(12) UK Patent (19) GB (11) 2 210 479 (13) B

(54) Title of Invention

Alias address support

(51) INT CL^5; **G06F 12/10**

(21) Application No
8819017.8

(22) Date of filing
10.08.1988

(30) Priority Data

(31) **104635**

(32) **02.10.1987**

(33) **US**

(43) Application published
07.06.1989

(45) Patent published
17.06.1992

(52) Domestic classification
(Edition K)
G4A AMC ANV

(56) Documents cited
None

(58) Field of search

As for published application
2210479 A *viz:*
UK CL(Edition J) **G4A AMC ANV**
INT CL^4 **G06F, G11C**
updated as appropriate

(72) Inventor(s)
William Van Loo
John Watkins
Joseph Moran
William Shannon
Ray Cheng

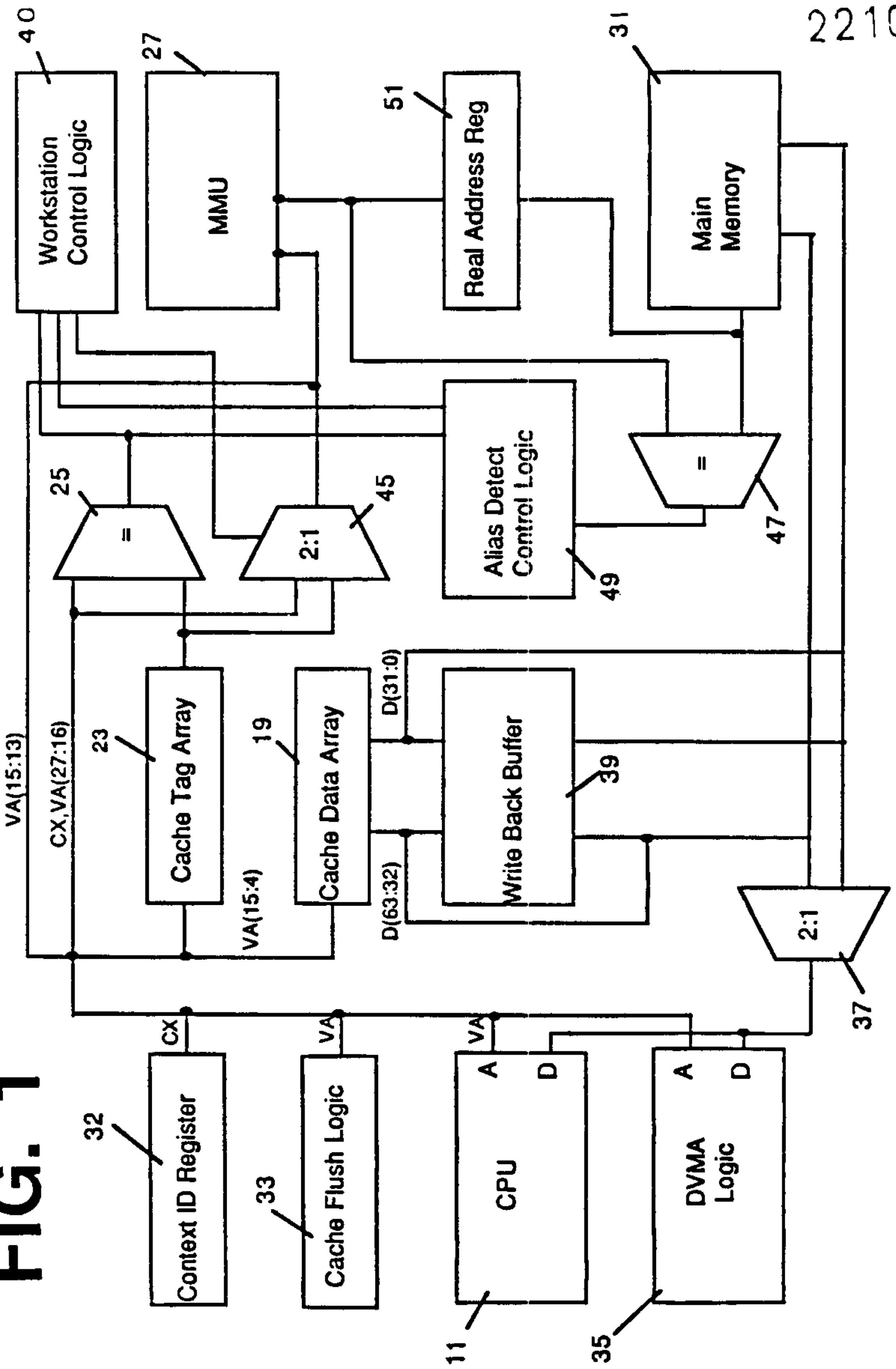
(73) Proprietor(s)
Sun Microsystems Inc

(Incorporated in USA - Delaware)

2550 Garcia Avenue
Mountain View
California 94043
United States of America

(74) Agent and/or
Address for Service
Potts, Kerr & Co
15 Hamilton Square
Birkenhead
Merseyside
L41 6BR
United Kingdom

FIG. 1


Workstation Control Logic
40
MMU
27
Real Address Reg
51
Main Memory
31
Alias Detect Control Logic
49
=
47
=
25
2:1
45
Cache Tag Array
23
Cache Data Array
19
Write Back Buffer
39
2:1
37
VA(15:13)
CX,VA(27:16)
VA(15:4)
D(31:0)
D(63:32)
CX
VA
VA
A
D
Context ID Register
32
Cache Flush Logic
33
CPU
11
DVMA Logic
35

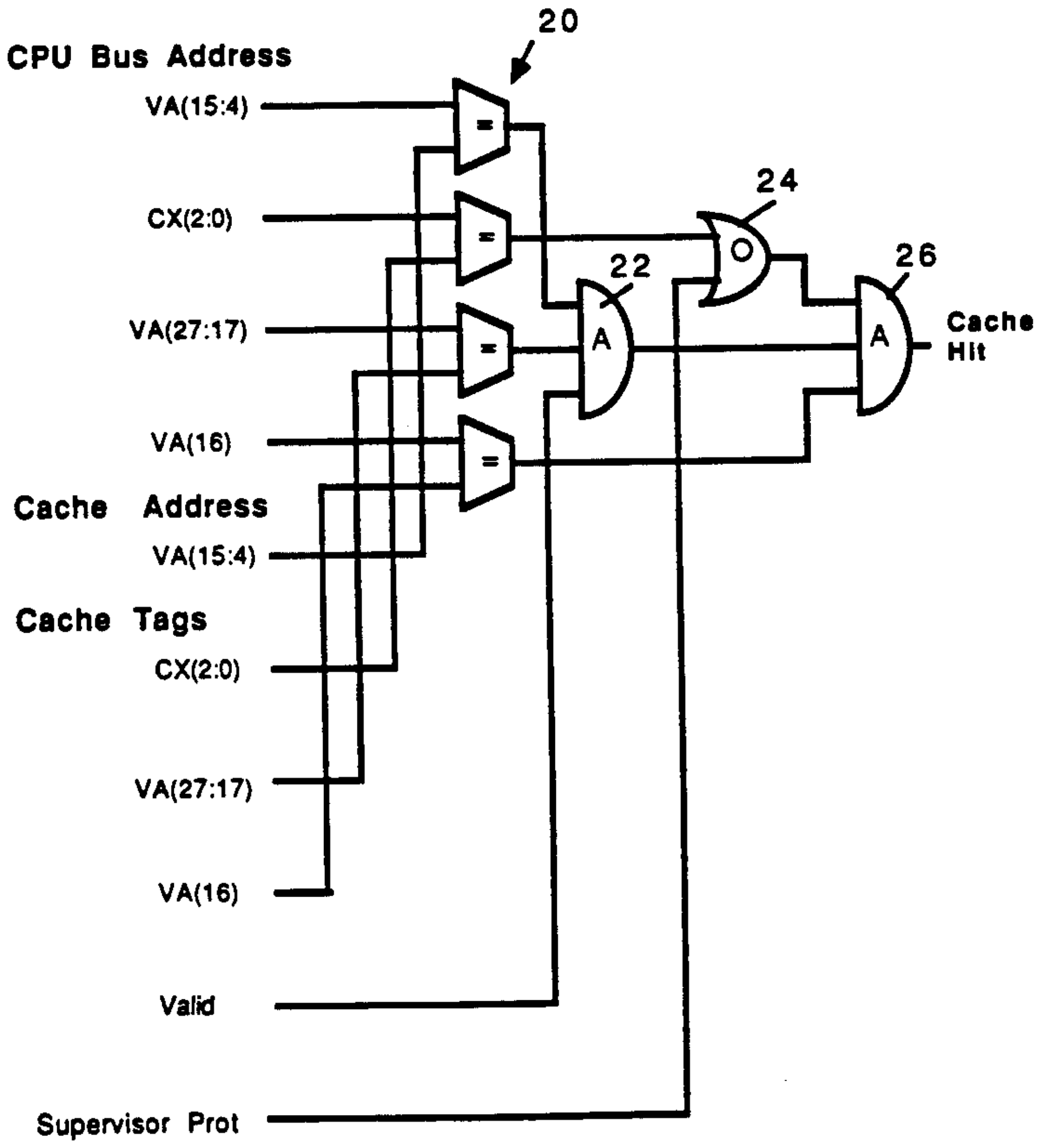

20
CPU Bus Address
VA(15:4)
CX(2:0)
VA(27:17)
VA(16)
Cache Address
VA(15:4)
Cache Tags
CX(2:0)
VA(27:17)
VA(16)
Valid
Supervisor Prot
=
=
=
=
22
A
24
O
26
A
Cache Hit

