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

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2386558 A1 2001/03/08

(21) 2 386 558

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

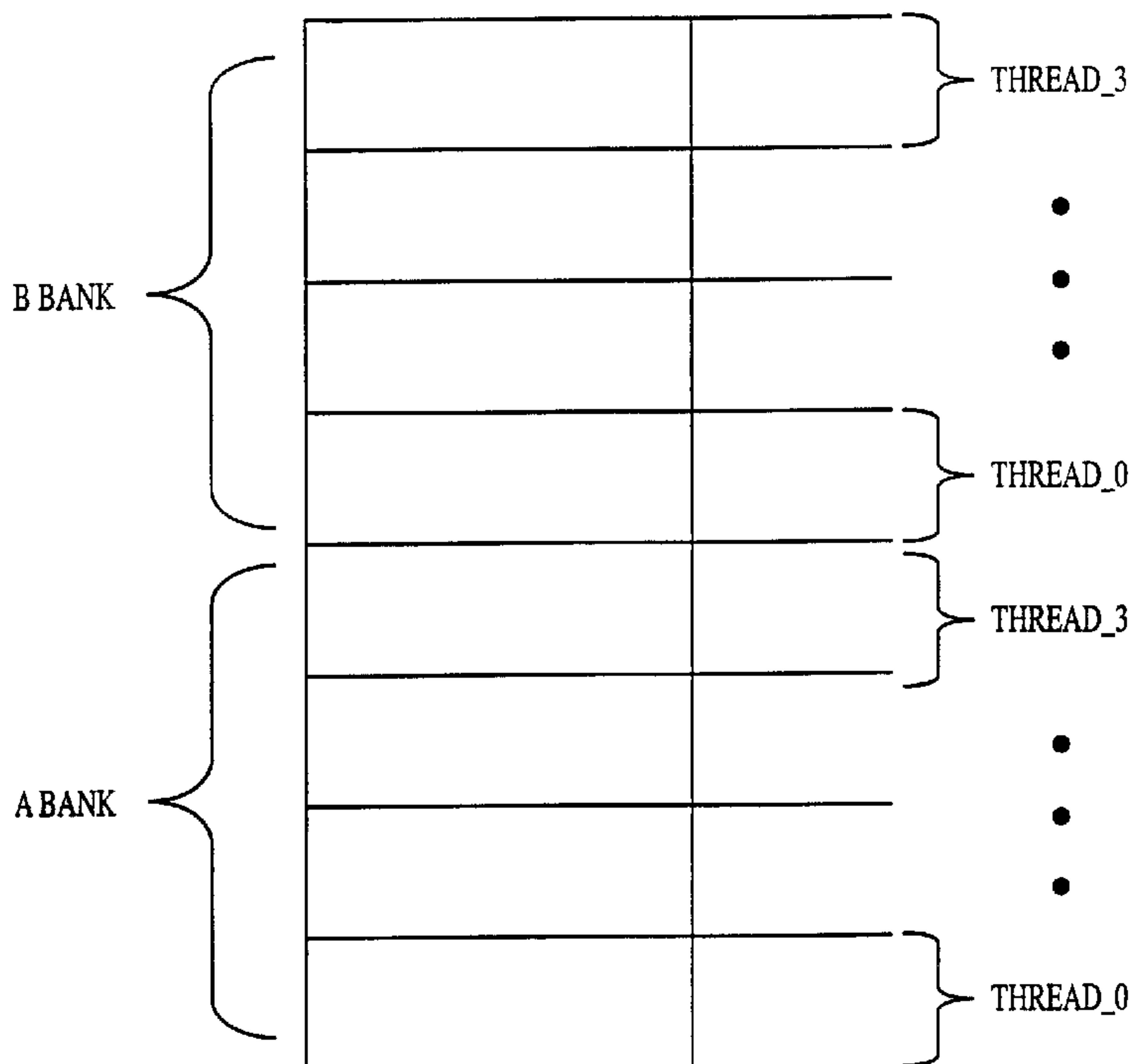
(86) Date de dépôt PCT/PCT Filing Date: 2000/08/31
(87) Date publication PCT/PCT Publication Date: 2001/03/08
(85) Entrée phase nationale/National Entry: 2002/02/28
(86) N° demande PCT/PCT Application No.: US 2000/023993
(87) N° publication PCT/PCT Publication No.: 2001/016702
(30) Priorité/Priority: 1999/09/01 (60/151,961) US

(51) Cl.Int.^{7}/Int.Cl.^{7} G06F 9/00, G06F 12/00
(71) Demandeur/Applicant:
INTEL CORPORATION, US

(72) Inventeurs/Inventors:
WOLRICH, GILBERT, US;
ADILETTA, MATTHEW J., US;
WHEELER, WILLIAM, US;
BERNSTEIN, DEBRA, US;
HOOPER, DONALD, US

(74) Agent: SMART & BIGGAR

(54) Titre : ENSEMBLE DE REGISTRES UTILISE DANS UNE ARCHITECTURE DE PROCESSEURS MULTIFILIERE PARALLELES
(54) Title: REGISTER SET USED IN MULTITHREADED PARALLEL PROCESSOR ARCHITECTURE



(57) Abrégé/Abstract:
A parallel hardware-based multithreaded processor is described. The processor includes a general purpose processor that coordinates system functions and a plurality of microengines that support multiple hardware threads or contexts (THREAD_3 ... THREAD_0). The processor maintains execution threads (THREAD_3 ... THREAD_0). The execution threads (THREAD_3 ... THREAD_0) access a register set organized into a plurality of relatively addressable windows of registers that are relatively addressable per thread (THREAD_3 ... THREAD_0).

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16702 A1

(51) International Patent Classification[7]: G06F 9/00, 12/00

(21) International Application Number: PCT/US00/23993

(22) International Filing Date: 31 August 2000 (31.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/151,961 1 September 1999 (01.09.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/151,961 (CIP)
Filed on 1 September 1999 (01.09.1999)

(71) Applicant (*for all designated States except US*): **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*): **WOLRICH, Gilbert** [US/US]; 4 Cider Mill Road, Framingham, MA 01701 (US). **ADILETTA, Matthew, J.** [US/US]; 20 Monticello Drive, Worcester, MA 01603 (US). **WHEELER, William** [US/US]; 745 School Street, Webster, MA 01570 (US). **BERNSTEIN, Debra** [US/US]; 38 Helen Street, Waltham, MA 02452 (US). **HOOPER, Donald** [US/US]; 19 Main Circle, Shrewsbury, MA 01545 (US).

