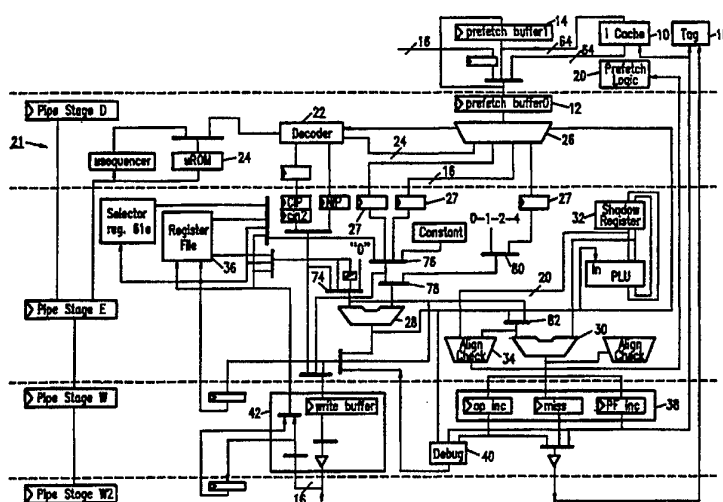




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 9/38	A1	(11) International Publication Number: WO 96/35165 (43) International Publication Date: 7 November 1996 (07.11.96)
(21) International Application Number: PCT/US96/06146 (22) International Filing Date: 1 May 1996 (01.05.96) (30) Priority Data: 08/445,564 6 May 1995 (06.05.95) US (71) Applicant: NATIONAL SEMICONDUCTOR CORPORATION [US/US]; 2900 Semiconductor Drive Mailstop D3579, Santa Clara, CA 95051 (US). (72) Inventors: DIVIVIER, Robert; 1244 Washoe Drive, San Jose, CA 95120 (US). NEMIROVSKY, Mario; 599 West Walbrook Drive, San Jose, CA 95129 (US). (74) Agent: CONSER, Eugene; National Semiconductor Corporation, 2900 Semiconductor Drive Mailstop D3579, Santa Clara, CA 95051 (US).		(81) Designated States: DE, KP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE). Published <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>

(54) Title: INSTRUCTION MEMORY LIMIT CHECK IN MICROPROCESSOR



(57) Abstract

A method and apparatus for efficiently detecting and reporting instruction segment limit violations in the prefetch stage of a pipelined processor and reporting an exception only when it is no longer possible to branch around the limit violation. Whenever a branch occurs in a processor, the execute stage of the processor calculates the number of bytes between the branch destination address and the last valid instruction byte which can be sequentially addressed from the branch destination address. The value is provided to a register in the prefetch stage of the processor. In the prefetch stage, unless and until another branch occurs, the value is decremented each cycle by the number of bytes prefetched during that cycle. If a branch is executed, the new branch destination address is provided to the register from the execute stage. When the value reaches zero, prefetching is halted, but the rest of the pipeline is allowed to continue normal operation until there is no longer a possibility that a branch instruction exists in the pipeline between the prefetch stage and the execute stage. Only then is a segment limit exception reported to the execute stage upon the next program counter pulse.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgyzstan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic	KR	Republic of Korea	SE	Sweden
CG	Congo	KZ	Kazakhstan	SG	Singapore
CH	Switzerland	LI	Liechtenstein	SI	Slovenia
CI	Côte d'Ivoire	LK	Sri Lanka	SK	Slovakia
CM	Cameroon	LR	Liberia	SN	Senegal
CN	China	LT	Lithuania	SZ	Swaziland
CS	Czechoslovakia	LU	Luxembourg	TD	Chad
CZ	Czech Republic	LV	Latvia	TG	Togo
DE	Germany	MC	Monaco	TJ	Tajikistan
DK	Denmark	MD	Republic of Moldova	TT	Trinidad and Tobago
EE	Estonia	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	UG	Uganda
FI	Finland	MN	Mongolia	US	United States of America
FR	France	MR	Mauritania	UZ	Uzbekistan
GA	Gabon			VN	Viet Nam

INSTRUCTION MEMORY LIMIT CHECK IN MICROPROCESSOR

Field of the Invention:

The invention relates to detecting and reporting instruction memory boundary violations. More particularly, the invention relates to a method and apparatus for detecting when the boundary of an instruction memory has been reached in the prefetch stage of a pipelined microprocessor, but not reporting an exception until it is no longer possible to branch around the limit violation.

Background of the Invention:

Modern microprocessors employ pipelining techniques which allow multiple, consecutive instructions to be prefetched, decoded, and executed in separate stages simultaneously. Accordingly, in any given clock cycle, a first instruction may be executed while the next (second) instruction is simultaneously being decoded, and the instruction after that one (a third instruction) is simultaneously being fetched. Since less processing is performed on each instruction per cycle, cycle time can be made shorter. Thus, while it requires several clock cycles for a single instruction to be pre-fetched, decoded, and executed, it is possible to have a processor completing instructions as fast as one instruction per cycle with a very short cycle period, because multiple consecutive instructions are in various stages simultaneously.

