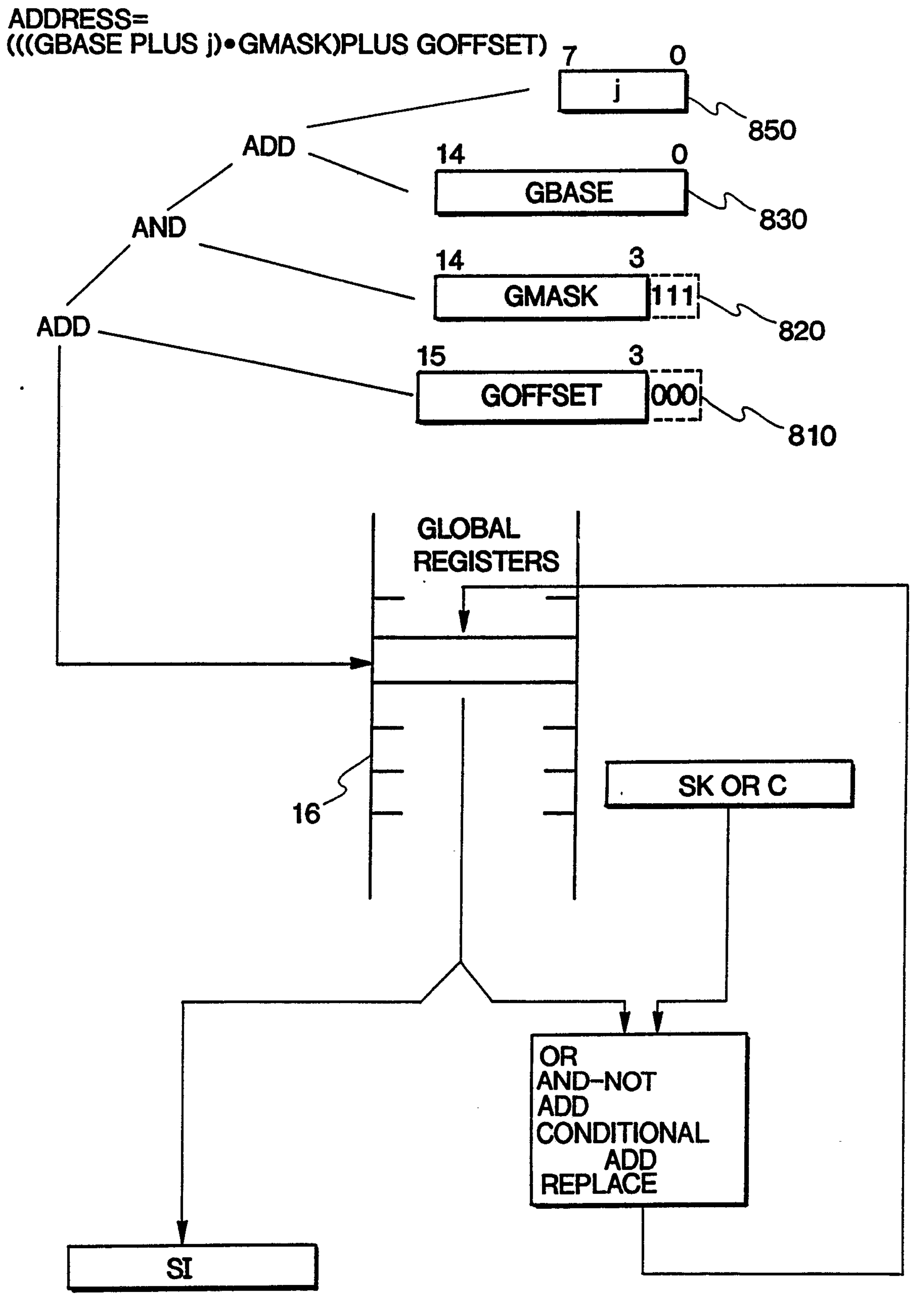

PRIMARY EXIT REGISTER
613
614
634
REG
ECC
REG
617
OUTPUT DATA
633
637
640
635
615
638
REG
ECC
REG
REG
MUX
REG
618
616
631
636
637
606
MUX
REG
620
PRIMARY ADDER
MUX
REG
626
621
REG
602
WRAP ADDER
460
603
MUX
LOGIC UNIT
604
625

Fig. 8


ADDRESS=
(((GBASE PLUS j)•GMASK)PLUS GOFFSET)
7
0
j
850
ADD
14
0
GBASE
830
AND
14
3
GMASK
111
820
ADD
15
3
GOFFSET
000
810
GLOBAL
REGISTERS
16
SK OR C
OR
AND-NOT
ADD
CONDITIONAL
ADD
REPLACE
SI

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US91/04058

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC
IPC(5): G06 F 15/16
U.S. Cl.: 364/200

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
U.S. CL.	364/200,900

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category *	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
Y	US, A, 4,924,380 (McKINNEY) 08 May 1990 (See Figures 2 and 5 and the description of elements 10 and 20)	13, 15
Y	US, A, 4,240,143 (BESEMER) 16 December 1980 (See Figure 1 and the description of Globol memory).	1-16
Y	US, A, 4,814,980 (PETERSON) 21 March 1989 (See Figures 1 and 4 and the description of FIFO).	1-16
A	US, A, 4,523,273 (ADAMS) 11 June 1985 (See Figure 2).	1-16

* Special categories of cited documents: [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search: 11 September 1991	Date of Mailing of this International Search Report: 07 OCT 1991
International Searching Authority: ISA/US	Signature of Authorized Officer: [illegible] David Y. Eng.

Form PCT/ISA/210 (second sheet) (Rev.11-87)

D.P. 9/29/91