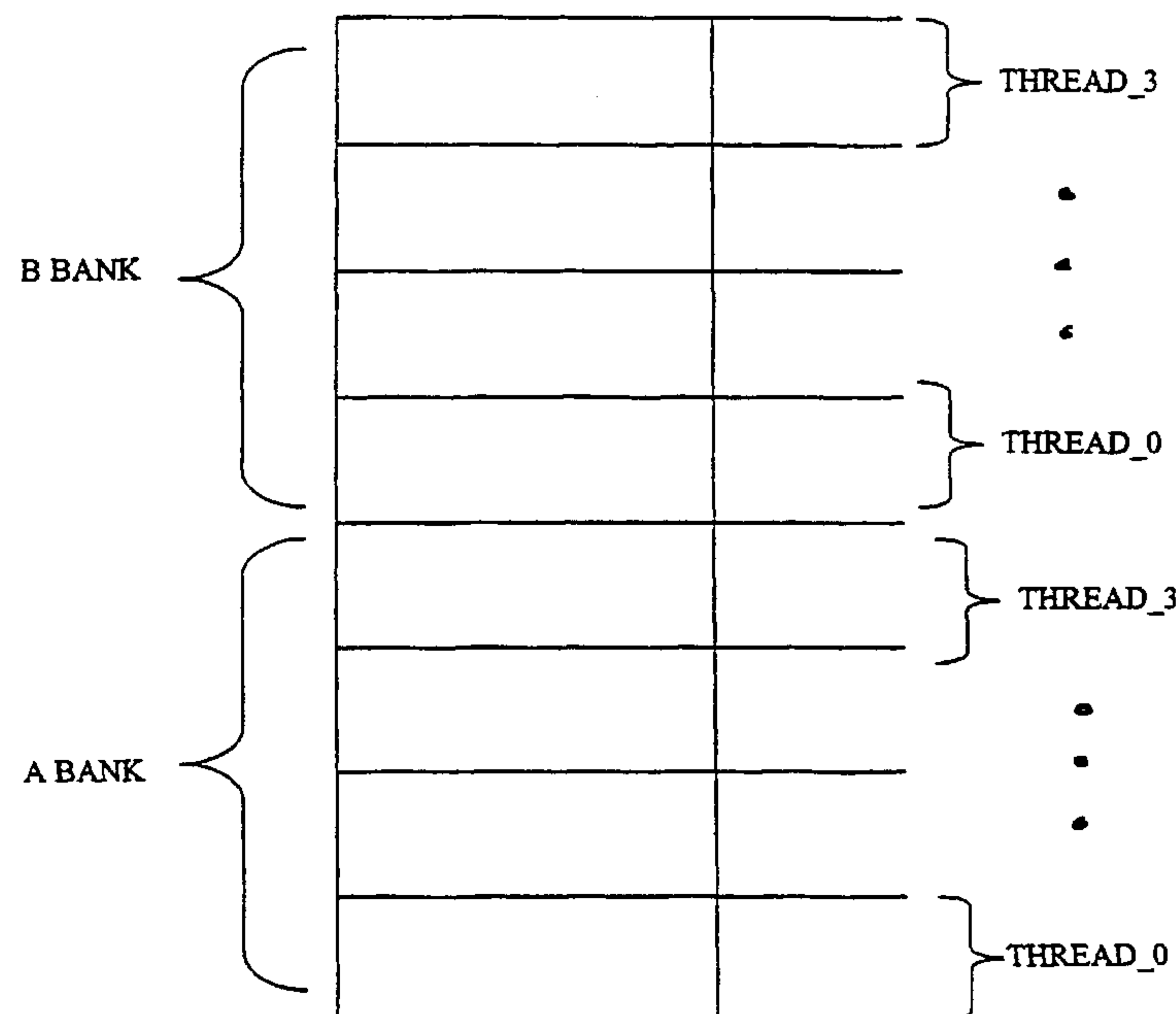
(74) Agents: **MALONEY, Denis, G.**; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 et al. (US).

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: REGISTER SET USED IN MULTITHREADED PARALLEL PROCESSOR ARCHITECTURE



(57) Abstract: A parallel hardware-based multithreaded processor is described. The processor includes a general purpose processor that coordinates system functions and a plurality of microengines that support multiple hardware threads or contexts (THREAD_3 ... THREAD_0). The processor maintains execution threads (THREAD_3 ... THREAD_0). The execution threads (THREAD_3 ... THREAD_0) access a register set organized into a plurality of relatively addressable windows of registers that are relatively addressable per thread (THREAD_3 ... THREAD_0).

WO 01/16702 A1

REGISTER SET USED IN MULTITHREADED PARALLEL PROCESSOR ARCHITECTURE

BACKGROUND

5 This invention relates to computer processors.

Parallel processing is an efficient form of information processing of concurrent events in a computing process. Parallel processing demands concurrent execution of many programs in a computer, in contrast to sequential processing. In the context of a parallel processor, parallelism involves doing more than one thing at the 10 same time. Unlike a serial paradigm where all tasks are performed sequentially at a single station or a pipelined machine where tasks are performed at specialized stations, with parallel processing, a plurality of stations are provided with each capable of performing all tasks. That is, in general all or a plurality of the stations work simultaneously and independently on the same or common elements of a problem. 15 Certain problems are suitable for solution by applying parallel processing.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a communication system employing a hardware-based multithreaded processor.

20 FIG. 2 is a detailed block diagram of the hardware-based multithreaded processor of FIG. 1.

FIG. 3 is a block diagram of a microengine functional unit employed in the hardware-based multithreaded processor of FIGS. 1 and 2.

FIG. 4 is a block diagram of a pipeline in the microengine of FIG. 3.

25 FIG. 5 is a block diagram showing general purpose register address arrangement.

DESCRIPTION

Referring to FIG. 1, a communication system 10 includes a parallel, hardware-based multithreaded processor 12. The hardware-based multithreaded 30 processor 12 is coupled to a bus such as a PCI bus 14, a memory system 16 and a second bus 18. The system 10 is especially useful for tasks that can be broken into parallel subtasks or functions. Specifically hardware-based multithreaded processor 12 is useful for tasks that are bandwidth oriented rather than latency oriented. The hardware-based

multithreaded processor 12 has multiple microengines 22 each with multiple hardware controlled threads that can be simultaneously active and independently work on a task.

The hardware-based multithreaded processor 12 also includes a central controller 20 that assists in loading microcode control for other resources of the hardware-based multithreaded processor 12 and performs other general purpose computer type functions such as handling protocols, exceptions, extra support for packet processing where the microengines pass the packets off for more detailed processing such as in boundary conditions. In one embodiment, the processor 20 is a Strong Arm® (Arm is a trademark of ARM Limited, United Kingdom) based architecture. The general purpose microprocessor 20 has an operating system. Through the operating system the processor 20 can call functions to operate on microengines 22a-22f. The processor 20 can use any supported operating system preferably a real time operating system. For the core processor implemented as a Strong Arm architecture, operating systems such as, MicrosoftNT® real-time, VXWorks and □CUS, a freeware operating system available over the Internet, can be used.

The hardware-based multithreaded processor 12 also includes a plurality of function microengines 22a-22f. Functional microengines (microengines) 22a-22f each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at any one time.

In one embodiment, there are six microengines 22a-22f as shown. Each microengines 22a-22f has capabilities for processing four hardware threads. The six microengines 22a-22f operate with shared resources including memory system 16 and bus interfaces 24 and 28. The memory system 16 includes a Synchronous Dynamic Random Access Memory (SDRAM) controller 26a and a Static Random Access Memory (SRAM) controller 26b. SDRAM memory 16a and SDRAM controller 26a are typically used for processing large volumes of data, e.g., processing of network payloads from network packets. The SRAM controller 26b and SRAM memory 16b are used in a networking implementation for low latency, fast access tasks, e.g., accessing look-up tables, memory for the core processor 20, and so forth.

The six microengines 22a-22f access either the SDRAM 16a or SRAM 16b based on characteristics of the data. Thus, low latency, low bandwidth data is stored

in and fetched from SRAM, whereas higher bandwidth data for which latency is not as important, is stored in and fetched from SDRAM. The microengines 22a-22f can execute memory reference instructions to either the SDRAM controller 26a or SRAM controller 16b.

5 Advantages of hardware multithreading can be explained by SRAM or SDRAM memory accesses. As an example, an SRAM access requested by a Thread_0, from a microengine will cause the SRAM controller 26b to initiate an access to the SRAM memory 16b. The SRAM controller controls arbitration for the SRAM bus, accesses the SRAM 16b, fetches the data from the SRAM 16b, and returns data to a
10 requesting microengine 22a-22b. During an SRAM access, if the microengine e.g., 22a had only a single thread that could operate, that microengine would be dormant until data was returned from the SRAM. By employing hardware context swapping within each of the microengines 22a-22f, the hardware context swapping enables other contexts with unique program counters to execute in that same microengine. Thus, another thread e.g.,
15 Thread_1 can function while the first thread, e.g., Thread_0, is awaiting the read data to return. During execution, Thread_1 may access the SDRAM memory 16a. While Thread_1 operates on the SDRAM unit, and Thread_0 is operating on the SRAM unit, a new thread, e.g., Thread_2 can now operate in the microengine 22a. Thread_2 can operate for a certain amount of time until it needs to access memory or perform some
20 other long latency operation, such as making an access to a bus interface. Therefore, simultaneously, the processor 12 can have a bus operation, SRAM operation and SDRAM operation all being completed or operated upon by one microengine 22a and have one more thread available to process more work in the data path.