Typically, buffers for temporarily holding data are used to define the boundary between consecutive stages of a microprocessor pipeline. The data calculated in a particular stage is written into these buffers before the end of the cycle. When the pipeline advances upon the start of a new cycle, the data is written out of the boundary buffers into the next stage where the data can be further processed during that next cycle.

Most pipelined microprocessor architectures have at least four stages including, in order of flow, 1) a prefetch stage, 2) a decode stage, 3) an execute stage, and 4) a writeback stage. In the prefetch stage, instructions are read out of memory (e.g., an instruction cache) and stored in a buffer. Depending on the particular microprocessor, in any given cycle, the prefetch buffer may receive one to several instructions.

In the decode stage, the processor reads an instruction out of the prefetch buffer and converts it into an internal instruction format which can be used by the microprocessor to perform one or more operations, such as arithmetic or logical operations. In the execute stage, the actual operations are performed. Finally, in the writeback stage, the results of the operations are written to the designated registers and/or other memory locations.

In more complex microprocessors, one or more of the four basic stages can be further broken down into smaller stages to simplify each individual stage and even further improve instruction completion speed.

Generally, instructions are read out of memory in a sequential address order. However, instruction branches, in which the retrieval of instructions from sequential address spaces is disrupted, are common, occurring on average about every four to nine instructions.

The hardware in an instruction prefetch stage typically comprises a prefetch buffer or prefetch queue which can temporarily hold instructions. Each cycle, the decode stage can take in the bytes of an instruction held in the prefetch stage for decoding during that cycle.

Some microprocessor architectures employ what are known as variable width instruction sets. In such architectures, the instructions are not all the same width. For instance, in the instruction set for the x86 family of microprocessors developed by Intel Corporation of Santa Clara, California, an instruction can be anywhere from 1 to 16 bytes wide.

Some microprocessor architectures utilize a segmented address space in which the total memory space is broken down into a plurality of independent, protected address spaces. Each segment is defined

by a base address and a segment limit. The base address, for instance, may be the lowest numerical address in the segment space. The segment limit defines the size of the segment. Accordingly, the end boundary of the segment is defined by the sum of the base address and the segment limit. Alternately, the base address may be the highest address and, as such, the end boundary of the segment would be the difference between the base address and the segment limit.

Software programs are written, compiled and assembled such that, when a program is running, instructions are normally retrieved from sequential addresses in memory for presentation into the pipeline. Accordingly, once a program is begun, the prefetch stage will normally continue to retrieve consecutive instructions for presentation to the decode stage from consecutive addresses in memory until that flow is interrupted. The most common way by which the sequential addressing of instructions can be interrupted is by a branch instruction. A branch instruction usually specifies, in some manner, the address from which the next instruction to be executed after the branch instruction is to be retrieved. Thus, when a branch instruction is executed in the execute stage, the execute stage halts the normal flow of instructions through the preceding stage of the pipe, e.g., the prefetch and decode stages, and instead supplies the next address for retrieving instructions to the prefetch stage. Accordingly, when a branch occurs, the instructions which had been retrieved from sequential addresses after the branch instruction which are in the pipe, i.e., the instructions in the prefetch and decode stages, should not be executed, but should be flushed from the pipe. The flow can be altered by mechanisms other than an executed branch instruction, such as an interrupt. Any change in program flow from sequential addressing is collectively referred to as a branch in this specification, even if it is not the result of a branch instruction.

To generate a linear address according to the x86 architecture, at the very least two quantities are added. Particularly, the base address of the particular segment, as indicated by the segment descriptor and an offset indicating the distance of the desired data (i.e., instruction) from the base of the segment must be added together. The offset itself may comprise up to three more parts, a base, index and displacement. If so, those quantities must be added to generate the offset before the offset could be added to the segment base. A more detailed discussion of segmented addressing in the x86 architecture can be found in INTEL486 Microprocessor Family Programmer's Reference Manual, 1992, Intel Corporation.

In the segmented address space scheme of the x86 architecture, means must be provided to assure that the microprocessor does not execute instructions from address spaces outside of the segment or segments dedicated to instruction memory. Whenever a branch occurs in a program sequence, a limit check should be performed to assure that the branch does not branch to a location outside of the instruction segment. Branching outside of an appropriate instruction segment should not normally occur, but is possible and should be guarded against, since such an error could not only cause a crash, but cause critical data to be overwritten. An error in programming is the most common cause of a branch to an improper address. Accordingly, it is important to prevent the microprocessor from attempting to execute invalid instructions from beyond an instruction segment. Accordingly, most microprocessors provide circuitry in the execute stage for detecting when the destination address of a branch instruction (hereinafter branch address) is illegal and preventing such an address from being presented to the prefetch stage. However, in addition to the possibility that a branch instruction may have an illegal destination address, it is also possible that a branch address is within appropriate segment boundaries, but that continued normal sequential retrieval of instructions from that branch address will eventually go beyond the segment boundary.

Accordingly, it is an object of the present invention to provide an improved limit check method and apparatus.