Fig. 2a

Virtual Address Cache:
Cache and MMU Protection Violations

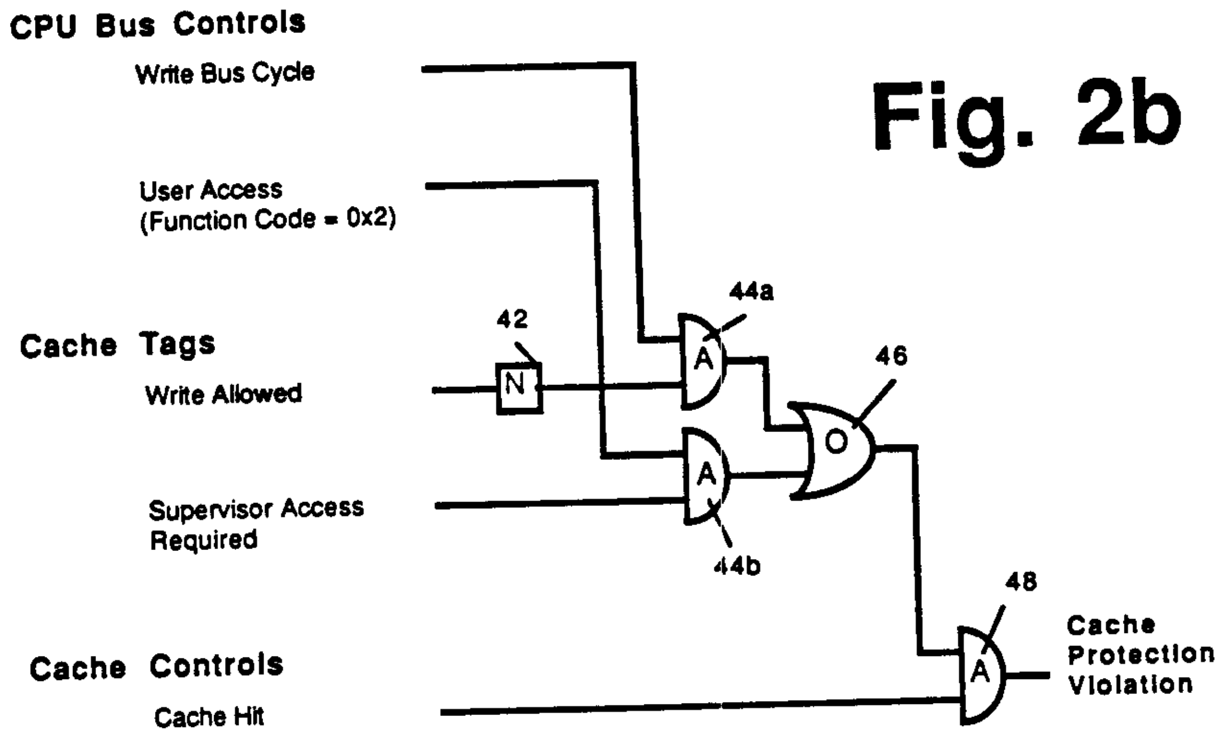


Fig. 2b

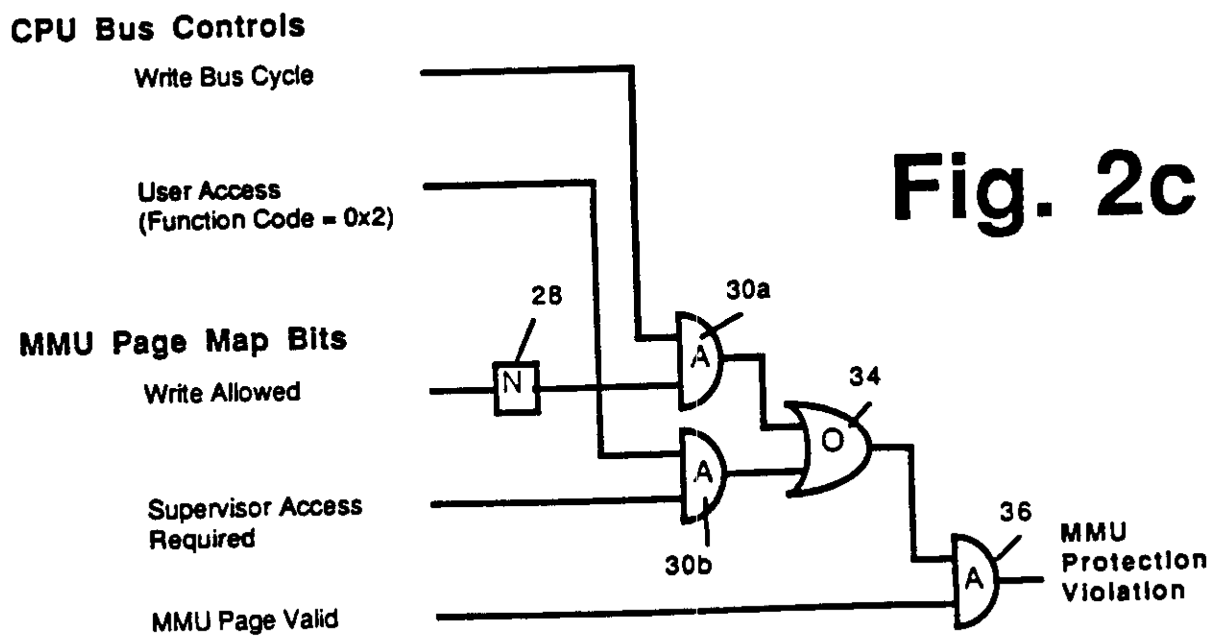


Fig. 2c

Alias Detect Address Path

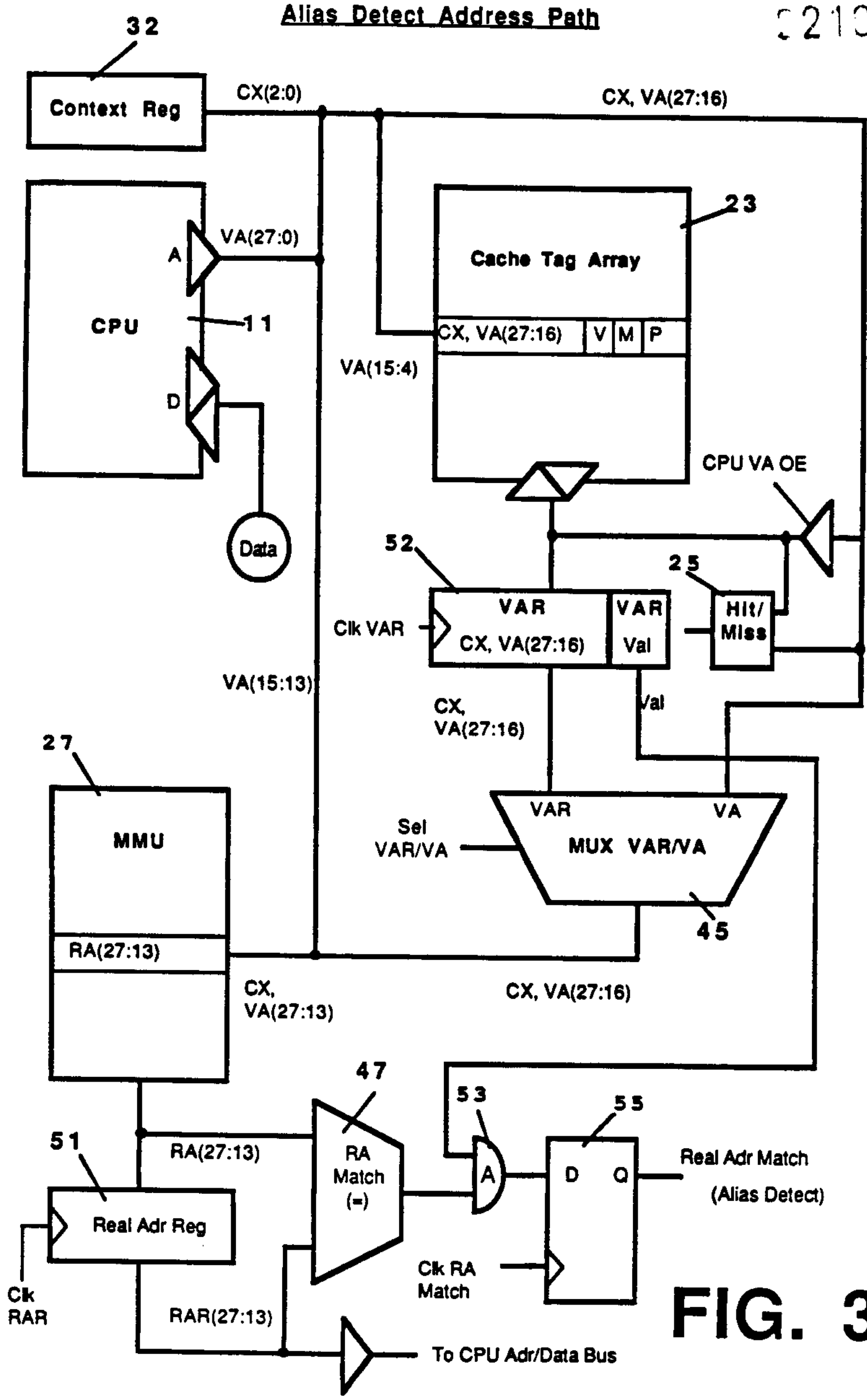


FIG. 3

Alias Detect: Address State Machine

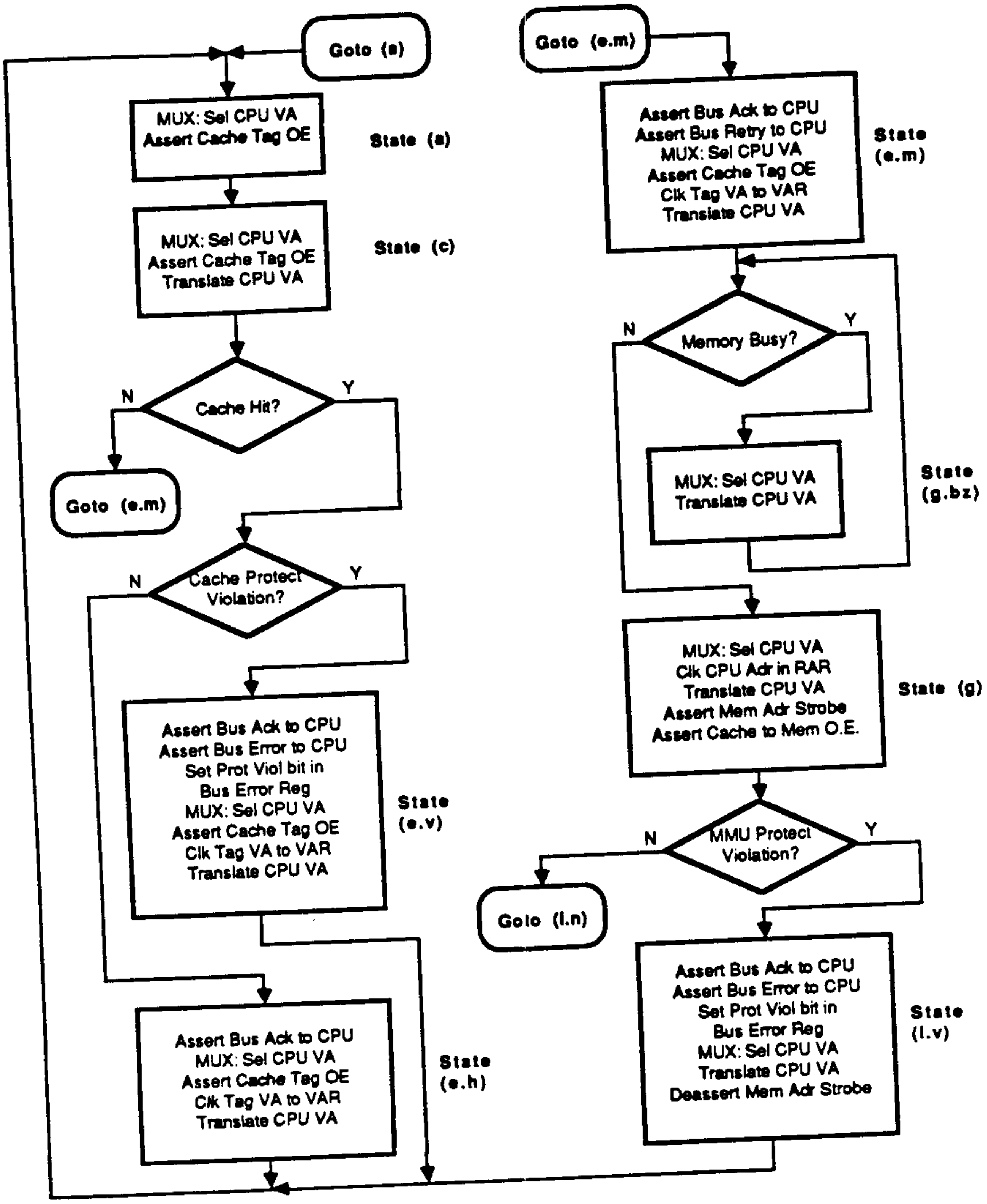


Fig. 4a

Alias Detect: Address State Machine, con't

2210479

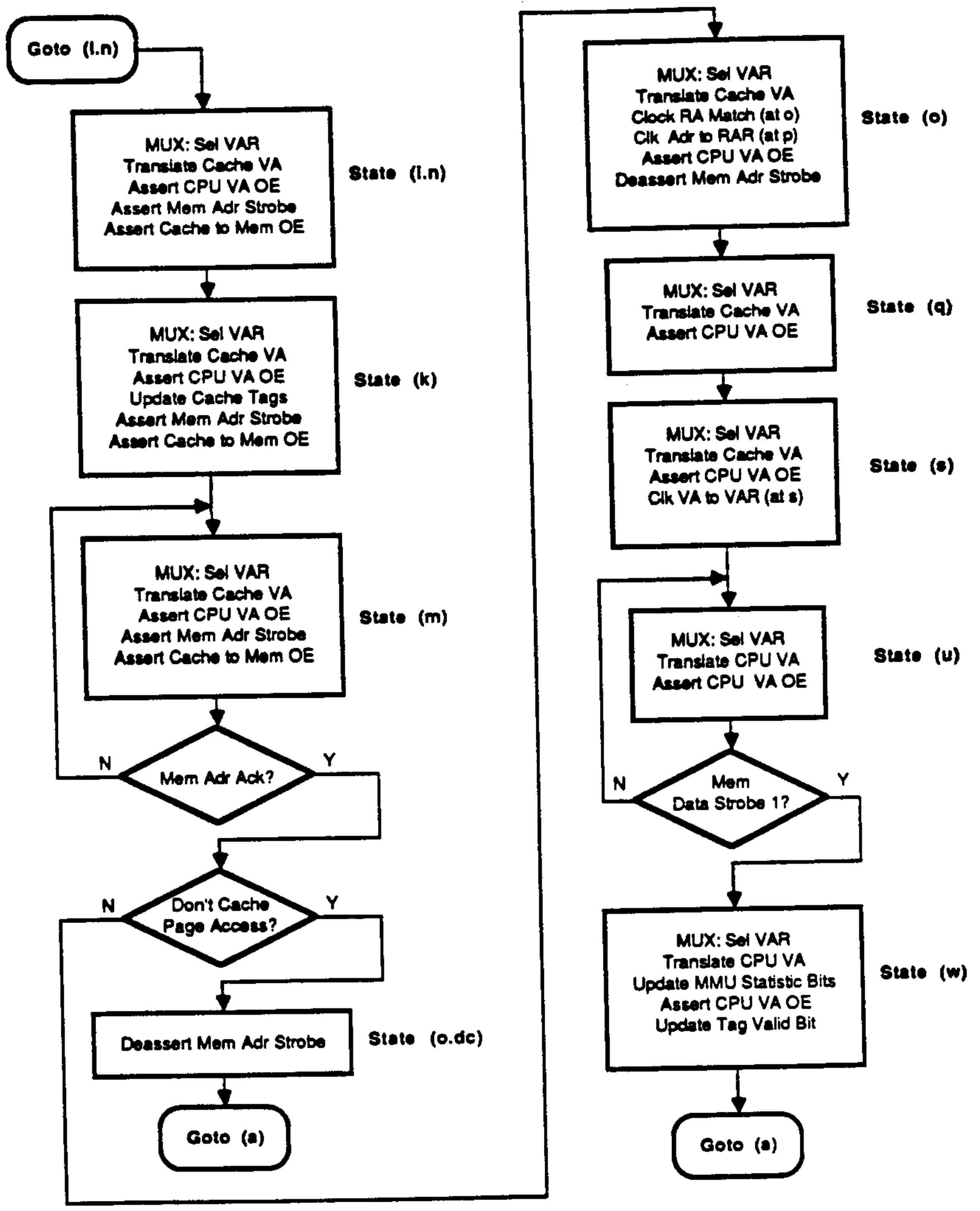