The hardware context swapping also synchronizes completion of tasks.
25 For example, two threads could hit the same shared resource e.g., SRAM. Each one of these separate functional units, e.g., the FBUS interface 28, the SRAM controller 26a, and the SDRAM controller 26b, when they complete a requested task from one of the microengine thread contexts reports back a flag signaling completion of an operation. When the flag is received by the microengine, the microengine can determine which
30 thread to turn on.

One example of an application for the hardware-based multithreaded processor 12 is as a network processor. As a network processor, the hardware-based multithreaded processor 12 interfaces to network devices such as a media access

controller device e.g., a 10/100BaseT Octal MAC 13a or a Gigabit Ethernet device 13b. In general, as a network processor, the hardware-based multithreaded processor 12 can interface to any type of communication device or interface that receives/sends large amounts of data. Communication system 10 functioning in a networking application
5 could receive a plurality of network packets from the devices 13a, 13b and process those packets in a parallel manner. With the hardware-based multithreaded processor 12, each network packet can be independently processed.

Another example for use of processor 12 is a print engine for a postscript processor or as a processor for a storage subsystem, i.e., RAID disk storage. A further
10 use is as a matching engine. In the securities industry for example, the advent of electronic trading requires the use of electronic matching engines to match orders between buyers and sellers. These and other parallel types of tasks can be accomplished on the system 10.

The processor 12 includes a bus interface 28 that couples the processor to
15 the second bus 18. Bus interface 28 in one embodiment couples the processor 12 to the so-called FBUS 18 (FIFO bus). The FBUS interface 28 is responsible for controlling and interfacing the processor 12 to the FBUS 18. The FBUS 18 is a 64-bit wide FIFO bus, used to interface to Media Access Controller (MAC) devices.

The processor 12 includes a second interface e.g., a PCI bus interface 24
20 that couples other system components that reside on the PCI 14 bus to the processor 12. The PCI bus interface 24, provides a high speed data path 24a to memory 16 e.g., the SDRAM memory 16a. Through that path data can be moved quickly from the SDRAM 16a through the PCI bus 14, via direct memory access (DMA) transfers. The hardware based multithreaded processor 12 supports image transfers. The hardware based
25 multithreaded processor 12 can employ a plurality of DMA channels so if one target of a DMA transfer is busy, another one of the DMA channels can take over the PCI bus to deliver information to another target to maintain high processor 12 efficiency. Additionally, the PCI bus interface 24 supports target and master operations. Target operations are operations where slave devices on bus 14 access SDRAMs through reads
30 and writes that are serviced as a slave to target operation. In master operations, the processor core 20 sends data directly to or receives data directly from the PCI interface 24.

Each of the functional units are coupled to one or more internal buses. As described below, the internal buses are dual, 32 bit buses (i.e., one bus for read and one for write). The hardware-based multithreaded processor 12 also is constructed such that the sum of the bandwidths of the internal buses in the processor 12 exceed the bandwidth
5 of external buses coupled to the processor 12. The processor 12 includes an internal core processor bus 32, e.g., an ASB bus (Advanced System Bus) that couples the processor core 20 to the memory controller 26a, 26c and to an ASB translator 30 described below. The ASB bus is a subset of the so called AMBA bus that is used with the Strong Arm processor core. The processor 12 also includes a private bus 34 that couples the
10 microengine units to SRAM controller 26b, ASB translator 30 and FBUS interface 28. A memory bus 38 couples the memory controller 26a, 26b to the bus interfaces 24 and 28 and memory system 16 including flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, each of the microengines 22a-22f includes an arbiter that examines flags to determine the available threads to be operated upon. Any thread
15 from any of the microengines 22a-22f can access the SDRAM controller 26a, SDRAM controller 26b or FBUS interface 28. The memory controllers 26a and 26b each include a plurality of queues to store outstanding memory reference requests. The queues either maintain order of memory references or arrange memory references to optimize memory bandwidth. For example, if a thread_0 has no dependencies or relationship to a thread_1,
20 there is no reason that thread 1 and 0 cannot complete their memory references to the SRAM unit out of order. The microengines 22a-22f issue memory reference requests to the memory controllers 26a and 26b. The microengines 22a-22f flood the memory subsystems 26a and 26b with enough memory reference operations such that the memory subsystems 26a and 26b become the bottleneck for processor 12 operation.

25 If the memory subsystem 16 is flooded with memory requests that are independent in nature, the processor 12 can perform memory reference sorting. Memory reference sorting improves achievable memory bandwidth. Memory reference sorting, as described below, reduces dead time or a bubble that occurs with accesses to SRAM. With memory references to SRAM, switching current direction on signal lines between
30 reads and writes produces a bubble or a dead time waiting for current to settle on conductors coupling the SRAM 16b to the SRAM controller 26b.

That is, the drivers that drive current on the bus need to settle out prior to changing states. Thus, repetitive cycles of a read followed by a write can degrade peak

bandwidth. Memory reference sorting allows the processor 12 to organize references to memory such that long strings of reads can be followed by long strings of writes. This can be used to minimize dead time in the pipeline to effectively achieve closer to maximum available bandwidth. Reference sorting helps maintain parallel hardware
5 context threads. On the SDRAM, reference sorting allows hiding of pre-charges from one bank to another bank. Specifically, if the memory system 16b is organized into an odd bank and an even bank, while the processor is operating on the odd bank, the memory controller can start precharging the even bank. Precharging is possible if memory references alternate between odd and even banks. By ordering memory references to
10 alternate accesses to opposite banks, the processor 12 improves SDRAM bandwidth. Additionally, other optimizations can be used. For example, merging optimizations where operations that can be merged, are merged prior to memory access, open page optimizations where by examining addresses an opened page of memory is not reopened, chaining, as will be described below, and refreshing mechanisms, can be employed.
15 The FBUS interface 28 supports Transmit and Receive flags for each port that a MAC device supports, along with an Interrupt flag indicating when service is warranted. The FBUS interface 28 also includes a controller 28a that performs header processing of incoming packets from the FBUS 18. The controller 28a extracts the packet headers and performs a microprogrammable source/destination/protocol hashed
20 lookup (used for address smoothing) in SRAM. If the hash does not successfully resolve, the packet header is sent to the processor core 20 for additional processing. The FBUS interface 28 supports the following internal data transactions:

	FBUS unit	(Shared bus SRAM)	to/from microengine.
25	FBUS unit	(via private bus)	writes from SDRAM Unit.
	FBUS unit	(via Mbus)	Reads to SDRAM.

The FBUS 18 is a standard industry bus and includes a data bus, e.g., 64 bits wide and sideband control for address and read/write control. The FBUS interface 28
30 provides the ability to input large amounts of data using a series of input and output FIFO's 29a-29b. From the FIFOs 29a-29b, the microengines 22a-22f fetch data from or command the SDRAM controller 26a to move data from a receive FIFO in which data has come from a device on bus 18, into the FBUS interface 28. The data can be sent

through memory controller 26a to SDRAM memory 16a, via a direct memory access. Similarly, the microengines can move data from the SDRAM 26a to interface 28, out to FBUS 18, via the FBUS interface 28.

Data functions are distributed amongst the microengines. Connectivity to
5 the SRAM 26a, SDRAM 26b and FBUS 28 is via command requests. A command request can be a memory request or a FBUS request. For example, a command request can move data from a register located in a microengine 22a to a shared resource, e.g., an SDRAM location, SRAM location, flash memory or some MAC address. The commands are sent out to each of the functional units and the shared resources. However,
10 the shared resources do not need to maintain local buffering of the data. Rather, the shared resources access distributed data located inside of the microengines. This enables microengines 22a-22f, to have local access to data rather than arbitrating for access on a bus and risk contention for the bus. With this feature, there is a 0 cycle stall for waiting for data internal to the microengines 22a-22f.