It is another object of the present invention to provide a limit check method and apparatus in the prefetch stage of a pipelined microprocessor for detecting limit violations during the normal sequential retrieval of instructions from consecutive memory addresses.

It is another object of the present invention to provide a method and apparatus for detecting the loading into the prefetch stage of a pipelined processor of data outside of the instruction segment, but not reporting an exception until it is no longer possible for instruction flow to branch around the limit violation.

- 5 It is a further object of the present invention to provide a limit check method and apparatus requiring very little hardware.

Summary of the Invention

The invention is an efficient method and apparatus for detecting and reporting instruction segment limit violations when programming flow branches to a valid address, but sequential instruction execution
10 from the branch address results in a segment limit violation. Particularly, a segment space value (SEGSPACE) is generated in the execute stage, where the linear address of an instruction to be fetched from memory is generated. In order to keep the execute architecture simple and inexpensive, yet allow most linear addresses to be computed in a single cycle, the execute stage includes two adders which can perform two address additions in the same cycle. Accordingly, all addresses which have only a scaled
15 index and a displacement or a base and a displacement can be calculated in a single cycle.

A first adder calculates the segment offset and a second adder adds the calculated offset to the segment base to generate the linear address. A limit check circuit in the execute state subtracts the calculated offset from the segment limit value to generate the SEGSPACE value which indicates the number of bytes between the instruction pointer and the segment limit. The SEGSPACE value is
20 provided into a SEGSPACE register in the prefetch stage and the prefetch stage starts retrieving instructions sequentially from the branch address. As bytes are brought into the prefetch stage, the value in the SEGSPACE register is decremented by the number of byte retrieved. When the value in the SEGSPACE register reaches zero, a LIMIT HIT latch is set. However, an instruction limit error is not reported, until (1) there are fewer bytes remaining in the prefetch stage than are necessary to complete
25 the instruction currently in the decode stage, (2) a branch has not occurred in the execute stage, and (3) the LIMIT HIT latch is set. When these conditions are met, the decode stage reports an instruction segment limit exception to the execute stage upon the next pulse of the program counter.

An interrupt handler can then perform any recovery functions it has available to it to attempt to recover from the error. If it cannot recover from the error, programming is stopped and a flag is set
30 indicating that the program has been stopped due to an instruction segment limit exception.

Brief Description of the Drawings

Figure 1 is an architectural block diagram of a microprocessor according to a preferred embodiment of the present invention.

Figure 2 is a block diagram showing a logical representation of the prefetch stage of a
35 microprocessor according to a preferred embodiment of the present invention.

Figure 3 is a block diagram of limit violation detecting and reporting circuitry in the prefetch and decode stages of a microprocessor according to a preferred embodiment of the present invention.

Figure 4 is a block diagram of an actual preferred implementation of limit violation detecting and reporting circuitry in the prefetch and decode stages according to a preferred embodiment of the present
40 invention.

Detailed Description of the Invention

The prefetch method and apparatus of the present invention is particularly adapted for use in a microprocessor having a variable width instruction set and more particularly a microprocessor using the instruction set for x86 microprocessors. However, the invention has broader application to any type of
45 processor.

Figure 1 is a block diagram generally illustrating the various pipeline stages of a microprocessor according to a preferred embodiment of the present invention. As shown, the microprocessor is pipelined into five stages, namely, 1) a prefetch stage, 2) a decode stage, 3) an execute stage, 4) a writeback stage, and 5) a second writeback stage.

5 As shown, the prefetch stage includes two prefetch buffers 12 and 14, termed prefetch buffer 0 and prefetch buffer 1, respectively. It also includes a 1 kilobyte instruction cache 16 and a tag memory 18 for storing tag data related to the data in the instruction cache 16. The instruction cache is direct mapped with a line size 8 bytes wide. Both prefetch buffers also are 8 bytes wide, containing byte positions 0 (least significant byte) through byte position 7 (most significant byte). The prefetch stage also includes
10 prefetch logic 20 for performing various functions relating to the control of the loading of the prefetch buffers with instructions.

The decode stage loads from prefetch buffer 0. Accordingly, prefetch buffer 0 defines the boundary between the prefetch stage and the decode stage. The decode stage includes a data extraction unit 26 for separating the various portions of an instruction and forwarding them to the appropriate logic
15 in the decode stage. The decode stage further includes decoder 22 for decoding instructions and a microcode-ROM 24. The data extraction unit 26 separates the instruction into its individual components, including, for instance, prefix byte(s), op-code and operand(s). The prefix and op-code portions are forwarded to the decoder 22. The decoder 22 addresses a particular location or locations in microcode-ROM 24 responsive to the op-code. Microcode-ROM 24 outputs decoded instruction controls for
20 controlling the execute stage to appropriate registers for clocking into the execute stage on the next cycle. Data extraction unit 26 extracts any operands and forwards them to appropriate registers for clocking into the execute stage on the next cycle.