Fig. 4b

Alias Detect Data Path

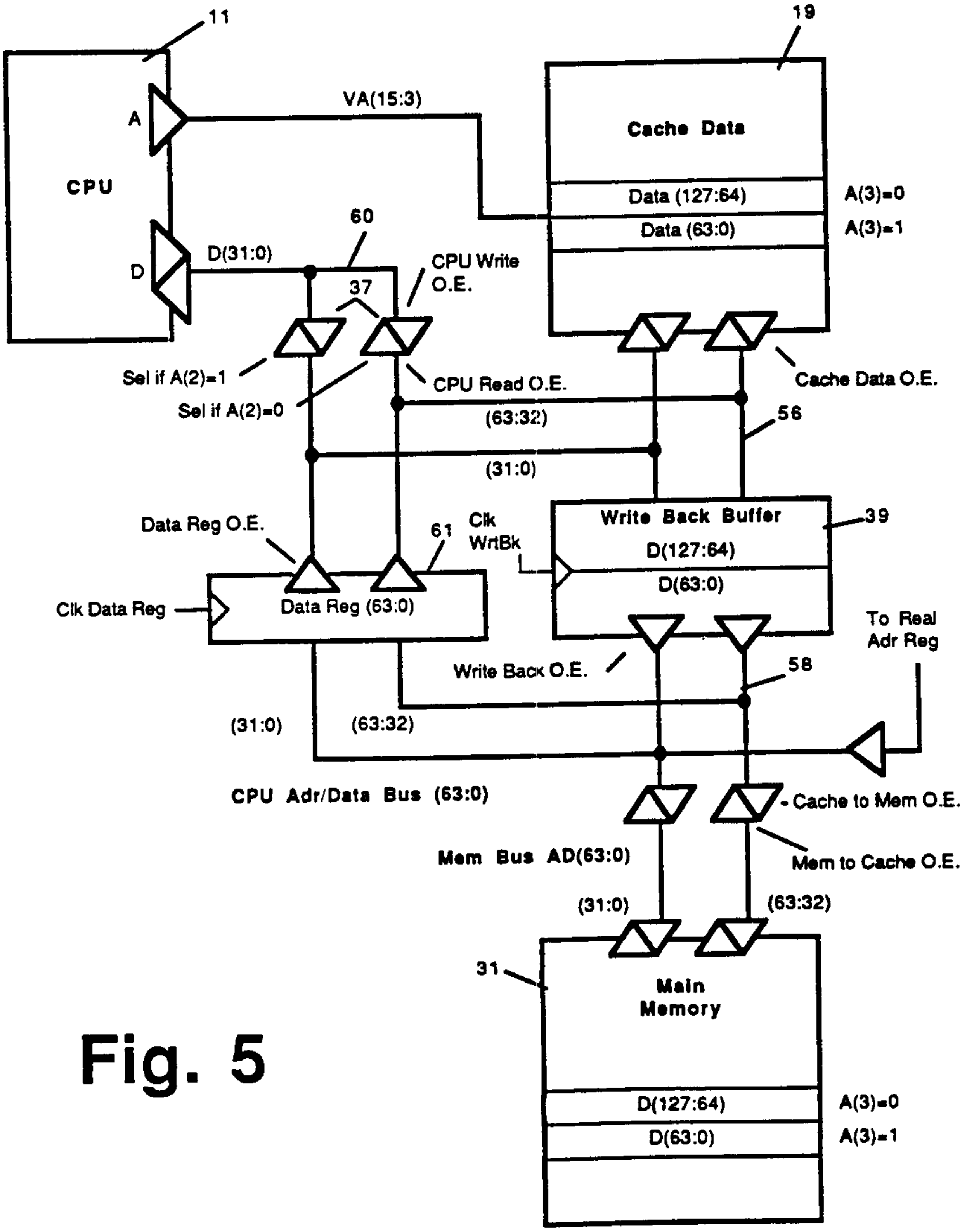


Fig. 5

Alias Detect: Data State Machine

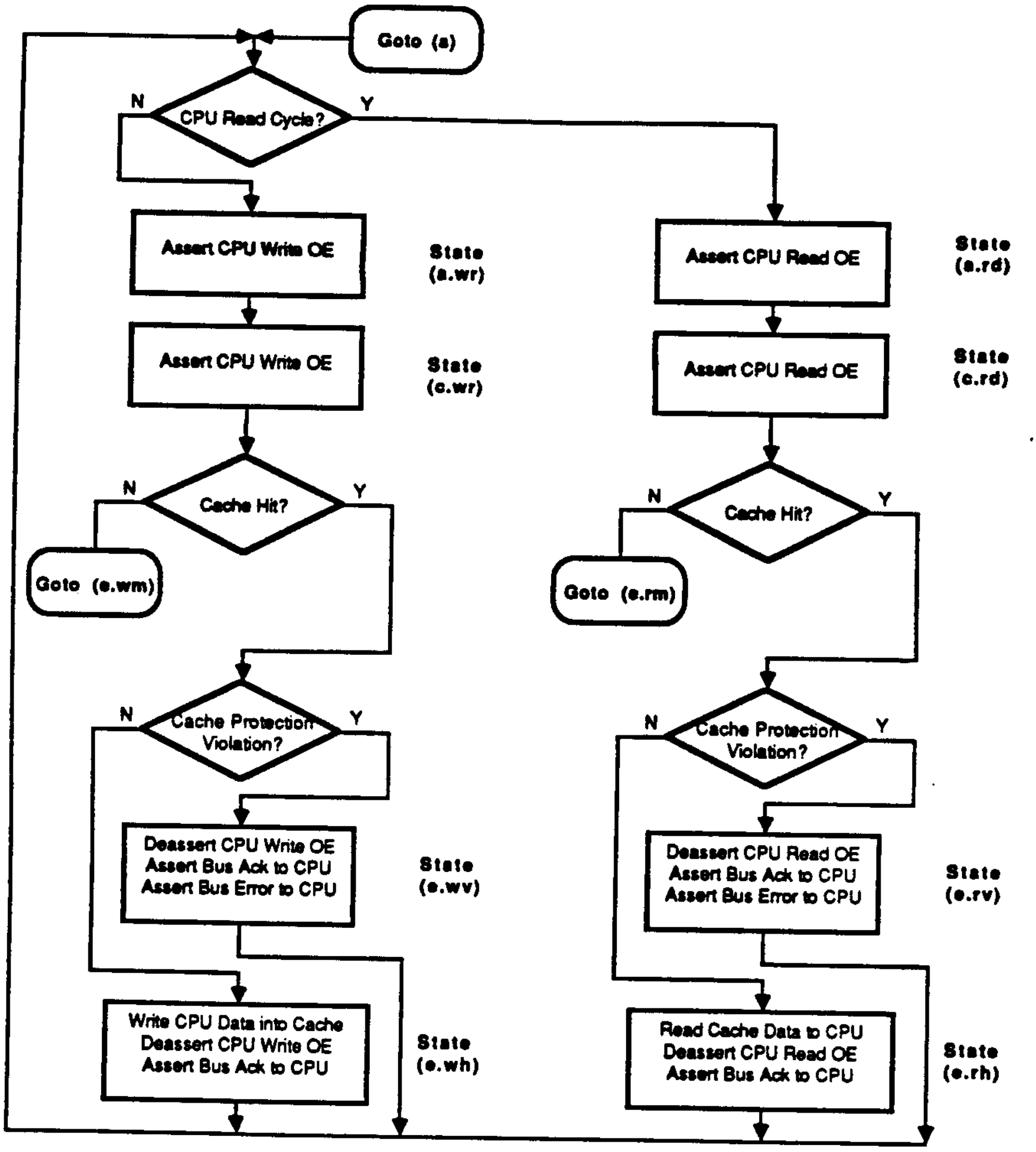


FIG. 6a

Alias Detect: Data State Machine

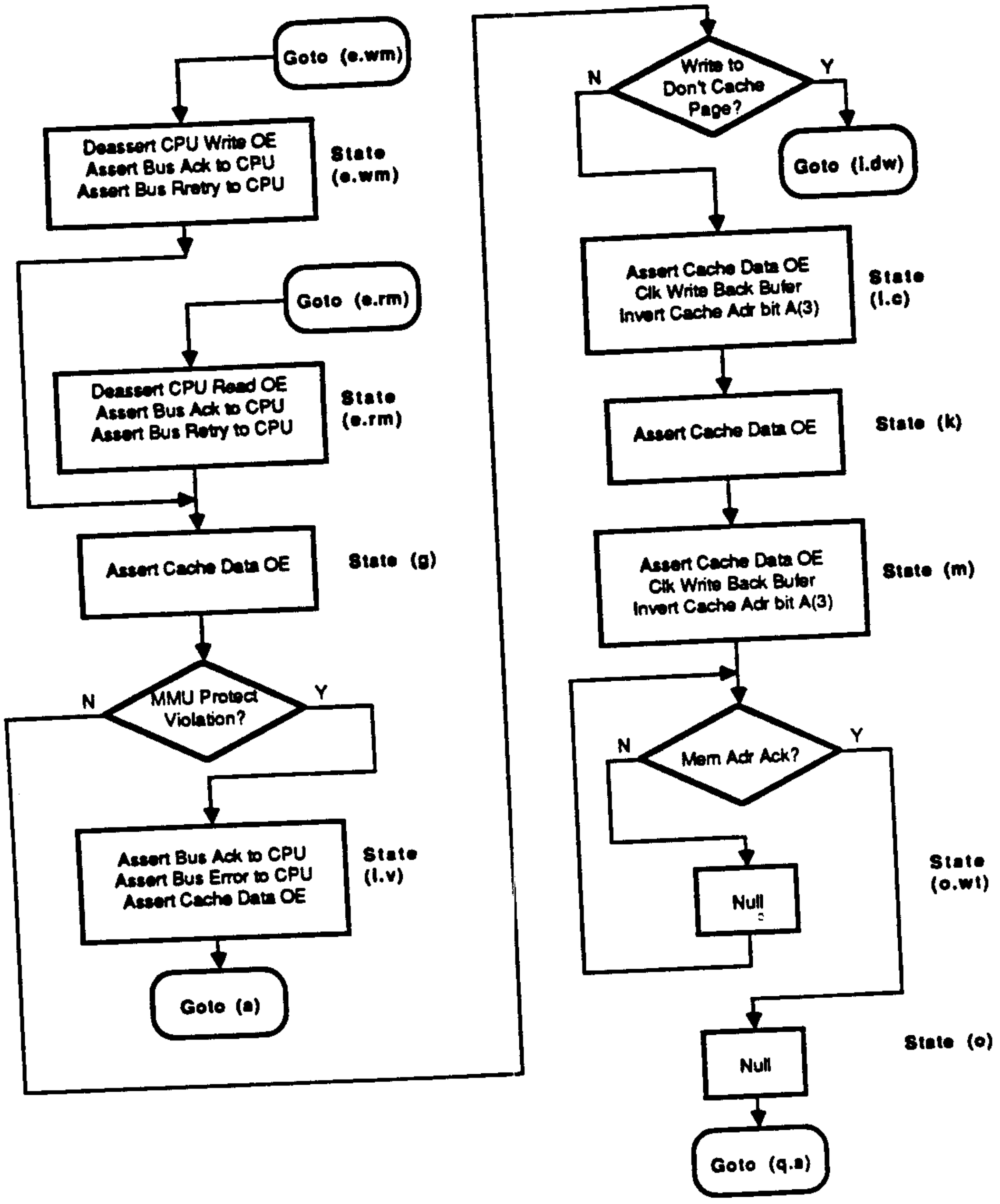


FIG. 6b

Alias Detect: Data State Machine

(Real Address Match)

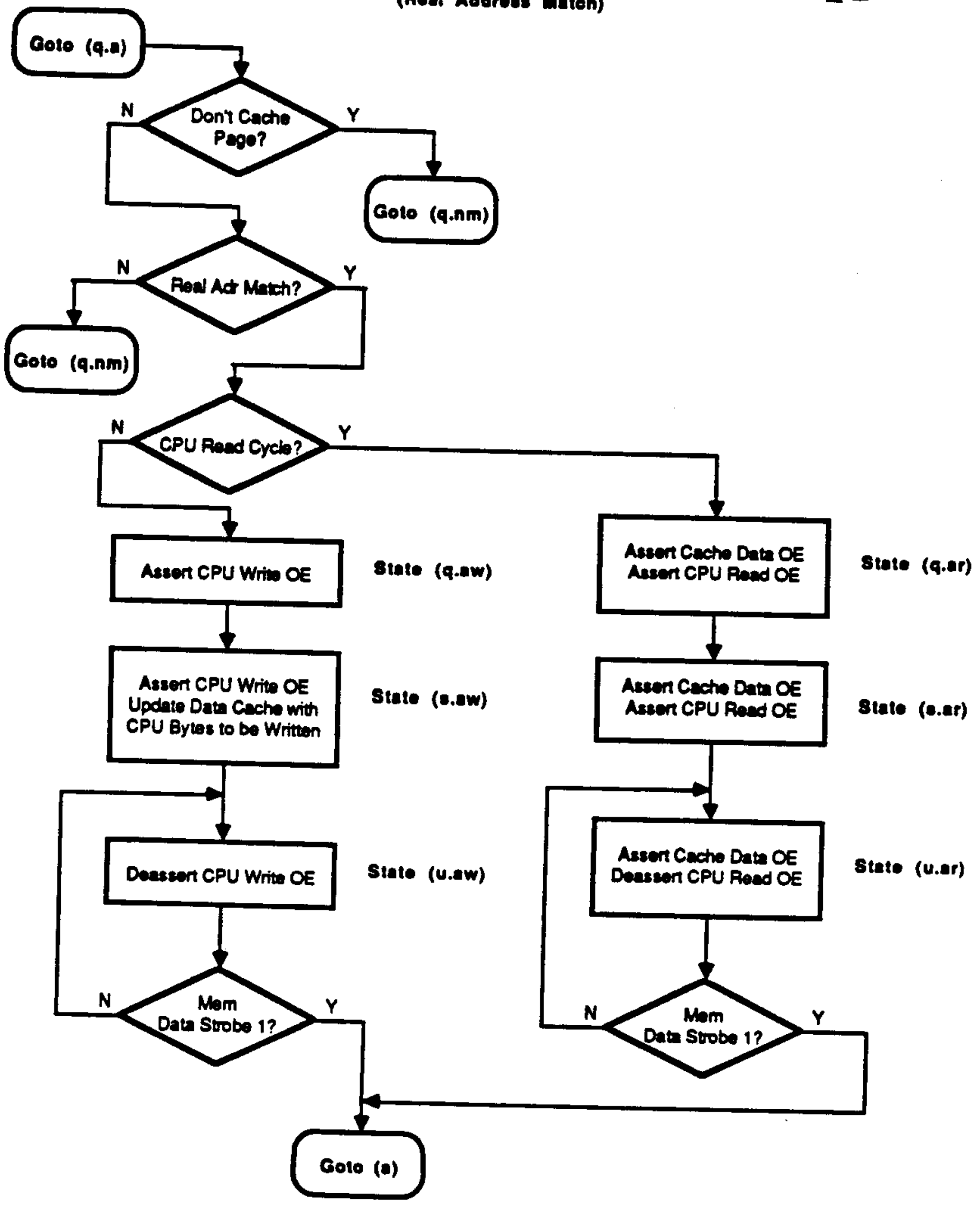


FIG. 7a

2210 ·79

Alias Detect: Data State Machine

(No Match, Write Bus Cycle)