15 The data buses, e.g., ASB bus 30, SRAM bus 34 and SDRAM bus 38 coupling these shared resources, e.g., memory controllers 26a and 26b are of sufficient bandwidth such that there are no internal bottlenecks. Thus, in order to avoid bottlenecks, the processor 12 has an bandwidth requirement where each of the functional units is provided with at least twice the maximum bandwidth of the internal buses. As an
20 example, the SDRAM can run a 64 bit wide bus at 83 MHz. The SRAM data bus could have separate read and write buses, e.g., could be a read bus of 32 bits wide running at 166 MHz and a write bus of 32 bits wide at 166 MHz. That is, in essence, 64 bits running at 166 MHz which is effectively twice the bandwidth of the SDRAM.

The core processor 20 also can access the shared resources. The core
25 processor 20 has a direct communication to the SDRAM controller 26a to the bus interface 24 and to SRAM controller 26b via bus 32. However, to access the microengines 22a-22f and transfer registers located at any of the microengines 22a-22f, the core processor 20 access the microengines 22a-22f via the ASB Translator 30 over bus 34. The ASB translator 30 can physically reside in the FBUS interface 28, but
30 logically is distinct. The ASB Translator 30 performs an address translation between FBUS microengine transfer register locations and core processor addresses (i.e., ASB bus) so that the core processor 20 can access registers belonging to the microengines 22a-22c.

Although microengines 22 can use the register set to exchange data as described below, a scratchpad memory 27 is also provided to permit microengines to write data out to the memory for other microengines to read. The scratchpad 27 is coupled to bus 34.

5 The processor core 20 includes a RISC core 50 implemented in a five stage pipeline performing a single cycle shift of one operand or two operands in a single cycle, provides multiplication support and 32 bit barrel shift support. This RISC core 50 is a standard Strong Arm® architecture but it is implemented with a five stage pipeline for performance reasons. The processor core 20 also includes a 16 kilobyte instruction 10 cache 52, an 8 kilobyte data cache 54 and a prefetch stream buffer 56. The core processor 20 performs arithmetic operations in parallel with memory writes and instruction fetches. The core processor 20 interfaces with other functional units via the ARM defined ASB bus. The ASB bus is a 32-bit bi-directional bus 32.

15 Microengines:

Referring to FIG. 3, an exemplary one of the microengines 22a-22f, e.g., microengine 22f is shown. The microengine includes a control store 70 which, in one implementation, includes a RAM of here 1,024 words of 32 bit. The RAM stores a microprogram. The microprogram is loadable by the core processor 20. The 20 microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-72d. The four micro program counters 72a-72d are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ_#_EVENT_RESPONSE; FBI_EVENT_RESPONSE; SRAM 25 _EVENT_RESPONSE; SDRAM _EVENT_RESPONSE; and ASB _EVENT_RESPONSE) from each one of the shared resources, e.g., SRAM 26a, SDRAM 26b, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a thread has completed and signaled completion, the thread 30 needs to wait for that completion signal, and if the thread is enabled to operate, then the thread is placed on an available thread list (not shown). The microengine 22f can have a maximum of e.g., 4 threads available.

In addition to event signals that are local to an executing thread, the microengines 22 employ signaling states that are global. With signaling states, an executing thread can broadcast a signal state to all microengines 22. Receive Request Available signal, Any and all threads in the microengines can branch on these signaling states. These signaling states can be used to determine availability of a resource or whether a resource is due for servicing.

The context event logic 74 has arbitration for the four (4) threads. In one embodiment, the arbitration is a round robin mechanism. Other techniques could be used including priority queuing or weighted fair queuing. The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions. The registers set 76b has a relatively large number of general purpose registers. As will be described in FIG. 6, in this implementation there are 64 general purpose registers in a first bank, Bank A and 64 in a second bank, Bank B. The general purpose registers are windowed as will be described so that they are relatively and absolutely addressable.

The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80. These registers are also windowed so that they are relatively and absolutely addressable. Write transfer register stack 78 is where write data to a resource is located. Similarly, read register stack 80 is for return data from a shared resource. Subsequent to or concurrent with data arrival, an event signal from the respective shared resource e.g., the SRAM controller 26a, SDRAM controller 26b or core processor 20 will be provided to context event arbiter 74 which will then alert the thread that the data is available or has been sent. Both transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path. In one implementation, the read transfer register has 64 registers and the write transfer register has 64 registers.

Referring to FIG. 4, the microengine datapath maintains a 5-stage micro-pipeline 82. This pipeline includes lookup of microinstruction words 82a, formation of the register file addresses 82b, read of operands from register file 82c, ALU, shift or compare operations 82d, and write-back of results to registers 82e. By providing a write-back data bypass into the ALU/shifter units, and by assuming the registers are implemented as a register file (rather than a RAM), the microengine can perform a simultaneous register file read and write, which completely hides the write operation.

The SDRAM interface 26a provides a signal back to the requesting microengine on reads that indicates whether a parity error occurred on the read request. The microengine microcode is responsible for checking the SDRAM read Parity flag when the microengine uses any return data. Upon checking the flag, if it was set, the act of branching on it clears it. The Parity flag is only sent when the SDRAM is enabled for checking, and the SDRAM is parity protected. The microengines and the PCI Unit are the only requestors notified of parity errors. Therefore, if the processor core 20 or FIFO requires parity protection, a microengine assists in the request.

Referring to FIG. 5, the two register address spaces that exist are Locally accessibly registers, and Globally accessible registers accessible by all microengines. The General Purpose Registers (GPRs) are implemented as two separate banks (A bank and B bank) whose addresses are interleaved on a word-by-word basis such that A bank registers have lsb=0, and B bank registers have lsb=1. Each bank is capable of performing a simultaneous read and write to two different words within its bank.

Across banks A and B, the register set 76b is also organized into four windows $76b_0$-$76b_3$ of 32 registers that are relatively addressable per thread. Thus, thread_0 will find its register 0 at 77a (register 0), the thread_1 will find its register_0 at 77b (register 32), thread_2 will find its register_0 at 77c (register 64), and thread_3 at 77d (register 96). Relative addressing is supported so that multiple threads can use the exact same control store and locations but access different windows of register and perform different functions. The uses of register window addressing and bank addressing provide the requisite read bandwidth using only dual ported RAMS in the microengine 22f.

These windowed registers do not have to save data from context switch to context switch so that the normal push and pop of a context swap file or stack is eliminated. Context switching here has a 0 cycle overhead for changing from one context to another. Relative register addressing divides the register banks into windows across the address width of the general purpose register set. Relative addressing allows access any of the windows relative to the starting point of the window. Absolute addressing is also supported in this architecture where any one of the absolute registers may be accessed by any of the threads by providing the exact address of the register.