The x86 architecture has over 200 instructions. Depending on the particular instruction, the instruction may include no operands, one operand, or two operands. A branch instruction usually
25 specifies in the operand field the address location to which instruction flow is to jump. The branch address can be specified as a relative address (by providing the number of addresses or bytes from the present address to the branch address) or as a more complex value which must be calculated from the operands in the branch instruction and/or other data). Alternately, the operand might specify a register or memory location from which the address or a portion of the address is to be fetched. The operand might
30 also specify a direct offset address from which the linear address can be generated by adding in the base address of the current segment. Any number of other methods could be used also.

In the execute stage, the instruction is executed and, in the case of branch instructions, a linear address is generated. To generate a linear address according to the x86 architecture, at the very least, two quantities must be added. Particularly, the base address of the particular segment and a value
35 indicating the distance of the desired data from the base of the segment (a segment offset value) must be added. The segment offset value itself may comprise up to three more parts, namely, a base, an index, and a displacement.

In order to keep the architecture simple and inexpensive yet allow most addresses to be computed in a single cycle, the execute stage employs two adders so that two address additions can be performed in
40 one cycle. Accordingly, all addresses which have only a scaled index and a displacement or a base and a displacement can be calculated in a single cycle. The first adder 28 is used to operate on instruction operands for all types of instructions such as add, subtract, multiply, divide, and logical operation instructions as well as instructions requiring address additions. The second adder 30 is only used for calculating linear addresses.

45 Generation of linear addresses is necessary, not only for branch instructions, but also for certain other types of instructions, such as memory and I/O reads. During branch instructions, a first adder 28 generates the segment offset by adding the necessary values. The two particular values which are added, of course, depend on the particular instruction, and are selected by multiplexers 74, 76, 78 and 80. The

second adder 30 adds the calculated segment offset from the first adder to the segment base in order to generate the linear address. The segment base is supplied from the shadow register 32 of the execute stage. The linear address is then used to fetch instructions into the prefetch stage.

As in all memory access operations, if the branch address itself is beyond a segment boundary, it is immediately reported as an exception irrespective of the method and apparatus described herein for detecting limit violations in the prefetch stage.

The segment offset value which is output by first adder 28 is input through multiplexer 82 to a first input terminal of a limit check circuit 34. A segment limit value is supplied from the shadow register 32 to the other input of limit check circuit 34. The limit check circuit subtracts the offset value from the segment limit, thus generating a value, SEGSPACE, which is the number of bytes between the branch address and the end of the segment within which that address resides. The output, SEGSPACE, of limit check circuit 34 is supplied to prefetch logic 20 in the prefetch stage and is used therein to detect and report memory segment boundary violations as will be described in greater detail below.

Figure 2 is a more detailed block diagram showing a logical representation of prefetch stage apparatus for retrieving and handling prefetched instructions according to a preferred embodiment of the present invention. The prefetch stage, according to a preferred embodiment of the invention, utilizes a two tier prefetch buffer system comprising buffers 12 and 14. The provision of a two tier prefetch buffer system helps keep prefetch buffer 0 as full as possible on each cycle and, thus, helps avoid pipeline stalls caused by the unavailability of instructions in prefetch buffer 0 for the decode stage to decode. The decode stage pulls instructions for decoding only out of primary prefetch buffer 12 (hereinafter prefetch buffer 0). Accordingly, prefetch buffer 0 is the data interface between the prefetch stage and the decode stage. Prefetch buffer 14 is a secondary buffer (hereinafter prefetch buffer 1) into which retrieved instruction bytes which cannot be loaded directly into prefetch buffer 0 are stored. The instruction bytes loaded into prefetch buffer 1 are held there until space becomes available in prefetch buffer 0, at which time the bytes are loaded from prefetch buffer 1 into prefetch buffer 0. The decode stage does not interface with prefetch buffer 1, but only with prefetch buffer 0.

As instruction bytes are returned from memory, they are loaded into byte positions within the prefetch buffers 0 and 1 which are dictated by their addresses. Particularly, the 3 LSBs of the address dictate the byte position in the 8 byte wide buffers into which the instruction byte should be loaded. Loading instruction bytes into the prefetch buffers in this manner substantially simplifies circuitry throughout the processor. Cache line and prefetch buffer widths of 8 bytes are preferred because it achieves a substantial reduction in semiconductor area as compared to prior art architectures in which the prefetch buffer (or queue) is as wide as the maximum possible instruction width or wider. However, it is wide enough to accommodate the vast majority of instructions in a single line. This allows the vast majority of instructions to be potentially loaded into the decode stage and decoded in a single cycle.

Instructions are returned to the prefetch stage, either on line 50 from the instruction cache or on line 52 from external memory, responsive to a memory request. The prefetch stage includes a transparent latch 54 for timing purposes. It also includes multiplexers 56 and 58.

The instruction bytes returned from memory (either external memory or cache memory) are loaded directly into prefetch buffer 0, if the appropriate byte position in prefetch buffer 0 is available. Those instructions bytes for which space does not currently exist in prefetch buffer 0 are loaded into the corresponding byte position in prefetch buffer 1. An instruction byte held in prefetch buffer 1 will be loaded into prefetch buffer 0 when the corresponding byte position in prefetch buffer 0 becomes available (i.e., when the decode stage consumes the instruction byte that is currently occupying that byte position in prefetch buffer 0).