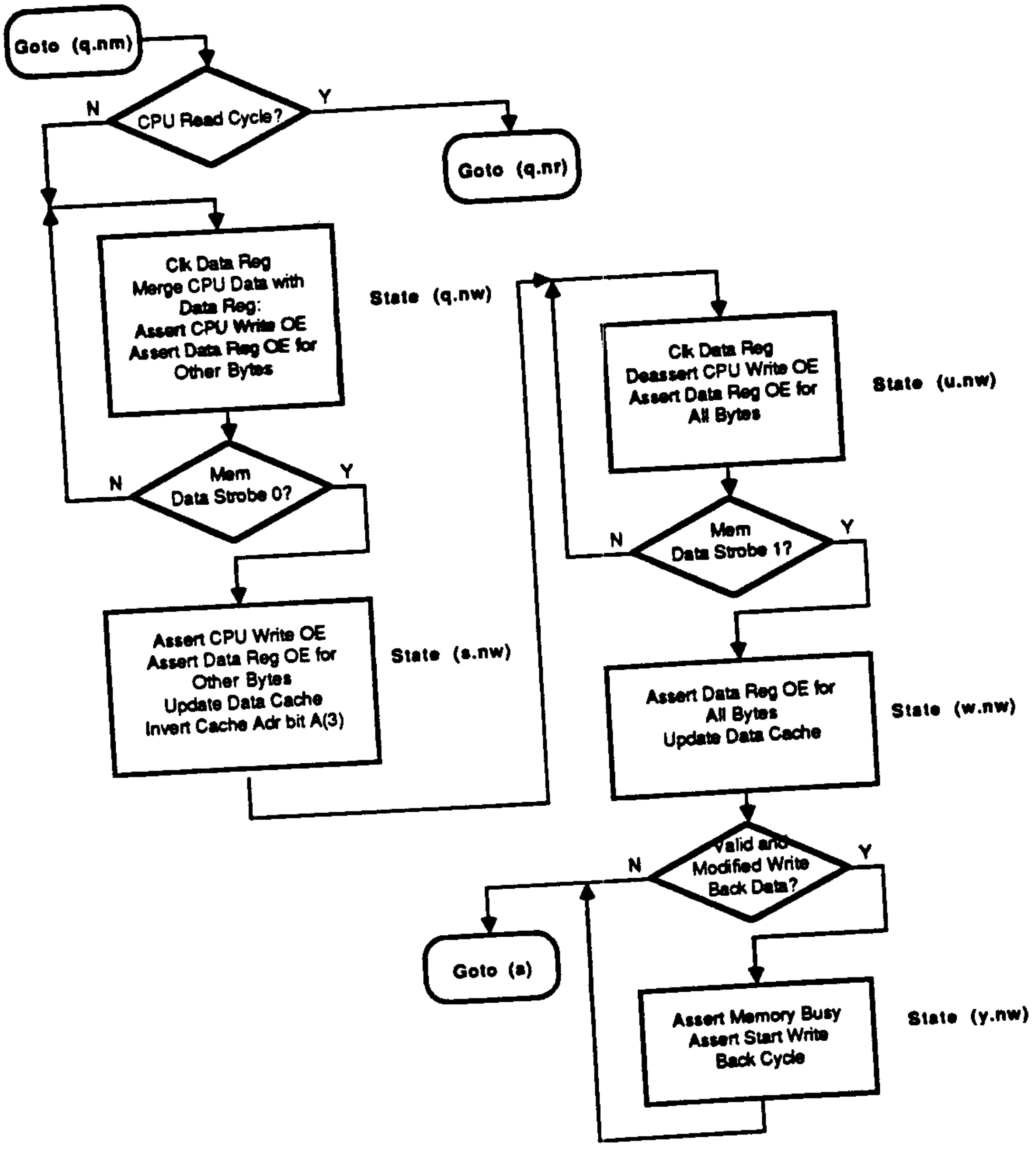


FIG. 7b

Alias Detect: Data State Machine

(No Match, Read Bus Cycle)

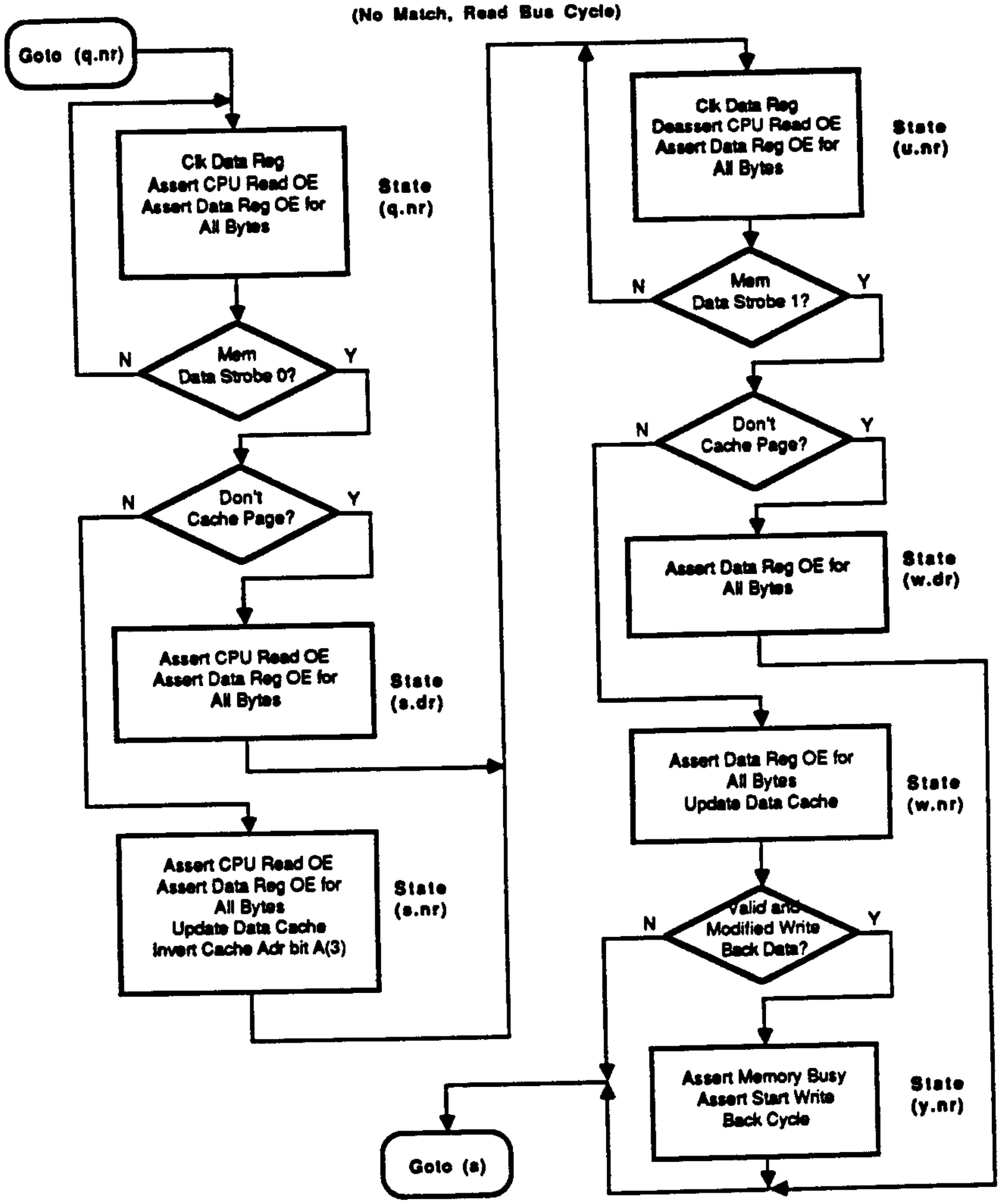


FIG. 7c

2210479

Alias Detect: Data State Machine

(Write Bus Cycle to Don't Cache Page)

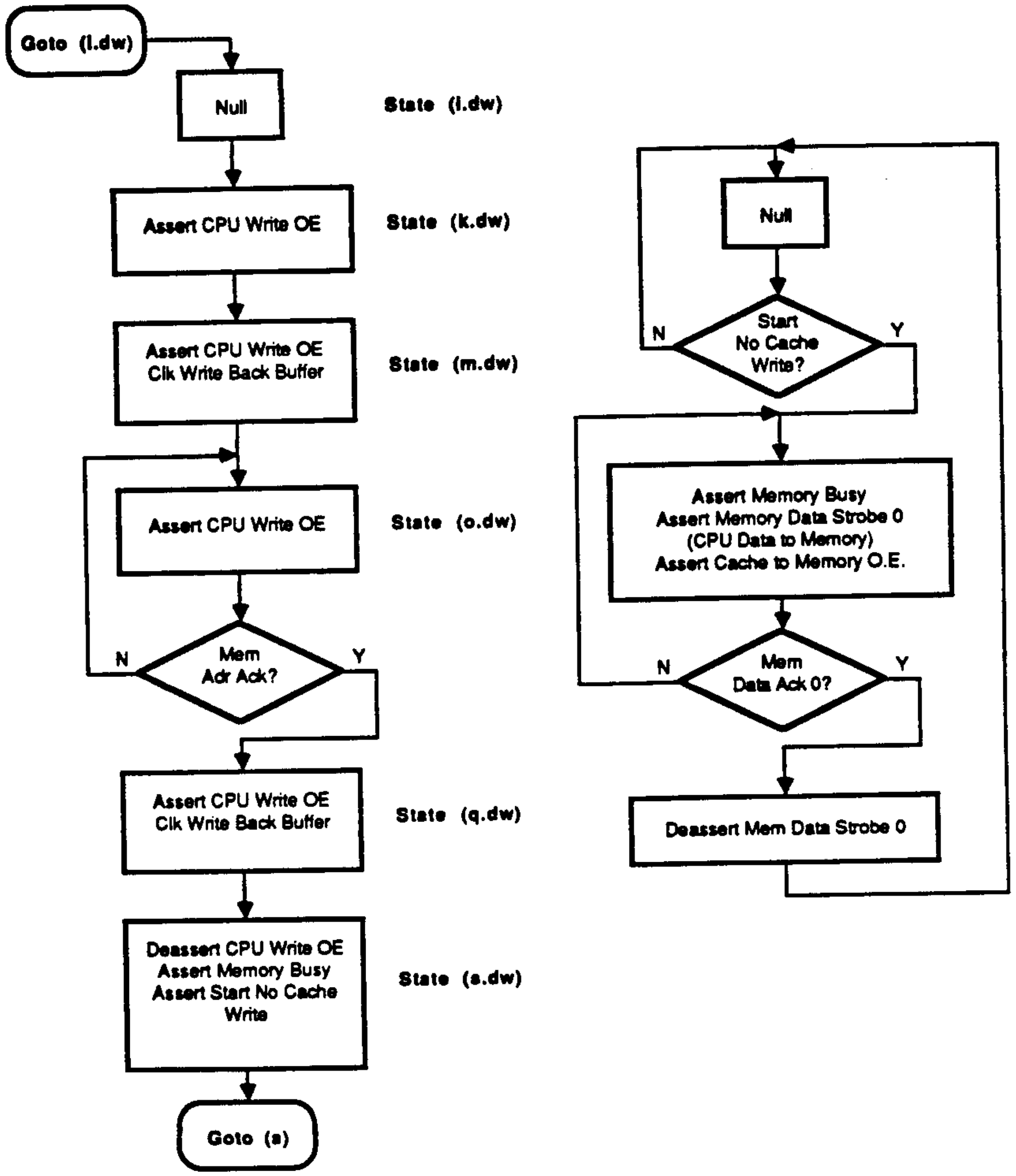
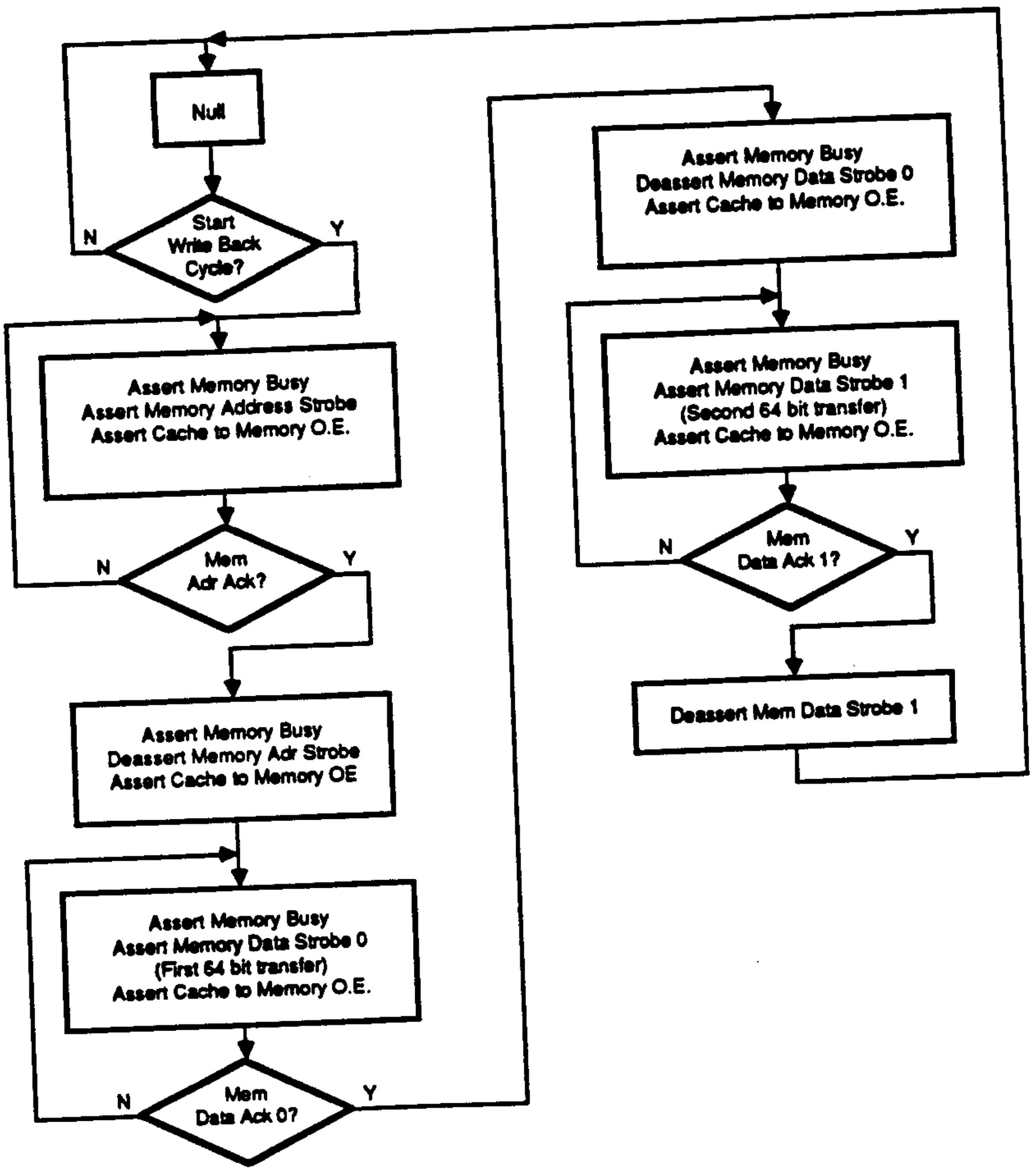