Addressing of general purpose registers 78 occurs in 2 modes depending on the microword format. The two modes are absolute and relative. In absolute mode, addressing of a register address is directly specified in 7-bit source field (a6-a0 or b6-b0):

```
              7   6   5   4   3   2   1   0
            +---+---+---+---+---+---+---+---+
A GPR:      | a6| 0 | a5| a4| a3| a2| a1| a0|   a6=0
B GPR:      | b6| 1 | b5| b4| b3| b2| b1| b0|   b6=0
SRAM/ASB:   | a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=0
SDRAM:      | a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=1
```

register address directly specified in 8-bit dest field (d7-d0):

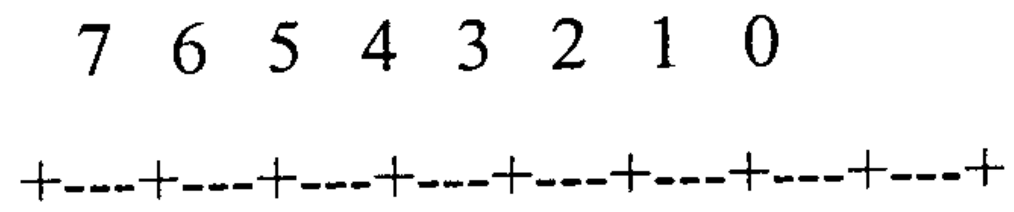
```
              7   6   5   4   3   2   1   0
            +---+---+---+---+---+---+---+---+
A GPR:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=0, d6=0
B GPR:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=0, d6=1
SRAM/ASB:   | d7| d6| d5| d4| d3| d2| d1| d0| d7=1, d6=0, d5=0
SDRAM:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=1, d6=0, d5=1
```

If <a6:a5>=1,1, <b6:b5>=1,1, or <d7:d6>=1,1 then the lower bits are interpreted as a context-relative address field (described below). When a non-relative A or B source address is specified in the A, B absolute field, only the lower half of the SRAM/ASB and SDRAM address spaces can be addressed. Effectively, reading absolute SRAM/SDRAM devices has the effective address space; however, since this restriction does not apply to the dest field, writing the SRAM/SDRAM still uses the full address space.

In relative mode, addresses a specified address is offset within context space as defined by a 5-bit source field (a4-a0 or b4-b0):

```
              7   6   5   4   3   2   1   0
            +---+---+---+---+---+---+---+---+
A GPR:      | a4| 0 |context| a3| a2| a1| a0| a4=0
B GPR:      | b4| 1 |context| b3| b2| b1| b0| b4=0
SRAM/ASB:   |ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=0
SDRAM:      |ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=1
```

or as defined by the 6-bit dest field (d5-d0):



A GPR: | d5| d4|context| d3| d2| d1| d0| d5=0, d4=0

B GPR: | d5| d4|context| d3| d2| d1| d0| d5=0, d4=1

SRAM/ASB:| d5| d4| d3|context| d2| d1| d0| d5=1, d4=0, d3=0

SDRAM: | d5| d4| d3|context| d2| d1| d0| d5=1, d4=0, d3=1

If <d5:d4>=1,1, then the destination address does not address a valid register, thus, no dest operand is written back.

Other embodiments are within the scope of the appended claims.

What is claimed is:

1. A method of maintaining execution threads in a parallel multithreaded processor comprises:

accessing, by an executing thread in the multithreaded processor, a register set organized into a plurality of relatively addressable windows of registers that are
5 relatively addressable per thread.

2. The method of claim 1 wherein multiple threads can use the same control store and relative register locations but access different window banks of registers.

10 3. The method of claim 1 wherein the relative register addressing divides the register banks into windows across the address width of the general purpose register set.

4. The method of claim 1 wherein relative addressing allows access any of the window registers relative to the starting point of a window of registers.

15

5. The method of claim 1 further comprising:

organizing the register set into windows according to the number of threads that execute in the processor.

20 6. The method of claim 1 wherein relative addressing allow the multiple threads to use the same control store and locations while allowing access to different windows of register and perform different functions.

7. The method of claim 1 wherein the window registers are implemented
25 using dual ported random access memories.

8. The method of claim 1 wherein relative addressing allows access to any of the windows of registers relative to the starting point of the window of registers.

30 9. The method of claim 1 wherein the register set is also absolutely addressable where any one of the absolutely addressable registers may be accessed by any of the threads by providing the exact address of the register.

10. The method of claim 9 wherein an absolute address of a register is directly specified in a source field or destination field of an instruction.

11. The method of claim 1 wherein relative addresses are specified in
5 instructions as an address offset within a context execution space as defined by a source field or destination field operand.

12. A hardware based multi-threaded processor comprises:
a processor unit comprising:
10 control logic including context event switching logic, the context switching logic arbitrating access to the microengine for a plurality of executable threads;
an arithmetic logic unit to process data for executing threads; and
a register set that is organized into a plurality of relatively addressable
15 windows of registers that are relatively addressable executable thread.

13. The processor of claim 12 wherein the control logic further comprises:
an instruction decoder; and
program counter units to track executing threads.
20

14. The processor of claim 13 wherein the program counters units are maintained in hardware.

15. The processor of claim 13 wherein the register banks are organized into
25 windows across an address width of the general purpose register set with each window relatively accessible by a corresponding thread.

16. The processor of claim 15 wherein the relative addressing allows access to any of the registers relative to the starting point of a window of registers.
30

17. The processor of claim 15 wherein the number of windows of the register set is according to the number of threads that execute in the processor.

18. The processor of claim 13 wherein relative addressing allow the multiple threads to use the same control store and locations while allowing access to different windows of register and perform different functions.

5 19. The processor of claim 13 wherein the window registers are provided using dual ported random access memories.

20. The processor of claim 12 wherein the processing unit is a microprogrammed processor unit.

10

21. A computer program product residing on a computer readable medium for managing execution of multiple threads in a multithreaded processor comprising instructions causing a processor to:

access, by an executing thread in the multithreaded processor, a register set

15 organized into a plurality of relatively addressable windows of registers that are relatively addressable per thread.

22. The product of claim 21 wherein the register set is also absolutely addressable where any one of the absolutely addressable registers may be accessed by any

20 of the threads by providing the exact address of the register.