Each byte in the prefetch buffers has associated with it a valid tag bit indicating whether or not the data contained in the corresponding byte position of the buffer is a valid instruction byte to be decoded. The valid tag bits are used to determine which bytes are available for loading of incoming instruction

bytes from memory and which bytes are occupied by valid instruction bytes to be decoded and, therefore, cannot be overwritten by the incoming instruction bytes.

The clock enable signals 70 and 72 of prefetch buffers 0 and 1, respectively, and the select control lines 62 and 64 of multiplexers 58 and 56, respectively, are used to direct the instruction bytes returned from memory into the appropriate prefetch buffer 0 or 1. Multiplexers 56 and 58 are each 8 bytes wide, which is the same width as the cache lines and the prefetch buffers. Multiplexers 56 and 58 also each have 8 select control lines 64 and 62, respectively, so that each byte can be individually selected. Also, each byte of the prefetch buffers 12 and 14 can be individually enabled by a separate clock enable signal. Accordingly, the clock enable signals 72 and 70 of prefetch buffers 0 and 1, respectively, also are 8 bits wide.

The select control lines 64 and 62 of multiplexers 56 and 58, respectively, and the clock enable signals 70 and 72 of prefetch buffers 0 and 1, respectively, are controlled in accordance with the following scheme to assure proper loading of data. As instructions are fetched from the cache or external memory, all bytes in byte positions for which there is room in prefetch buffer 0 (i.e., for which the corresponding byte positions in line buffer 12 are tagged invalid) are loaded directly into prefetch buffer 0, bypassing prefetch buffer 1. Simultaneously, valid instruction bytes in prefetch buffer 1 which are in byte positions corresponding to an invalid tagged byte position in prefetch buffer 0 are loaded from prefetch buffer 1 into prefetch buffer 0. Valid bytes from memory and valid bytes from prefetch buffer 1 will never co-exist in the same byte position since a memory request would not have been made unless the data in prefetch 1 in the byte position corresponding to the incoming data from memory was tagged invalid. In other words, a memory request would not be made by the prefetch stage unless there was room in prefetch buffer 1 for the returned data.

Those instruction bytes returned from memory for which room does not presently exist in prefetch buffer 0 are loaded into prefetch buffer 1. After each fetch, the valid tag bits of both prefetch buffers are updated. This updating operation includes consideration of not only the bytes which were loaded into prefetch buffers 1 and 0 during that cycle, but also those bytes in prefetch buffer 0 which were consumed by the decoder during that cycle and, therefore, can be reset from valid to invalid. A more detailed disclosure of the prefetch stage of the microprocessor of the present invention can be found in U.S. Patent Application Serial No. _____, entitled "Two Tier Prefetch Structure And Method With Bypass" (Attorney Docket No. NSC1-65000) filed on even date herewith and incorporated herein by reference.

Figure 3 is a logical representation of the componentry in the prefetch and decode stages for detecting and reporting a segment space violation in a microprocessor made in accordance with the present invention. As shown, the SEGSPACE value calculated by the limit check circuit 34 in the execute section is provided to the first input of a 2:1 multiplexer 84. Multiplexer 84 is controlled to select SEGSPACE responsive to a signal from the execute stage indicating that a branch has been taken in that cycle. The SEGSPACE value is provided through multiplexer 84 into a SEGSPACE register 86, where it is stored. Every cycle, the contents of the SEGSPACE register 86 are outputted to a decrement circuit 88. Decrement circuit 88 subtracts from the SEGSPACE value the number of bytes which were loaded into the prefetch stage during that cycle and returns the new SEGSPACE value to the SEGSPACE register 86 through the other input of 2:1 multiplexer 84. Each cycle, the SEGSPACE value also is provided to a zero detect circuit 90. Zero detect circuit 90 compares the SEGSPACE value to zero and outputs a one bit signal indicating whether the SEGSPACE value has reached zero or not. When zero detect circuit 90 detects that the SEGSPACE value in register 86 has reached 0, it sets a LIMIT HIT latch 92. Also, circuitry determines the last valid byte of instruction data and updates the valid tag bits such that the incoming bytes which are beyond the segment limit are tagged invalid.

Thus, the setting of the LIMIT HIT latch 92 occurs when the last byte in the memory segment has been loaded into one of the prefetch buffers. When this event occurs, there is no point in continuing to

prefetch consecutive instruction bytes out of memory since, by definition, there are no more valid instruction bytes sequentially available. Accordingly, the output of the LIMIT HIT latch 92 is provided to additional prefetch logic (not shown) which will halt the prefetching of instructions unless and until a branch occurs.