FIG. 7d

2210472

Write Back State Machine



Null
Start Write Back Cycle?
N
Y
Assert Memory Busy
Assert Memory Address Strobe
Assert Cache to Memory O.E.
Mem Adr Ack?
N
Y
Assert Memory Busy
Deassert Memory Adr Strobe
Assert Cache to Memory OE
Assert Memory Busy
Assert Memory Data Strobe 0
(First 64 bit transfer)
Assert Cache to Memory O.E.
Mem Data Ack 0?
N
Y
Assert Memory Busy
Deassert Memory Data Strobe 0
Assert Cache to Memory O.E.
Assert Memory Busy
Assert Memory Data Strobe 1
(Second 64 bit transfer)
Assert Cache to Memory O.E.
Mem Data Ack 1?
N
Y
Deassert Mem Data Strobe 1

FIG. 8

2210459

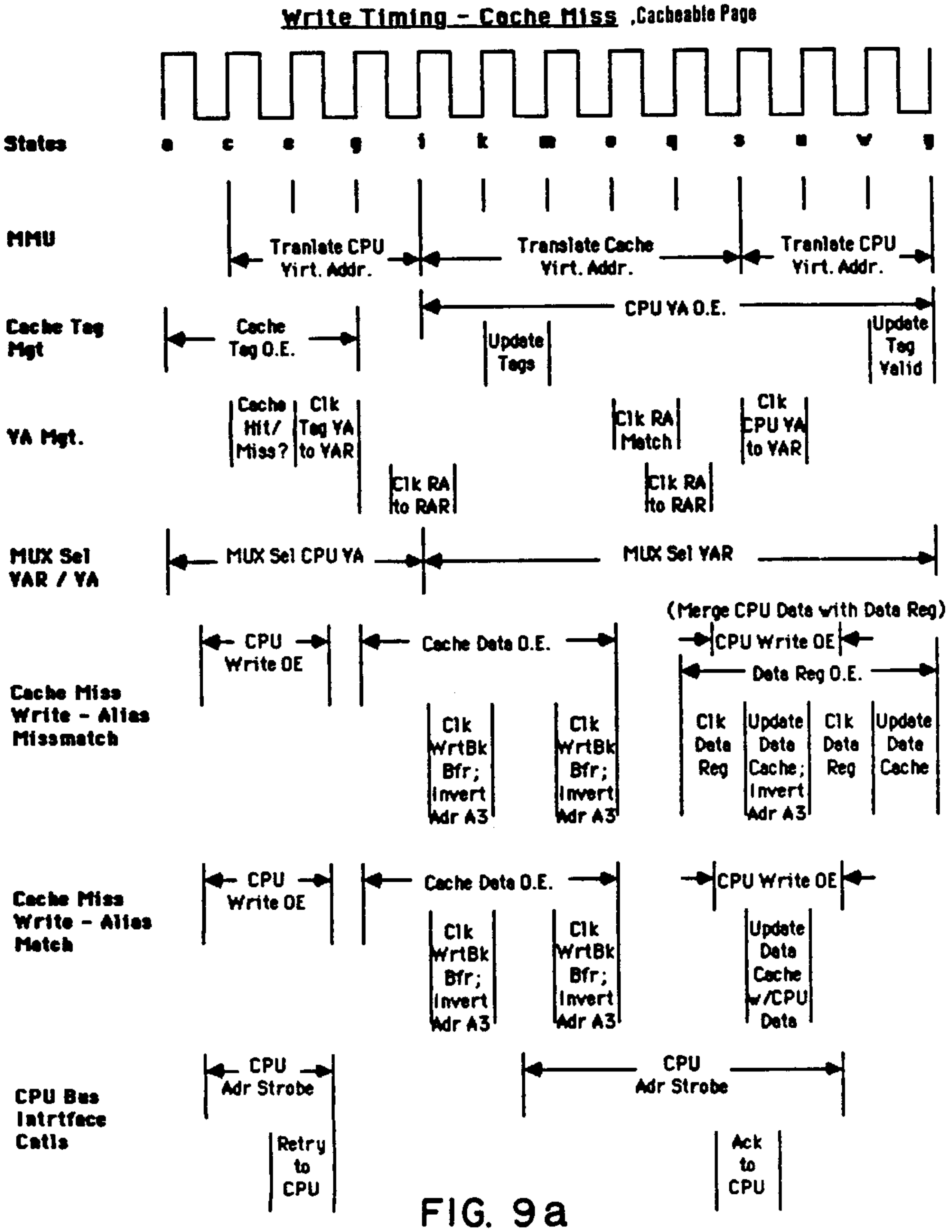


FIG. 9a

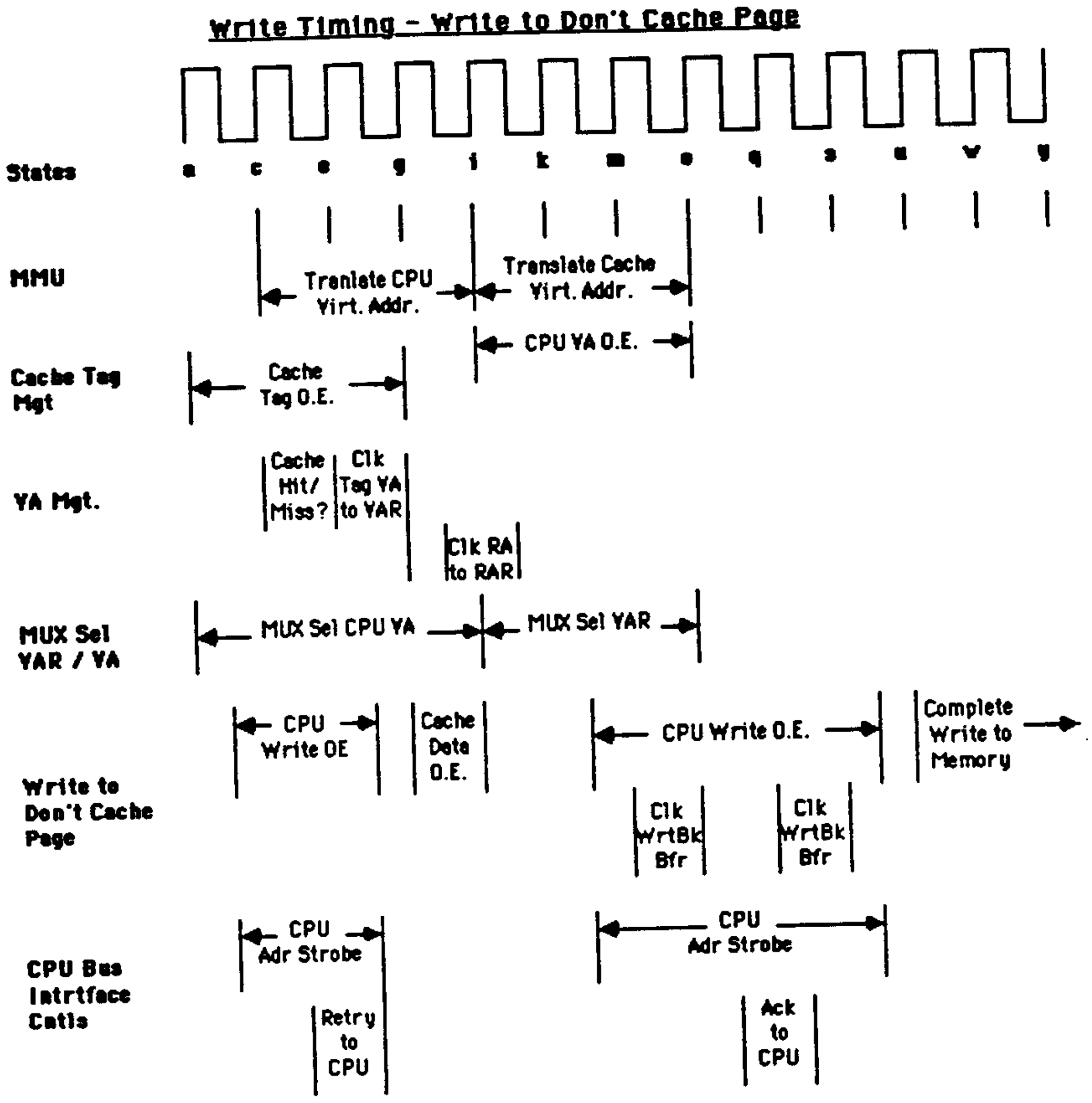
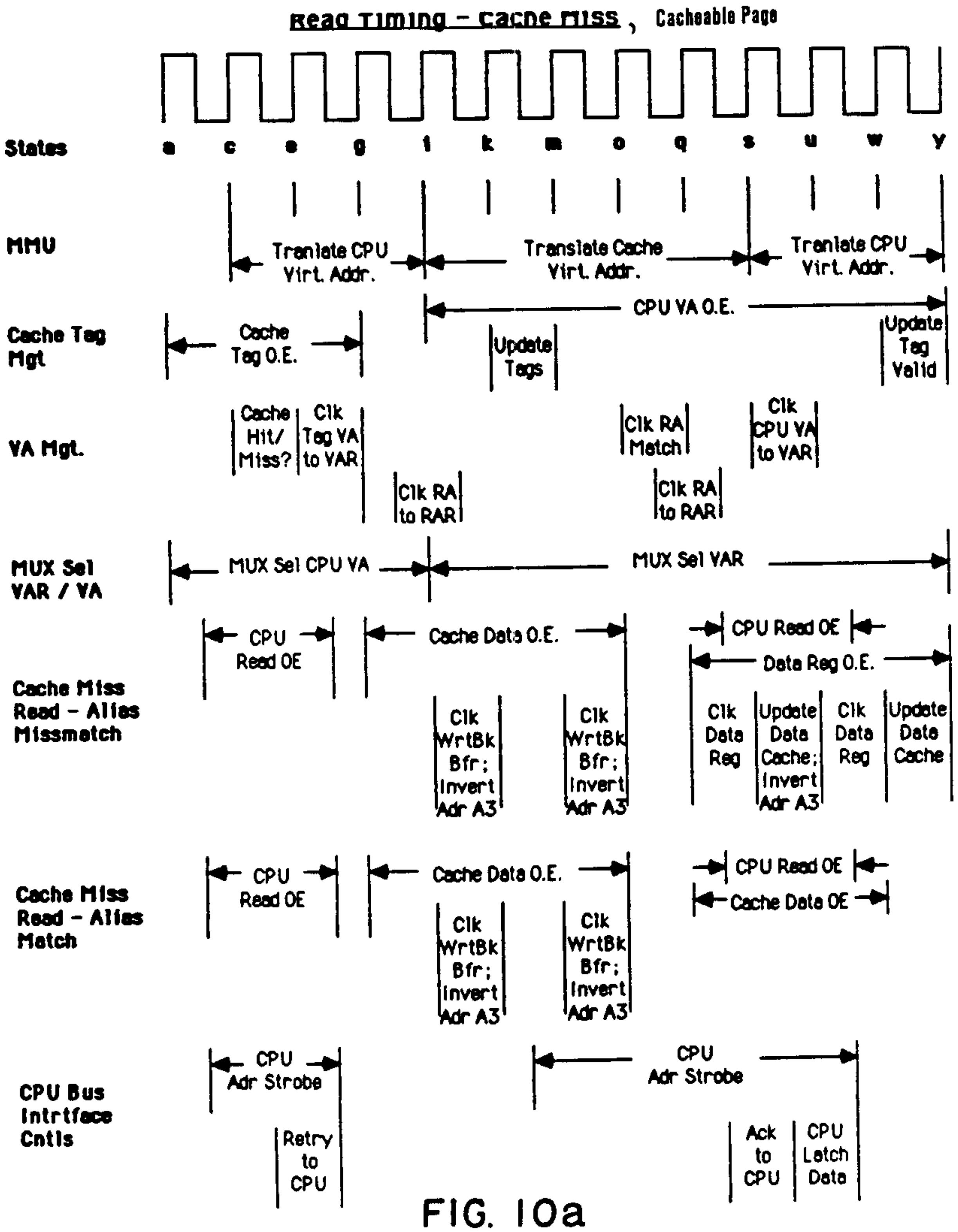

Write Timing - Write to Don't Cache Page
States
a c e g i k m o q s u w y
MMU
Tranlate CPU Virt. Addr.
Translate Cache Virt. Addr.
CPU VA O.E.
Cache Tag Mgt
Cache Tag O.E.
VA Mgt.
Cache Hit/ Miss?
Clk Tag VA to VAR
Clk RA to RAR
MUX Sel VAR / VA
MUX Sel CPU VA
MUX Sel VAR
Write to Don't Cache Page
CPU Write OE
Cache Data O.E.
CPU Write O.E.
Complete Write to Memory
Clk WrtBk Bfr
Clk WrtBk Bfr
CPU Bus Intrtface Cntls
CPU Adr Strobe
Retry to CPU
CPU Adr Strobe
Ack to CPU

FIG. 9b



Read Timing - Cache Miss, Cacheable Page
States
a c e g i k m o q s u w y
MMU
Tranlate CPU Virt. Addr.
Translate Cache Virt. Addr.
Tranlate CPU Virt. Addr.
Cache Tag Mgt
Cache Tag O.E.
CPU VA O.E.
Update Tags
Update Tag Valid
VA Mgt.
Cache Hit/ Miss?
Clk Tag VA to VAR
Clk RA to RAR
Clk RA Match
Clk RA to RAR
Clk CPU VA to VAR
MUX Sel VAR / VA
MUX Sel CPU VA
MUX Sel VAR
Cache Miss Read - Alias Missmatch
CPU Read OE
Cache Data O.E.
CPU Read OE
Data Reg O.E.
Clk WrtBk Bfr; Invert Adr A3
Clk WrtBk Bfr; Invert Adr A3
Clk Data Reg
Update Data Cache; Invert Adr A3
Clk Data Reg
Update Data Cache
Cache Miss Read - Alias Match
CPU Read OE
Cache Data O.E.
CPU Read OE
Cache Data OE
Clk WrtBk Bfr; Invert Adr A3
Clk WrtBk Bfr; Invert Adr A3
CPU Bus Intrtface Cntls
CPU Adr Strobe
Retry to CPU
CPU Adr Strobe
Ack to CPU
CPU Latch Data