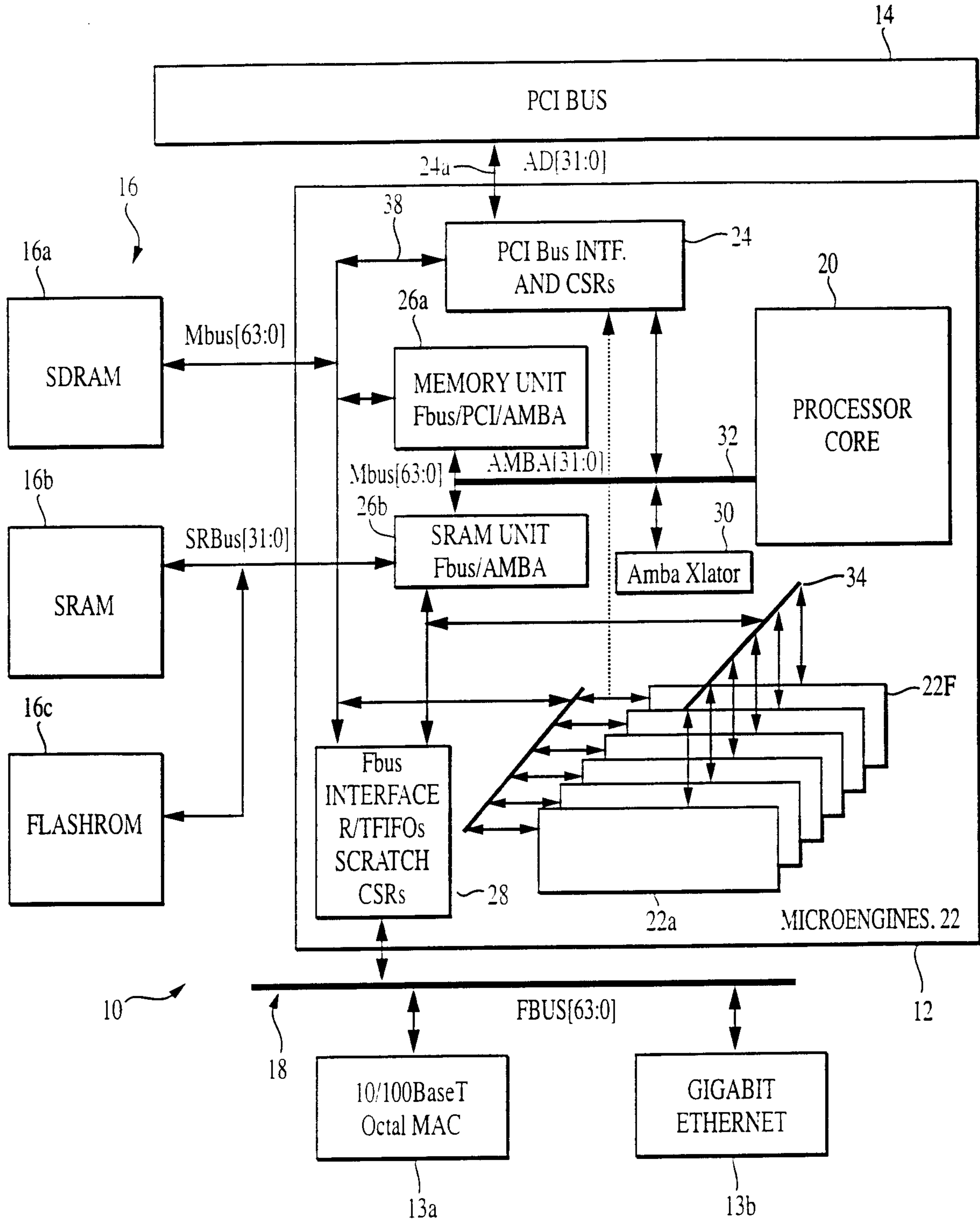


FIG. 1

2/6

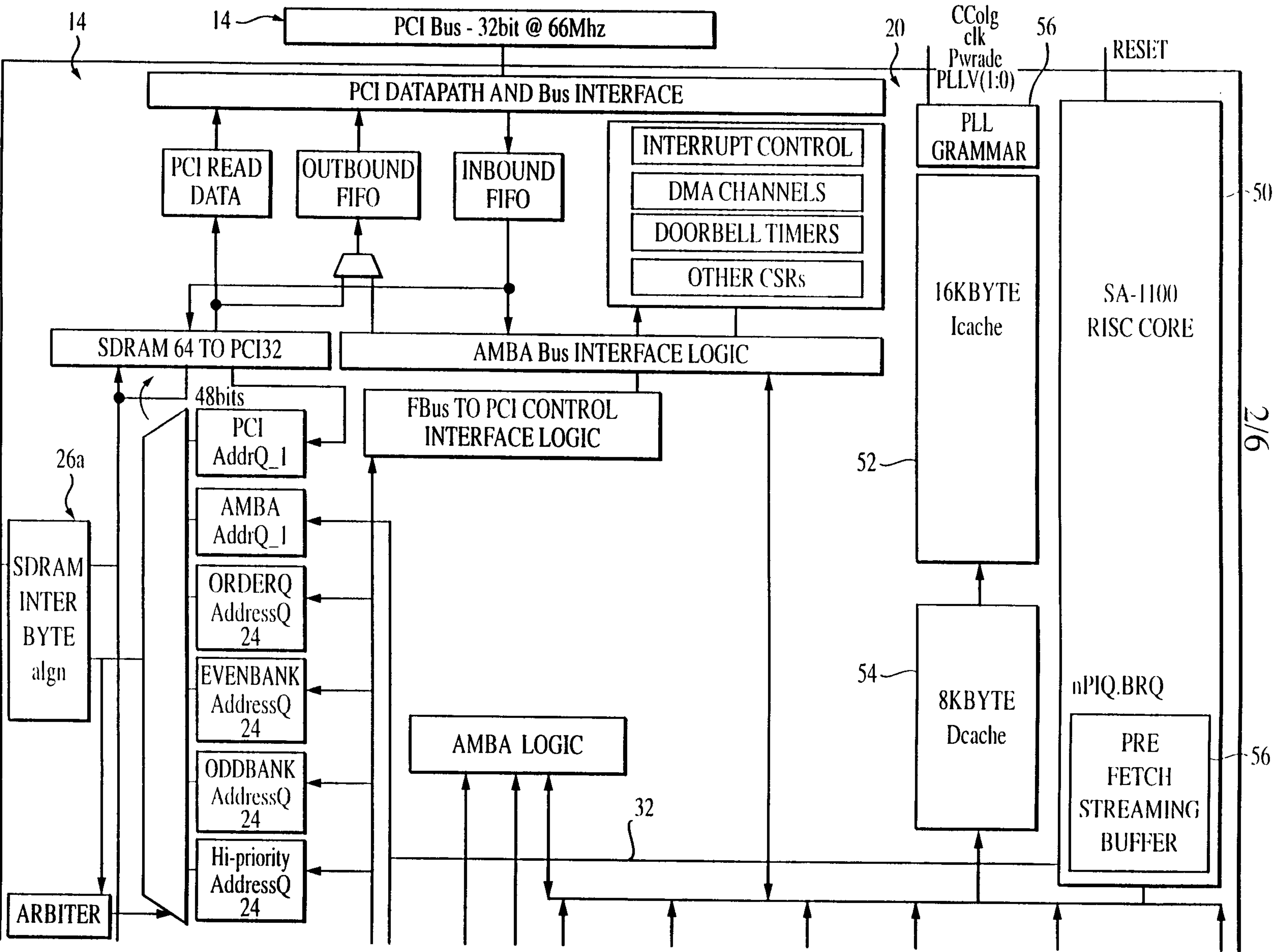


FIG. 2-1

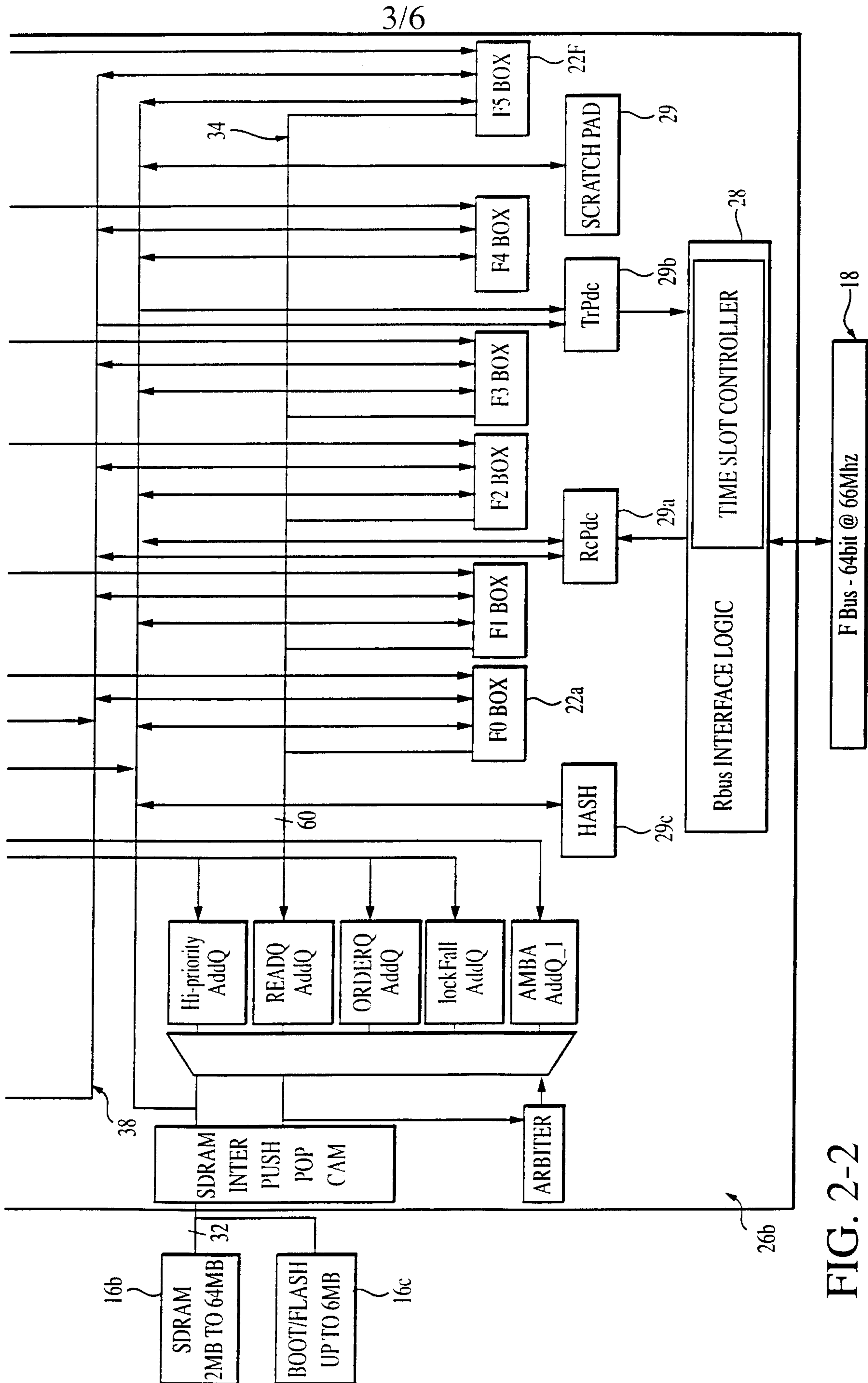