5 However, the setting of the LIMIT HIT latch 92 indicates only that the last possible sequential instruction has been prefetched. It does not necessarily mean that this instruction will ever reach the execute stage since a branch could occur before it reaches the execute stage. For instance, it is possible that a branch instruction exists in the pipeline somewhere between the instruction just executed in the execute stage and the last byte in the segment. Particularly, a branch instruction may exist (1) in prefetch
10 buffer 0, (2) in prefetch buffer 1, or (3) in the decode stage currently being decoded. Accordingly, despite the fact that a segment limit hit has been detected, an exception is not reported to the execute stage until it is confirmed that there is no possibility of a branch occurring before that last instruction in the segment space is executed.

 If there are valid instruction bytes in prefetch buffer 1 at the time LIMIT HIT is asserted, then
15 executable instructions may exist in the pipeline which will be executed before the last instruction byte in the segment reaches the execution stage. Such instruction bytes may cause a branch. Thus, in order to assure that the exception is not reported until it is confirmed that no branch will be taken, the one bit output of the LIMIT HIT latch 92 is ANDed in the prefetch stage with the inverse of the valid tag bits of the prefetch buffer 1 by AND gate 94. If there are any valid bytes in prefetch buffer 1, then the output
20 of AND gate 94 will not be asserted. If there are no valid bytes in prefetch buffer 1, then further assurances must be made that there are no instructions in the pipe which might result in a branch.

 Particularly, the output of AND gate 94 is supplied to the decode stage where it is ANDed with the inverse of a BRANCH TAKEN signal 95 and the inverse of an ENOUGH BYTES signal 97. The BRANCH TAKEN signal 95 is received from the execute stage and indicates whether or not the execute
25 stage is currently executing a branch. If it is a taken branch, then the limit error exception should not be reported since the pipeline will be flushed and instruction execution will continue from a new location before the last instruction in the segment is executed.

 The ENOUGH BYTES signal 97 input to AND gate 96 is a one bit signal generated in the decode stage which indicates whether or not enough bytes are tagged valid in prefetch buffer 0 to complete the
30 instruction currently being decoded. The decode stage keeps track of the valid tag bits of prefetch buffer 0 for normal operation regardless of the limit checking function. Accordingly, the decode stage is aware of the number of bytes needed to complete the decoding of the instruction currently in the decode stage as well as the number of valid instruction bytes in prefetch buffer 0. The ENOUGH BYTES signal can be derived from these values through a simple combinational logic circuit.

35 If there are enough bytes available in prefetch buffer 0 to complete decoding of the instruction currently in the decode stage, then it is possible that the instruction currently being decoded is a branch instruction which may avoid the segment limit violation. However, if there are not enough bytes to completely decode the instruction currently in the decode stage, then it is known that an executable branch instruction does not exist in the decode stage. Thus, if (1) the output of AND gate 94 is asserted
40 (indicating that (a) there are no valid instruction bytes in prefetch buffer 1 and (b) the LIMIT HIT latch is set), (2) the BRANCH TAKEN signal 95 is not asserted (indicating that the execute stage is not currently executing a branch), and (3) the ENOUGH BYTES signal is not asserted (indicating that there are not enough bytes in prefetch buffer 0 to complete the instruction currently being decoded), then instruction execution cannot continue without violating the segment limit. Thus, an exception should be
45 reported. Accordingly, the output of AND gate 96, is provided to latch 98 in the decode stage, the contents of which will be forwarded to the execute stage on the next advancement of the program counter. When the output of AND gate 96 has set the latch 98, the execute stage can perform whatever

error recovery operations are available to attempt to recover from the error. If it cannot recover from the error, program execution is halted and the error logged.

Figure 4 is a block diagram illustrating a preferred actual implementation of the prefetch limit detecting and reporting apparatus illustrated in Figure 3. In this implementation, the SEGSPACE register is broken down into two sections. The first section 104a includes the 29 MSBs of the 32 bit SEGSPACE value and the second portion 104b comprises the three LSBs of the SEGSPACE value. Further, the limit check circuit 34 in the execute stage actually generates the one's complement of the number of bytes between the branch address and the segment limit, rather than a positive value. Thus, it should be borne in mind with respect to the discussion of Figure 4, that all arithmetic is in 1's complement rather than in positive values, as were discussed above with respect to Figure 3. As shown, the 29 MSBs of the SEGSPACE value (actually the 1's complement thereof) is input to a first input of a multiplexer 102a, while the 3 LSBs of the SEGSPACE value are input to the first input of a second multiplexer 102b. The select control input signal to multiplexers 102a and 102b, is the BRANCH TAKEN signal 95 generated in the execute stage. If a branch is taken, multiplexers 102a and 102b select the value coming in from the execute stage on SEGSPACE line 120. Otherwise, they select their second inputs on lines 118 and 106, respectively. Thus, when a branch is executed, the 29 MSBs of SEGSPACE are sent through multiplexer 102a to SEGSPACE register 104a and the three LSBs of the SEGSPACE value are forwarded through multiplexer portion 102b to SEGSPACE register 104b. The clock enable for register 104b is asserted whenever the prefetch stage receives data from memory (cache or external) or a branch is taken.