FIG. 10a

2210479

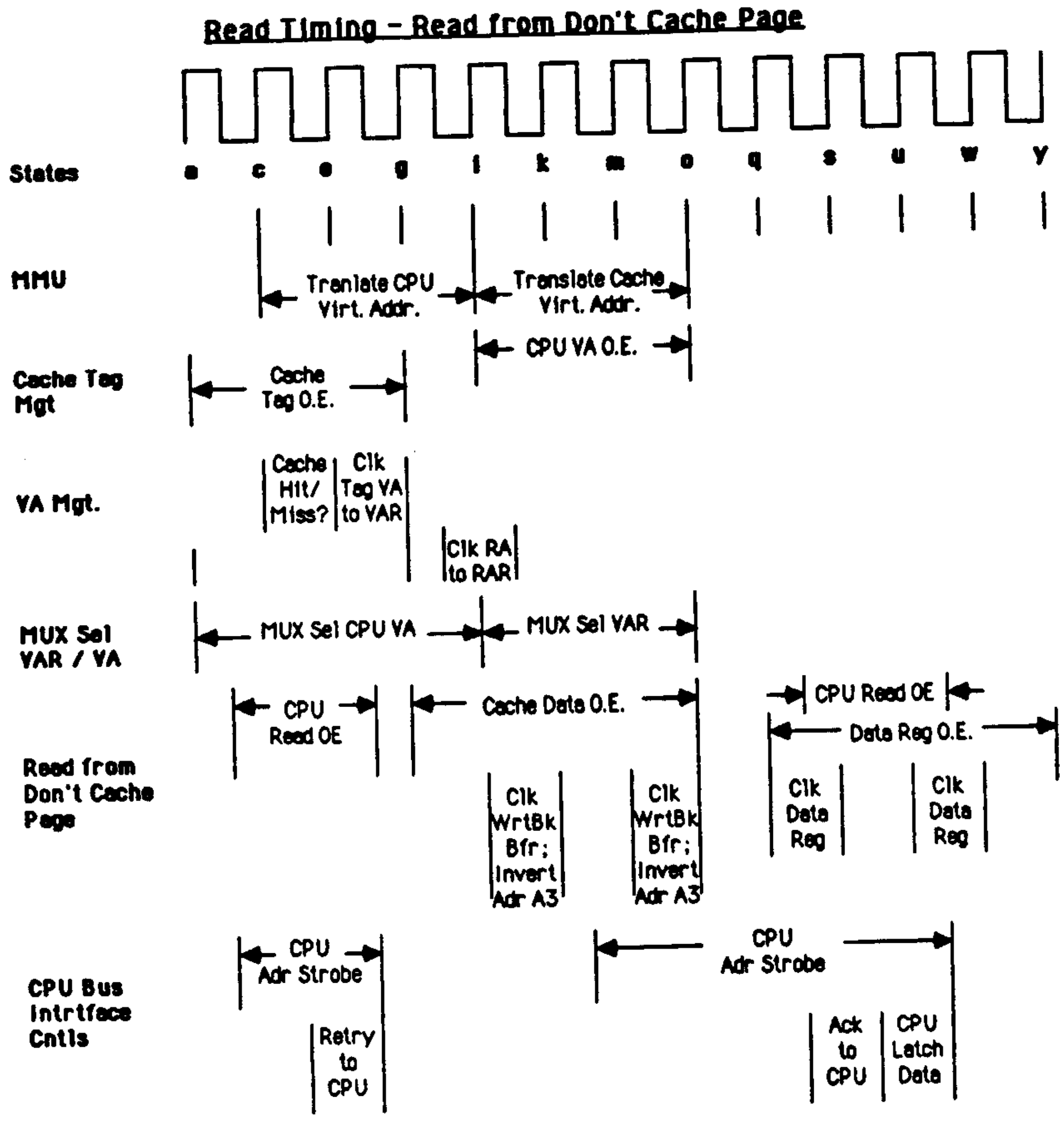


FIG. 10b

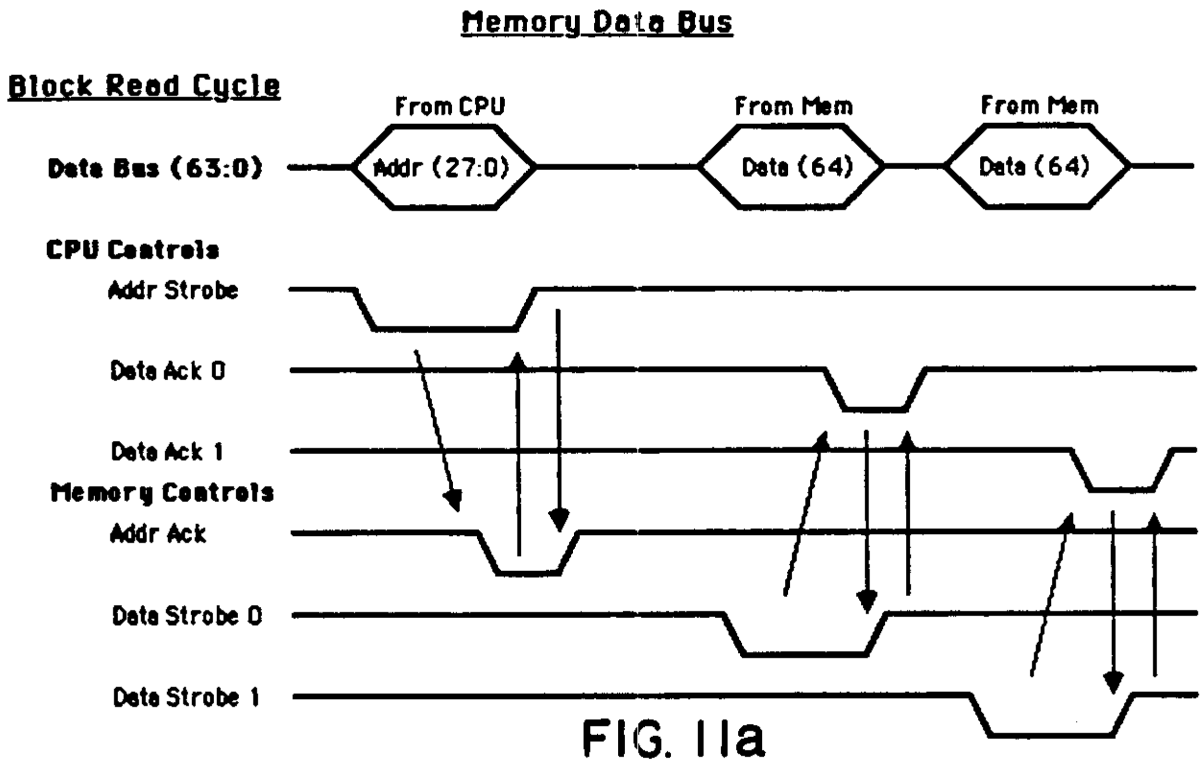


FIG. 11a

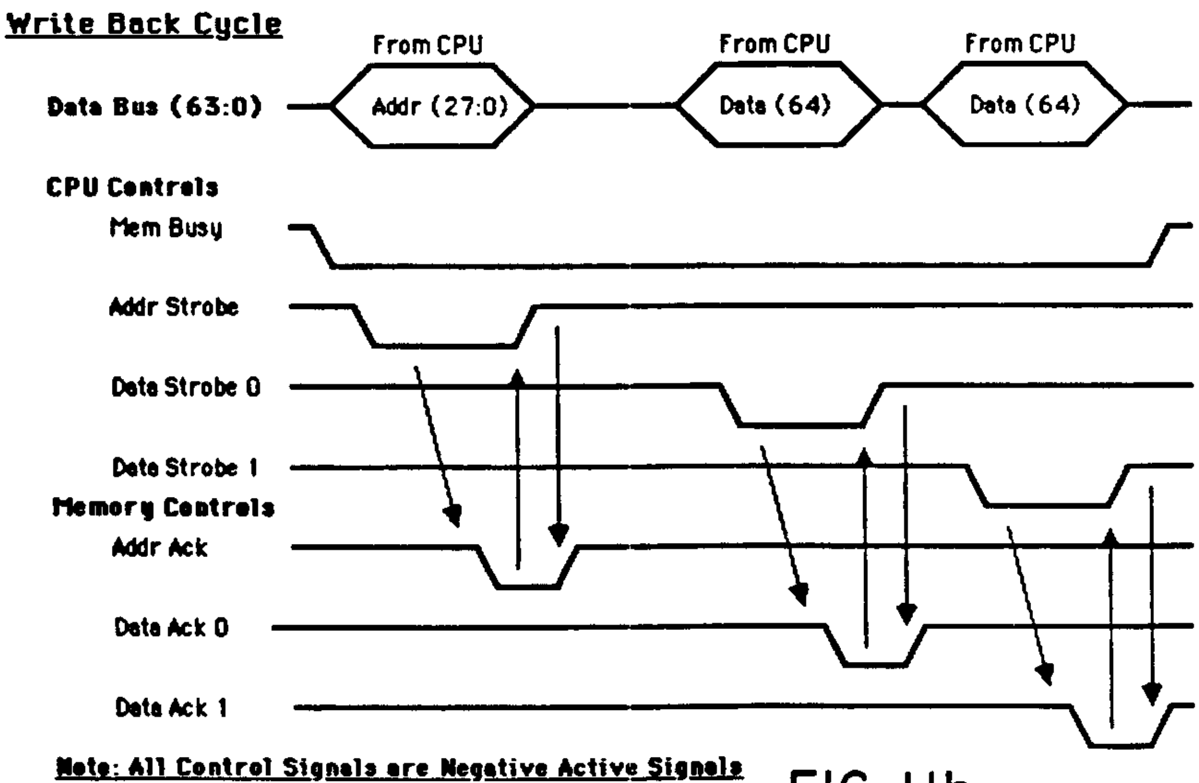


FIG. 11b

Memory Data Bus, con't

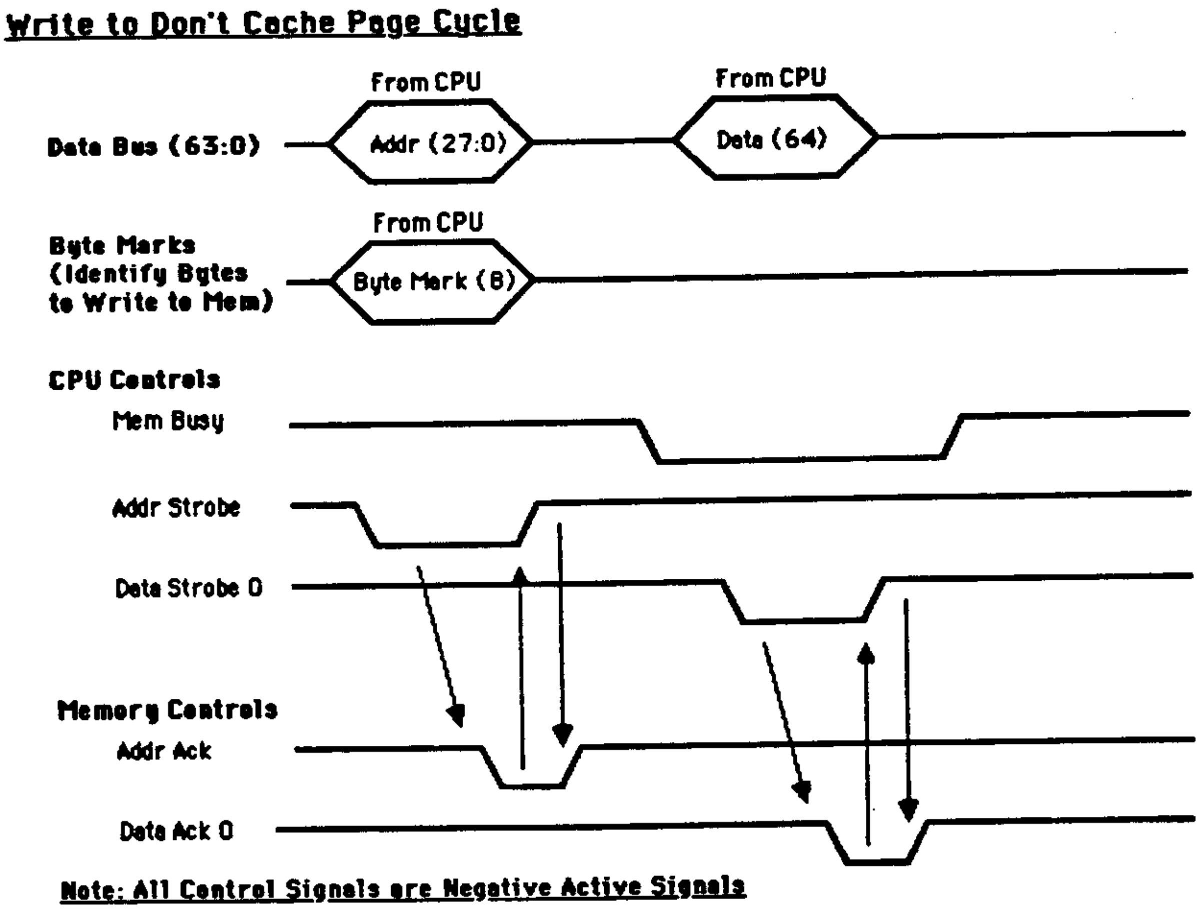


FIG. 11C

ALIAS ADDRESS SUPPORT

SUMMARY OF THE INVENTION

This invention is directed to certain hardware and software improvements in workstations which utilize virtual addressing in multi-user operating systems with write back caches, including operating systems which allow each user to have multiple active processes. In this connection, for convenience the invention will be described with reference to a particular multi-user, multiple active processes operating system, namely the Unix operating system. However, the invention is not limited to use in connection with the Unix operating system, nor are the claims to be interpreted as covering an invention which may be used only with the Unix operating system.

In a Unix based workstation, system performance may be improved significantly by including a virtual address write back cache as one of the system elements. However, one problem which arises in such systems is in the support of alias addresses, i.e., two or more virtual addresses which map to the same physical address in real memory.

The problem arises because any data update into a write back cache which is made through one alias address will not be seen through a cache access to the alias address, since the two alias addresses will not match.

More specifically, virtual addressing allows aliasing, i.e., the possibility of multiple virtual addresses mapping to the same physical address. If a direct mapped, virtual address write back cache were used in a system without page mapping restrictions, any two arbitrary virtual addresses could occupy any two arbitrary cache locations and still map to the same physical address. When cache blocks are modified, in general, it is impossible to check between arbitrary cache locations for data consistency. Data can become inconsistent when changes at one cache location are not seen at another cache location. Ultimately, the data at the common physical address in main memory will include only part of the modifications made by the CPU or I/O device into the several cache locations.