FIG. 2-2

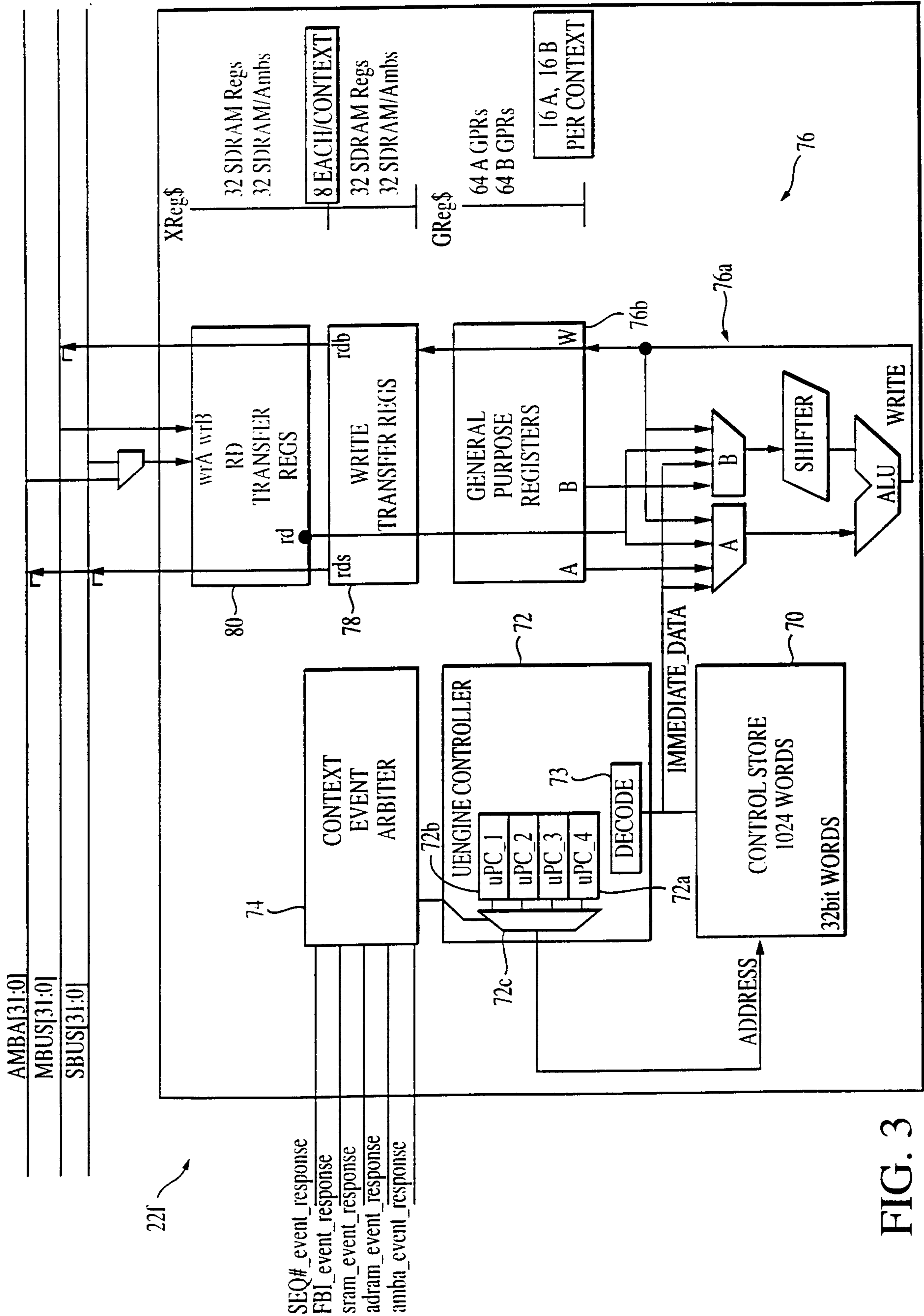


FIG. 3

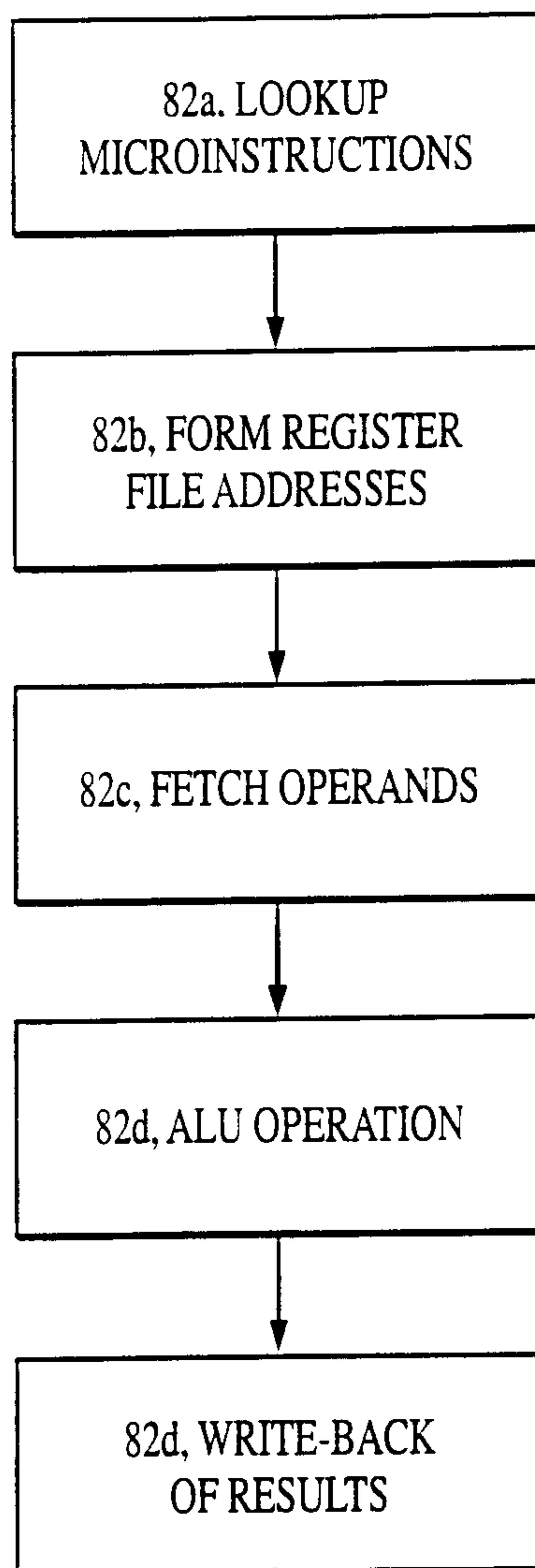


FIG. 4