The three LSBs of the SEGSPACE value stored in register 104b are incremented with wrap around from seven (i.e., binary 111) to zero (i.e., binary 000) by the number of instruction bytes retrieved from memory each cycle. In other words, it is incremented in modulo 8. The output 114 of register 104b is added in adder 122 to the number of incoming bytes, e.g., 2 if it is a memory access or 8 if it is a cache hit (assuming no branch). If the carry bit in the third bit position of adder 122 is set, it means that an 8 byte boundary has been crossed and that the 29 MSBs of SEGSPACE, as stored in register 104a, should be incremented by 1. Accordingly, the clock enable input 124 into register 104a is the carry bit in the third bit position of adder 122. The output of register 104a is provided to an INCREMENT-BY-ONE circuit 108 every cycle. However, register 104a is enabled to load data only when the carry bit out of the third bit position of adder 122 is asserted. Thus, the incremented value output by INCREMENT-BY-ONE circuit 108 every cycle does not replace the value in register 104a unless an 8 byte boundary has been crossed in the 3 LSBs of the SEGSPACE value stored in register 104b.

The LIMIT HIT latch is set as follows. The three LSBs of SEGSPACE stored in register 104b are provided on line 114 to a first input of a comparator circuit 122. The second input of comparator circuit 122 is coupled to receive a signal indicating the number of bytes which have been returned from memory. If the 3 LSBs of SEGSPACE are less than or equal to the number of bytes prefetched during this cycle, then comparator 122 asserts its output 126. If the 29 MSBs of SEGSPACE are all ones, the assertion of line 126 indicates that the last instruction byte in the segment has been retrieved into the prefetch stage (it does not matter whether it is in prefetch buffer 0 or prefetch buffer 1). If the 29 MSBs of SEGSPACE are not all ones yet, the condition of line 26 is irrelevant. Accordingly, the output of register 104a also is provided to detect-all-1's circuit 110. When circuit 110 detects that all 29 MSBs of the SEGSPACE value are 1, it asserts line 112. A detection of all 1's in the 29 MSBs of the SEGSPACE value in register 104a indicates that the latest instruction or instructions prefetched are at an address location that is within 8 bytes of the segment limit and, thus, the signal on line 126 indicates whether the last instruction byte in the segment has been loaded into the prefetch stage. Accordingly, the SPACE-WITHIN-EIGHT signal on line 112 is used to validate the signal on line 126. Specifically, the signal on line 126 is ANDed with the output 112 of DETECT-ALL-ONES circuit 110 by AND gate 130. The output of AND gate 130 sets the LIMIT HIT latch 92.

It should be noted that output 126 of comparator 122 and output T12 of DETECT-ALL-ONES circuit 110 are relevant, not only during sequential addressing after a branch, but even during the cycle in which the branch address is first loaded into the registers 104a and 104b. Particularly, the branch address is the address of the first byte after the branch. However, it is possible, and in fact likely, that more than one instruction byte is loaded starting at the branch address. Accordingly, even though the branch address is within the segment space, it is possible that some instruction bytes within the segment and some beyond the segment have been prefetched during the branch cycle.

The remaining circuitry shown in Figure 4 is identical to circuitry shown in Figure 3 and, therefore, will not be described again. The circuit elements in Figure 4 which are identical to circuit elements shown in Figure 3 are identified by the same reference numerals.