In the present invention, the foregoing problem is solved by combining two distinct strategies to handling aliases.

The first strategy is to create alias addresses so that their low order address bits are identical, modulo the size of the cache (as a minimum). This strategy is applicable to all user programs which use alias addresses generated by the kernel, or wholely within the kernel. These alias addresses for this strategy are generated by modifications to the kernel and are invisible to user programs. The alias addresses so generated will map to the same cache block within a direct mapped (one-way set associative) cache, or within the same cache set within a multi-way set associative cache. Alias hardware detection logic is then used to guarantee data consistency within this cache block (or cache set).

The second strategy covers those alias addresses in the operating system, rather than user programs, which cannot be made to match in their low order address bits. These are handled by assigning their pages as "Don't Cache" pages in the memory management unit (MMU) employed by workstations which utilize virtual addressing.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram showing the main components of a workstation utilizing virtual addresses with write back cache.

Figure 2a is a schematic diagram of cache "hit" logic 25.

Figure 2b is a schematic diagram of a cirçuit for detecting a cache protection violation.

Figure 2c is a schematic diagram of a circuit for detecting a MMU protection violation.

Figure 3 is a detailed block diagram showing the address path utilized by the alias detection logic of the present invention.

Figure 4 (4(a), 4(b)) is a flow diagram of a state machine implementation for certain controls related to the addressing of a virtual address write back cache.

Figure 5 is a detailed block diagram showing the data path utilized by the alias detection logic of the present invention.

Figure 6 (6a, 6b) is a flow diagram of a state machine implementation for certain controls related to data transfers to and from a virtual address write back cache (states (a) - (o)).

Figure 7a is a flow diagram of a state machine implementation for the data path when there is a real address match (states (q) - (u)).

Figure 7b is a flow diagram of a state machine implementation for the data path when there is no real address match during a CPU write bus cycle (states (q) - (y)).

Figure 7c is a flow diagram of a state machine implementation for the data path when there is no real address match during a CPU read bus cycle (states (q) - (y)).

Figure 7d is a flow diagram of a state machine implementation for the data path during a CPU write bus cycle when the MMU indicates a Don't Cache Page.

Figure 8 is a flow diagram of a state machine implementation for controlling Write Back bus cycles to memory.

Figure 9a is a timing diagram for the best case timing for a CPU write bus cycle when the MMU indicates a cacheable page.

Figure 9b is a timing diagram for the best case timing of a CPU write bus cycle when the MMU indicates a Don't Cache page.

Figure 10a is a timing diagram for the best case timing for a CPU read bus cycle when the MMU indicates a cacheable page.

Figure 10b is a timing diagram for the best case timing of a CPU read bus cycle when the MMU indicates a Don't Cache page.

Figure 11a is a timing diagram of the memory bus cycle for performing a block read cycle.

Figure 11b is a timing diagram of the memory bus cycle for performing a write back cycle.

Figure 11c is a timing diagram of the memory bus cycle for performing a write to a Don't Cache page.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 shows the functional blocks in a typical workstation using virtual addresses in which the present invention is implemented.

Specifically, such a workstation includes a microprocessor or central processing unit (CPU) 11, cache data array 19, cache tag array 23, cache hit comparator 25, memory management unit (MMU) 27, main memory 31, write back buffer 39 and workstation control logic 40. Such workstations may, optionally, also include context ID register 32, cache flush logic 33, direct virtual memory access (DVMA) logic 35, and multiplexor 37.

In addition to the foregoing elements, to implement the present invention, also needed are multiplexor 45, alias detect logic 47, alias detect control logic 49 and real address register 51. The foregoing elements support alias addresses without the problems inherent in prior art implementations utilizing a virtual address write back cache.

Each of the foregoing workstation elements will now be described, including changes which must be made to the operating system kernel, with particular emphasis on the components unique to the present invention.

Description of Necessary Elements of Workstation

CPU 11 issues bus cycles to address instructions and data in memory (following address translation) and possibly other system devices. The CPU address itself is a virtual address of (A) bits in size which uniquely identifies bytes of instructions or data within a virtual context. The bus cycle may be characterized by one or more control fields to uniquely identify the bus cycle. In particular, a Read/Write indicator is required, as well as a "Type" field. This field identifies the memory instruction and data address space as well as the access priority (i.e., "Supervisor" or "User" access priority) for the bus cycle. A CPU which may be utilized in a workstation having virtual addressing and capable of supporting a multi-user operating system is a MC68020.

Another necessary element in a virtual address workstation with write back cache shown in Figure 1 is virtual address cache data array 19, which is organized as an array of 2^N blocks of data, each of which contains 2^M bytes. The 2^M bytes within each block are uniquely identified with the low order M address bits. Each of the 2^N blocks is uniquely addressed as an array element by the next lowest N address bits. As a virtual address cache, the (N+M) bits addressing bytes within the cache are from the virtual address space of (A+C) bits. (The (C) bits are context bits from optional context ID register 32 described below.) The (N+M) bits include, in general, the (P) untranslated page bits plus added virtual bits from the (A+C-P) bits defining the virtual page address.

Virtual address cache data array 19 described herein is a "direct mapped" cache, or "one way set associative" cache. While this cache organization is used to illustrate the invention, it is not meant to restrict the scope of the invention which may also be used in connection with multi-way set associative caches.

Another required element shown in Figure 1 is virtual address cache tag array 23 which has one tag array element for each block of data in cache data array 19. The tag array thus contains 2^N elements, each of which has a Valid bit (V), a Modified bit (M), two protection bits (P) consisting of a Supervisor Protect bit (Supvsr Prot) and a Write Allowed bit, and a virtual address field (VA, and optionally CX) as shown in Figure 3. The contents of the virtual address field, together with low order address bits used to address the cache tag and data arrays, uniquely identify the cache block within the total virtual address space of (A+C) bits. That is, the tag virtual address field must contain ((A+C) - (M+N)) virtual address bits.

Cache "Hit" logic 25 compares virtual access addresses with the contents of the virtual address cache tag address field. Within the access address, the lowest order M bits address bytes within a block; the next lowest N bits address a block within the cache; and the remaining ((A+C) - (M+N)) bits compare with the tag virtual address field, as part of the cache "hit" logic.

The cache "hit" logic must identify, for systems with a shared operating system, accesses to user instructions and data, and to supervisor instructions and data. A "hit" definition which satisfies these requirements is illustrated in Figure 2a which comprises comparators 20, AND gate 22, OR gate 24 and AND gate 26.

MMU 27, which translates addresses within the virtual space into a physical address, is another required element. MMU 27 is organized on the basis of pages of size (2^P) bytes, which in turn are grouped as segments of size (2^S) pages. Addressing within a page requires (P) bits. These (P) bits are physical address bits which require no translation. The role of MMU 27 is to translate the virtual page address bits ((A+C-P) or (A-P)) into physical page addresses of (MM) bits. The composite physical address is then (MM) page address bits with (P) bits per page.

MMU 27 is also the locus for protection checking, i.e., comparing the access bus cycle priority with the protection assigned to the page. To illustrate this point, there are two types of protection that may be assigned to a page namely, a Supervisor/User access designator and a Write Protect/Write Allowed designator. Although the subject invention is not limited to such types of protection, given this page protection, a "Protection Violation" can result if either a "User" priority bus cycle accesses a page with "Supervisor" protection; or if a "Write" bus cycle accesses a page with a "Write Protect" designation.

The application of MMU protection checking through the MMU is shown in Figure 2c which comprises inverter 28, AND gates 30a and 30b, OR gate 34 and AND gate 36. In addition, with a virtual address write back cache, the concept of protection checking can be extended to cache only CPU cycles which do not access the MMU. Such cache only protection logic is shown in Figure 2b comprising inverter 42, AND gates 44a and 44b, OR gate 46 and AND gate 48.

Also shown in Figure 1 is main memory 31 which is addressable within the physical address space; control of main memory access is through workstation control logic 40.

Write back buffer 39 is a register containing one block of cache data loaded from cache data array 19. Write back buffer 39 is loaded whenever an existing cache block is to be displaced. This may be caused by a need to update the cache block with new contents, or because the block must be flushed. In either case, in a write back cache, the state of the cache tags for the existing cache block determine wheter this block must be written back to memory. If the tags indicate that the block is valid and modified, as defined below, then the block contents must be written back to memory 31 when the cache block is displaced. Write back buffer 39 temporarily holds such data before it is written to memory.

Workstation control logic 40 controls the overall operation of the workstation elements shown in Figure 1. In the preferred embodiment, control logic 40 is implemented as several state machines which are shown in Figures 4 and 6 - 8 as will be described more fully below in conjunction with the description of alias detect control logic 49 which is also, in the preferred embodiment, integrated into the workstation control logic.

Description of Optional Elements of Workstation

Context ID register 32 is an optional external address register which contains further virtual address bits to identify a virtual context or process. This register, containing C bits, identifies a total of (2**C) active user processes; the total virtual address space is of size 2**(A+C)

An important component in this virtual address space of 2**(A+C) bits is the address space occupied by the operating system. The operating system is common to all user processes, and so it is assigned to a common address space across all active user processes. That is, the (C) context bits have no meaning in qualifying the addresses of pages within the operating system. Rather, the operating system is assumed to lie within a common, exclusive region at the top of the (2**A) bytes of virtual address space for each active context. No user pages may lie within this region. So the operating system page addresses for two distinct user processes are identical, while the user pages for the two processes are distinct. All pages within the operating system are marked as having "Supervisor" protection.

Workstations of the type in which the present invention may be utilized may also include cache flush logic 33 to remove selected blocks from the virtual cache when virtual addresses are to be reassigned.

Cache flush logic 33 is described here only to indicate its role as a component in a virtual address, write back cache system. If a range of addresses (a virtual page address, for example) is to be reassigned, then all instances of addresses from within this range must be removed, or "flushed", from the cache before the new address assignment can be made. A cache block is "flushed" by invalidating the valid bit in its tags and writing the block back to memory, if the block has been modified.

In addition to CPU 11 as a source of bus cycles, the workstation may include one or more external Input/Output (I/O) devices such as DVMA logic 35. These external I/O devices are capable of issuing bus cycles which parallel the CPU in accessing one or more "Types" of virtual address spaces. The virtual address from either the CPU 11 or DVMA logic 35, together with the address in context ID register 32, is referred to as the access address.

Another optional element is data bus buffer 37, which in the preferred embodiment is implemented as two buffers to control data flow between a 32 bit bus and a 64 bit bus. Such buffers are needed when the CPU data bus is 32 bits and the cache data array data bus is 64 bits.

<u>Description of Elements Unique to the Invented Workstation</u>

As noted above, in the present invention, two distinct strategies are utilized to solve the data consistency problems resulting from alias addresses. Both strategies require the interaction of the operating system with special cache hardware to ensure consistent data.

The first strategy requires that all alias addresses which map to the same data must match in their low order address bits to ensure that they will use the same cache location, if the data is to be cached. The present invention utilizes alias detection logic 47, which is a real address comparator, to detect alias addresses on memory accesses that "miss" the cache and to control the cache data update to ensure that all alias addresses point to consistent data within the same cache location.

The kernel addres operation modules implementing this first strategy force alias addresses to match in their low order address bits, so that alias addresses will be guaranteed to use the same cache location. If the cache is of size 2^M blocks of data, each with 2^N bytes, then at least the low order (N+M) bits of the alias addresses must match. This applies to alias addresses within the same process as well as alias addresses between processes. So long as this requirement is met, in direct mapped caches, alias addresses map to the same cache block, and in multi-way set associative caches alias addresses will map to the same cache set. The second strategy prevents data from being cached through the use of a "Don't Cache" bit which is defined for each page in MMU 27. In other words, each page descripter in MMU 27 has a "Don't Cache" bit, which controls whether instructions and data from that page may be written into the cache. If this control bit is set for a page, then all data accesses to this page are made directly to and from main memory, bypassing the cache. In bypassing the cache, the virtual cache data consistency problem is avoided.

Since alias addressing is possible, if a page is marked "Don't Cache" in one MMU page entry, then it must be marked

"Don't Cache" in all alias page entries. Data consistency is not guaranteed otherwise.

Alias address generation for user processes is controlled through the kernel, so that all user processes utilize the first strategy to ensure data consistency among alias addresses. Some addresses for the operating system, however, cannot be altered to meet the addressing requirements of the first strategy. These system alias addresses are handled instead by the second strategy, assignment to "Don't Cache" pages.

The following is a functional description of what is needed to produce data consistency in a direct mapped virtual address, write back cache, using a combination of the two strategies.

If a CPU 11 or DVMA 35 memory access cycle "misses" the cache, then the access virtual address will be translated by the MMU. The MMU translation will determine if the accessed page is a "Don't Cache" page and whether the access has a protection violation. If the access is valid and to a cacheable page, then the cache will be updated with the cache block corresponding to the access address.

The current contents of the cache at the location corresponding to the access address must be examined to detect a possible alias address. If the current cache block is valid and modified, then the translated address of the cache block must be compared to the translated access address to determine the source of valid data to update the cache.

The real address comparison performed by alias detection logic 47 takes as inputs the translated bus cycle access address from real address register 51 and the translated cache address from MMU 27.

If the current cache block is valid, and the translated addresses compare, then the access address and cache block address are aliases. If the cache block is modified, then the current cache data is the most current data, and the main memory data at this address is stale.

If the translated addresses compare but the cache block is not modified, then the old cache data and memory data are identical, and either can be used as the source for the cache update.

Once the source of valid block data has been determined, the access cycle can be completed. On Read cycles, the bus cycle returns data either directly from the source or from the cache following the cache update, depending on the implementation. On Write cycles, the access data may be written into the cache. Both the size of cache updates and cache data alignment are implementation dependent.

To guarantee data consistency, any write to a page requires that all references to that page (read or write) adhere to this restriction. No requirement is placed on alias addressing to read only pages.

The preferred embodiment for the address path incorporating alias detection logic 47 is shown in Figure 3. As shown in Figure 3, the address path includes the fundamental elements to support address control in a virtual address write back cache. For alias address support, also needed are a virtual address register 52 (VAR) for the virtual address (CX and VA) and cache block Valid bit (V), multiplexer 45 which multiplexes the virtual address and virtual address register, real address register 51, alias detect logic 47, AND gate 53 (with the Valid bit from the VAR and the alias detect logic output as inputs), and Real Address Match flip-flop 55 which is set when a real address match is detected.