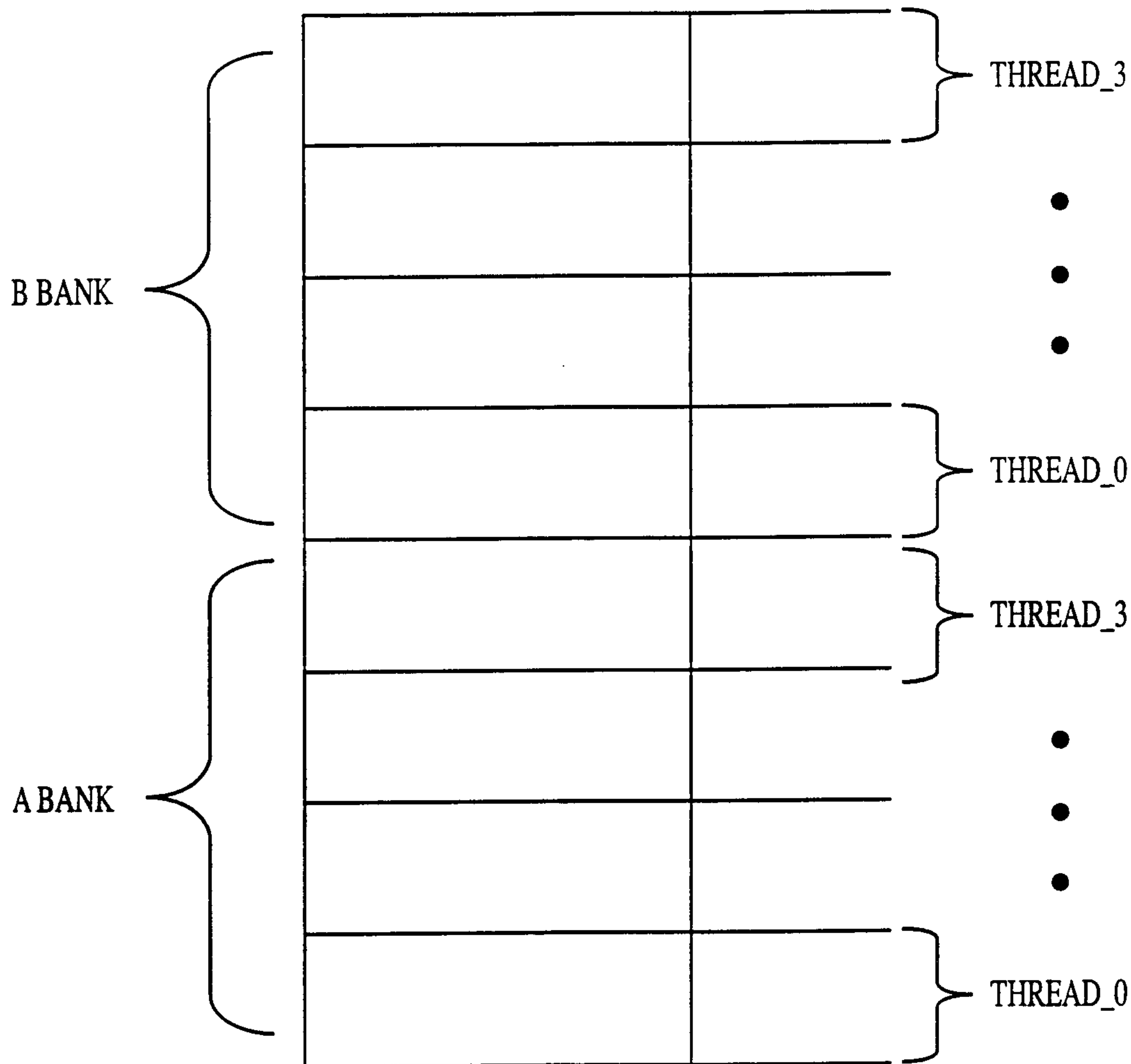
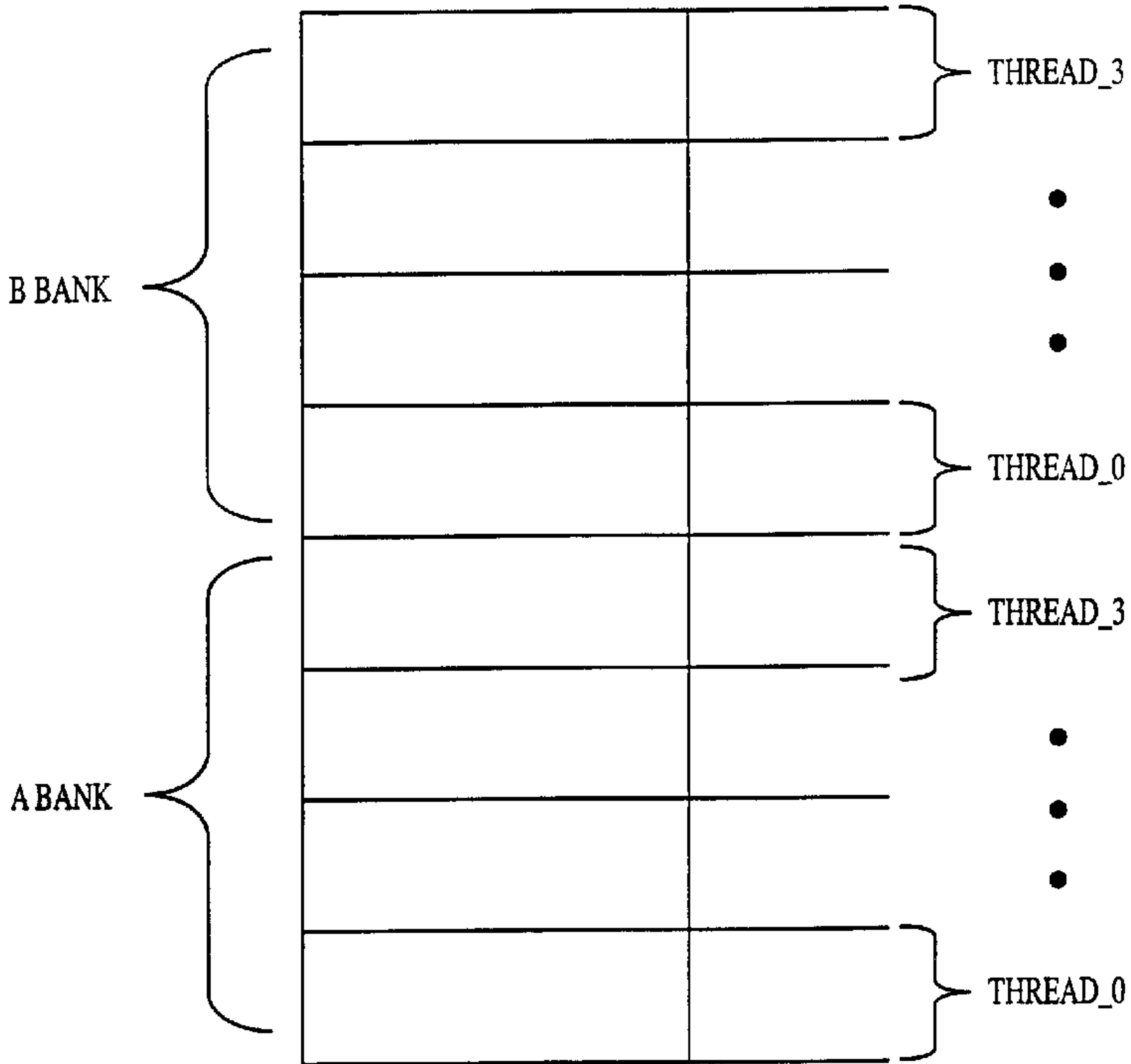


FIG. 5



THREAD_3
B BANK
THREAD_0
THREAD_3
A BANK
THREAD_0