The data path from the cache 19 to main memory 31 is over two 64 bit busses 56 and 58. The CPU data path 60 is 32 bits, indicated as D(31:0). On read bus cycles, the cache address bit A(2) selects which of two 32 bit buffers 37 may enable data from the 64 bit cache data bus 56 onto the 32 bit CPU data bus 60. Alias detection logic 49 controls the source of the data on read cycle cache misses (the cache or memory) and whether the cache is updated with memory data on write cycle cache misses as described in the data state machine, Figures 6 and 7.

In Figures 3 and 5, to avoid unnecessarily cluttering the Figures, not all control lines are shown. However, the control lines necessary for proper operation of the invention can be ascertained from the flow chart of the state machines shown in Figures 4 and 6 - 8.

In the flow charts, the following abbreviations are utilized:

MUX	- multiplexor 45
Sel	- select
VA	- virtual address
RA	- real address
OE	- output enable
Ack	- acknowledge
Cache Hit?	- Did cache "hit" logic 25 detect a cache hit? (Fig 2a)
Cache Protect Violation ?	- Did control logic 40 detect a detect a cache protect violation? (Fig 2b)

Memory Busy?	- Has Memory Busy been asserted?
MMU Protect Viol?	- Did control logic 40 detect a MMU protect violation?

(Fig 2c)

RAR	- real address register 51
CLK	- clock
Adr	- address
Mem Adr Strobe	- memory 31 address strobe
VAR	- virtual address register 52
Mem Adr Ack?	- Has a memory address acknowledge been asserted by memory 31?
Mem Data Strobe 0?	- Has memory data strobe 0 been asserted?
Mem Data Ack 0?	- Has memory data acknowledge 0 been asserted?
Mem Data Strobe 1?	- Has memory data strobe 1 been asserted?
Mem Data Ack 1?	- Has memory data acknowledge 1 been

	asserted?
Clk Write Back Buffer	- clock write back buffer 39
Real Adr Match?	- Has a real address match been detected (flip-flop 55)
Dont't Cache Page?	- Has control logic 40 detected a Don't Cache Page from MMU 27
CPU Read Cycle?	- Is CPU 11 in a read cycle
Clk Data Reg	- clock data register 61
Valid and Modified Write Back Data?	- Has control logic 40 detected Valid bit(V) and Modified bit(M)
Write to Don't Cache Page	- Has control logic 40 detected a CPU write to a Don't Cache Page?
Start No Cache Write?	- Has control logic 40 asserted Start No Cache Write?
Start Write Back Cycle?	- Has control logic 40 asserted Start Write Back Cycle

Similar abbreviations are used in the timing diagrams of Figures 9 - 11.

The address state machine shown in Figures 4a and 4b defines certain of the controls related to the address handling portion of the cache. The invention is integrated through the clocking of the Real Address Match flip-flop 55 at state (o). The cache tags 23 are written as Valid during state (w), following a successful transfer of all block data from memory 31.

The data state machine shown in Figures 6a and 6b and 7a - 7d defines certain controls related to the data transfer portion of the cache. As illustrated, following state (g), a test is made for a write to a Don't Cache Page; the handling of this write to memory is also described in the path following state (i.dw) in the data state machine. Following state (o), a test is made for a Don't Cache Page access (this time for Read data). The Don't Cache Read control takes the same path as the No-Real Address Match path, until states (q.nr) and (u.nr). Here a test for Don't Cache Pages inhibits cache updates in states (s.nr) and (w.nr).

The write back state machine shown in Figure 8 defines the control of the Write Back bus cycle to memory. This cycle may be performed in parallel with CPU cache accesses, since both the Write Back controls and data path are independent of the cache access controls and data path. As described below, the "Memory Busy" signal causes the address and data state machines to wait until a previous Write Back cycle has completed.

The write cache miss timing diagram shown in Figure 9a defines the overall timing of a CPU write bus cycle to a cacheable page in memory which misses the cache. The cache Hit and Protection Check occur in cycle (c) in this diagram.

A part of the miss handling sequence includes the loading of the current cache block which is being replaced into write back buffer 39 in cycles (i) and (m). The translated address for the current cache block is also loaded into real address register 51 in cycle (o). The Real Address Match latch (flip-flop 55) is also clocked at cycle (o). If the current cache block is both Valid and Modified from a previous CPU (or DVMA) write cycle, then this cache block will be writtten back to memory 31 through a Write Back bus cycle, described in both the Memory Data Bus Timing and the Write Back State Machine, Figures 11b and 8 respectively.

An active Real Address Match latch (flip-flop 55) signifies an alias address match. If there is no Alias Match, the CPU write data is merged with block data returned from memory on the first data transfer of a Block Read memory bus cycle. During cycles (q) through (u), the CPU Write Output Enable controlling buffers 37 will be active for only those bytes to be written by the CPU, while the

Data Register Output Enable controlling data register 61 will be active for all other bytes. During the second data transfer, cycle (w), the Data Register Output Enables for all bytes will be active.

If there is an Alias Match, the CPU data is written into the data cache at state (s), and the data from memory 31 is ignored.

The Write to Don't Cache Page timing shown in Figure 9b defines the overall timing of a CPU write bus cycle to memory for accesses to a Don't Cache Page. The cache Hit, which occurs in cycle (c), will always indicate a miss (no Hit).

The Write to a Don't Cache page case differs from the cache miss case for a write to a cacheable page in that the cache is not updated with either CPU or memory data. The implementation uses a special memory bus cycle, called the Write to Don't Cache Page cycle (Figure 11c), to directly update memory. Note that the Real Address Match latch has no meaning for this case.

The read cache miss timing diagram shown in Figure 10a defines the overall timing of a CPU read bus cycle to a cacheable page in memory which misses the cache. The cache Hit and Protection Check occur in cycle (c) in this diagram.

A part of the miss handling sequence includes the loading of the current cache block which is being replaced into write back buffer 39 in cycles (i) and (m). The translated address for the current cache block is also loaded into real address register 51 in cycle (o). The Real Address Match latch (flip-flop 55) is also clocked at cycle (o). If the current cache block is both Valid and Modified from a previous CPU (or DVMA) write cycle, then this cache block will be writtten back to memory 31 through a Write Back bus cycle, described in both the Memory Data Bus Timing and the Write Back State Machine, Figures 11b and 8 respectively.

An active Real Address Match latch (flip-flop 55) signifies an alias address match. If there is no alias address match, data is read to the CPU by simultaneously bypassing the data to the CPU through buffers 37 enabled by control signal CPU Read Output Enable, active in states (q) through (u), and updating the cache, in state (s). The memory is designed to always return the "missing data" on the first 64 bit transfer, of a Block Read memory bus cycle and the alternate 64 bits on the subsequent transfer. After the CPU read bus cycle data is returned, the CPU may run internal cycles while the cache is being updated with the second data transfer from memory.

If there is an alias address match, data is read directly from the cache 19 to the CPU 11, and the data from memory 31 is ignored.

The Read from Don't Cache Page timing shown in Figure 10b defines the overall timing of a CPU read bus cycle to memory for accesses to a Don't Cache Page. The cache Hit, which occurs in state (c), will always indicate a miss (no Hit).

The Read from a Don't Cache page case differs from the cache miss case for reading from a cacheable page in that the cache is not updated with memory data. The implementation uses the same Block Read memory bus cycle as the cache miss case (see the Memory Data Bus Timing, below). The Real Address Match latch (flip-flop 55) has no meaning for this case.

The Memory Data Bus Timing shown in Figure 11a - 11c shows the timing of Block Read, Write Back, and Write to Don't Cache Page bus cycles. Since the cache block size is 128 bits, each cache block update requires two data transfers. As indicated above the 64 bits containing the data addressed by CPU 11 are always returned on the first transfer for Block Read bus cycles. The "Memory Busy" control signal active during the Write Back cycle is used to inhibit the start of the next cache miss cycle until the previous Write Back cycle can complete.

On Write to Don't Cache Page bus cycles, the 8 bit Byte Mark field, sent during the address transfer phase of the cycle, defines which of the 8 bytes of data, sent during the data phase, are to be updated in memory 31.

In addition to the foregoing hardware, the operating system kernel must be modified in two fundamental ways to support alias addressing:

1) The operating system utilities which generate user alias addresses must be modified to guarantee that alias addresses conform to the rule requiring that their low order (N+M) address bits, as a minimum, must match.

2) Instances of alias addresses inside the operating system, which cannot be made to conform to the rule requiring the match of the low order (N+M) bits, must be assigned to "Don't Cache" pages.

The kernel changes needed to support alias addressing for the Unix operating system are shown in Appendix A.

CLAIMS

1. In a work station having an operating system utilizing a virtual address write back cache, said work station including a central processor coupled to a cache tag array, a cache data array, a write back buffer, a memory management unit, a real address register, a main memory having physical addresses, a cache hit detector and work station control logic, the improvement comprising:

a) first means for ensuring that all alias addresses which map to the same physical address in said main memory, other than a predetermined set of alias addresses which map to physical addresses used exclusively by the operating system, match in their low order address bits thereby using the same location in said cache data array;

b) second means for ensuring that said predetermined set of alias addresses which map to physical addresses used exclusively by the operating system, have their pages marked as Don't Cache pages.

2. The improvement defined by Claim 1 wherein said first means comprises:

a) alias detection logic means for detecting alias addresses which map to physical addresses in said main memory;

b) alias detect control logic means for obtaining the data used on read cycle and write cycle cache misses from a selected one of said cache data array and said memory, and for controlling the updating of the cache data array on write cycle cache misses.

3. The improvement defined by Claim 2, wherein said alias detection logic means comprises:

a) a comparator coupled to said memory management unit and said real address register, said comparator generating a logic one when the address stored in said real address register matches a predetermined cache address in said memory management unit;

b) an AND gate having one input coupled to the output of said comparator and a second input coupled to a cache valid bit within a virtual address register, said virtual address register storing a predetermined virtual address loaded from said cache tag array;

c) a flip-flop coupled to the output of said AND gate, said flip-flop being set when a real address match is detected as determined by the output of said AND gate.

4. The improvement defined by Claim 2, wherein said alias detect control logic means comprises a state machine.

5. The improvement defined by Claim 1, wherein said second means comprises means for indicating in said memory management unit that a page in said main memory is a Don't Cache page.

6. The improvement defined by Claim 5, wherein said indicating means comprises a bit within a page descriptor word for each page in said memory management unit, wherein when said bit is set for a page, all data accesses to said page are made directly to and from said memory thereby bypassing said cache data array.

7. In a work station having an operating system utilizing a virtual address write back cache, said work station including a central processor coupled to a cache tag array, a cache data array, a write back buffer, a memory management unit, a real address register, a main memory having physical addresses, a cache hit detector and work station control logic, a method for detecting data inconsistencies in said data cache array and correcting detected data inconsistencies, said method comprising the steps of:

a) ensuring that all alias addresses which map to the same physical address in said main memory, other than a predetermined set of alias addresses which map to physical addresses used exclusively by the operating system, match in their low order address bits thereby using the same location in said cache data array;

b) marking said predetermined set of alias addresses which map to physical addresses used exclusively by the operating system as Don't Cache pages.

8. The improvement defined by Claim 7 wherein said ensuring step comprises the steps of:

a) detecting alias addresses which map to physical addresses in said main memory;

b) obtaining the data used on read cycle and write cycle cache misses from a selected one of said cache data array and said main memory;

c) selectively updating the cache data array on write cycle cache misses.

9. The improvement defined by Claim 8, wherein said detecting step comprises the steps of:

a) generating a comparator output which is a logic one when the address stored in said real address register matches a predetermined cache address in said memory management unit;

b) inputting to an AND gate one input coupled to the output of said comparator and a second input coupled to a cache valid bit within a virtual address register which stores a predetermined virtual address loaded from said cache tag array;

c) setting a flip-flop coupled to the output of said AND gate when a real address match is detected as determined by the output of said AND gate.

10. The improvement defined by Claim 7, wherein said marking step comprises the step of indicating in said memory management unit that a page in said main memory is a Don't Cache page.

11. The improvement defined by Claim 10, wherein said indicating step comprises setting a bit within a page descriptor word for each page in said memory management unit, wherein when said bit is set for a page, all data accesses to said page are made directly to and from said memory thereby bypassing said cache data array.

12. A work station substantially as hereinbefore described with reference to the accompanying drawings.

REGISTER ENTRY FOR GB2210479

Form 1 Application No GB8819017.8 filing date 10.08.1988

Priority claimed:
02.10.1987 in United States of America - doc: 104635

Title ALIAS ADDRESS SUPPORT

Applicant/Proprietor
SUN MICROSYSTEMS INC, Incorporated in USA - Delaware, 2550 Garcia Avenue, Mountain View, California 94043, United States of America [ADP No. 03961703001]

Inventors
WILLIAM VAN LOO, 1487 College Avenue, Palo Alto, California 94306, United States of America [ADP No. 04089421001]

JOHN WATKINS, 957 Reed Avenue, Sunnyvale, California 94086, United States of America [ADP No. 04089447001]

JOSEPH MORAN, 544 Wildwood Way, Santa Clara, California 95054, United States of America [ADP No. 04089470001]

WILLIAM SHANNON, 261 Trianon Way, Los Altos, California 94022, United States of America [ADP No. 04089496001]

RAY CHENG, 10402 Somerset Court, Cupertino, California 95014, United States of America [ADP No. 04089538001]

Classified to
G4A
G06F

Address for Service
POTTS, KERR & CO, 15 Hamilton Square, BIRKENHEAD, Merseyside, L41 6BR, United Kingdom [ADP No. 00001313002]

Publication No GB2210479 dated 07.06.1989

Examination requested 08.11.1989

Patent Granted with effect from 17.06.1992 (Section 25(1)) with title ALIAS ADDRESS SUPPORT.

**** END OF REGISTER ENTRY ****

```
0A80-01                 OPTICS - PATENTS                18/03/93     13:43:50
FG                                                                  PAGE:     1

                         RENEWAL DETAILS

PUBLICATION NUMBER             GB2210479

 PROPRIETOR(S)

 Sun Microsystems Inc, Incorporated in USA - Delaware, 2550 Garcia
 Avenue, Mountain View, California 94043, United States of America


DATE FILED                     10.08.1988

DATE GRANTED                   17.06.1992

DATE NEXT RENEWAL DUE          10.08.1993

DATE NOT IN FORCE

DATE OF LAST RENEWAL           30.07.1992

YEAR OF LAST RENEWAL           05

STATUS                         PATENT IN FORCE
```