The invention embodiments described herein have been implemented in an integrated circuit which includes a number of additional functions and features which are described in the following co-pending, commonly assigned patent applications, the disclosure of each of which is incorporated herein by reference: U.S. patent application Serial No. 08/_____, entitled "DISPLAY CONTROLLER CAPABLE OF ACCESSING AN EXTERNAL MEMORY FOR GRAY SCALE MODULATION DATA" (atty. docket no. NSC1-62700); U.S. patent application Serial No. 08/_____, entitled "SERIAL INTERFACE CAPABLE OF OPERATING IN TWO DIFFERENT SERIAL DATA TRANSFER MODES" (atty. docket no. NSC1-62800); U.S. patent application Serial No. 08/_____, entitled "HIGH PERFORMANCE MULTIFUNCTION DIRECT MEMORY ACCESS (DMA) CONTROLLER" (atty. docket no. NSC1-62900); U.S. patent application Serial No. 08/_____, entitled "OPEN DRAIN MULTI-SOURCE CLOCK GENERATOR HAVING MINIMUM PULSE WIDTH" (atty. docket no. NSC1-63000); U.S. patent application Serial No. 08/_____, entitled "INTEGRATED CIRCUIT WITH MULTIPLE FUNCTIONS SHARING MULTIPLE INTERNAL SIGNAL BUSES ACCORDING TO DISTRIBUTED BUS ACCESS AND CONTROL ARBITRATION" (atty. docket no. NSC1-63100); U.S. patent application Serial No. 08/_____, entitled "EXECUTION UNIT ARCHITECTURE TO SUPPORT x86 INSTRUCTION SET AND x86 SEGMENTED ADDRESSING" (atty. docket no. NSC1-63300); U.S. patent application Serial No. 08/_____, entitled "BARREL SHIFTER" (atty. docket no. NSC1-63400); U.S. patent application Serial No. 08/_____, entitled "BIT SEARCHING THROUGH 8, 16, OR 32-BIT OPERANDS USING A 32-BIT DATA PATH" (atty. docket no. NSC1-63500); U.S. patent application Serial No. 08/_____, entitled "DOUBLE PRECISION (64-BIT) SHIFT OPERATIONS USING A 32-BIT DATA PATH" (atty. docket no. NSC1-63600); U.S. patent application Serial No. 08/_____, entitled "METHOD FOR PERFORMING SIGNED DIVISION" (atty. docket no. NSC1-63700); U.S. patent application Serial No. 08/_____, entitled "METHOD FOR PERFORMING ROTATE THROUGH CARRY USING A 32-BIT BARREL SHIFTER AND COUNTER" (atty. docket no. NSC1-63800); U.S. patent application Serial No. 08/_____, entitled "AREA AND TIME EFFICIENT FIELD EXTRACTION CIRCUIT" (atty. docket no. NSC1-63900); U.S. patent application Serial No. 08/_____, entitled "NON-ARITHMETICAL CIRCULAR BUFFER CELL AVAILABILITY STATUS INDICATOR CIRCUIT" (atty. docket no. NSC1-64000); U.S. patent application Serial No. 08/_____, entitled "TAGGED PREFETCH AND INSTRUCTION DECODER FOR VARIABLE LENGTH INSTRUCTION SET AND METHOD OF OPERATION" (atty. docket no. NSC1-64100); U.S. patent application Serial No. 08/_____, entitled "PARTITIONED DECODER CIRCUIT FOR LOW POWER OPERATION" (atty. docket no. NSC1-64200); U.S. patent application Serial No. 08/_____, entitled "CIRCUIT FOR DESIGNATING INSTRUCTION POINTERS FOR USE BY A PROCESSOR DECODER" (atty. docket no. NSC1-64300); U.S. patent application Serial No. 08/_____, entitled "CIRCUIT FOR GENERATING A DEMAND-BASED GATED CLOCK" (atty. docket no. NSC1-64500); U.S. patent application Serial No. 08/_____, entitled "INCREMENTOR/DECREMENTOR" (atty. docket no.

- NSC1-64700); U.S. patent application Serial No. 08/_____, entitled "A PIPELINED MICROPROCESSOR THAT PIPELINES MEMORY REQUESTS TO AN EXTERNAL MEMORY" (atty. docket no. NSC1-64800); U.S. patent application Serial No. 08/_____, entitled "CODE BREAKPOINT DECODER" (atty. docket no. NSC1-64900); U.S. patent application Serial No.
- 5 08/_____, entitled "TWO TIER PREFETCH BUFFER STRUCTURE AND METHOD WITH BYPASS" (atty. docket no. NSC1-65000); U.S. patent application Serial No. 08/_____, entitled "A PIPELINED MICROPROCESSOR THAT MAKES MEMORY REQUESTS TO A CACHE MEMORY AND AN EXTERNAL MEMORY CONTROLLER DURING THE SAME CLOCK CYCLE" (atty. docket no. NSC1-65200); U.S. patent application Serial No. 08/_____, entitled "APPARATUS
- 10 AND METHOD FOR EFFICIENT COMPUTATION OF A 486TM MICROPROCESSOR COMPATIBLE POP INSTRUCTION" (atty. docket no. NSC1-65700); U.S. patent application Serial No. 08/_____, entitled "APPARATUS AND METHOD FOR EFFICIENTLY DETERMINING ADDRESSES FOR MISALIGNED DATA STORED IN MEMORY" (atty. docket no. NSC1-65800); U.S. patent application Serial No. 08/_____, entitled "METHOD OF IMPLEMENTING FAST
- 15 486TM MICROPROCESSOR COMPATIBLE STRING OPERATION" (atty. docket no. NSC1-65900); U.S. patent application Serial No. 08/_____, entitled "A PIPELINED MICROPROCESSOR THAT PREVENTS THE CACHE FROM BEING READ WHEN THE CONTENTS OF THE CACHE ARE INVALID" (atty. docket no. NSC1-66000); U.S. patent application Serial No. 08/_____, entitled "DRAM CONTROLLER THAT REDUCES THE TIME REQUIRED TO PROCESS MEMORY
- 20 REQUESTS" (atty. docket no. NSC1-66300); U.S. patent application Serial No. 08/_____, entitled "INTEGRATED PRIMARY BUS AND SECONDARY BUS CONTROLLER WITH REDUCED PIN COUNT" (atty. docket no. NSC1-66400); U.S. patent application Serial No. 08/_____, entitled "SUPPLY AND INTERFACE CONFIGURABLE INPUT/OUTPUT BUFFER" (atty. docket no. NSC1-66500); U.S. patent application Serial No. 08/_____, entitled "CLOCK GENERATION
- 25 CIRCUIT FOR A DISPLAY CONTROLLER HAVING A FINE TUNEABLE FRAME RATE" (atty. docket no. NSC1-66600); U.S. patent application Serial No. 08/_____, entitled "CONFIGURABLE POWER MANAGEMENT SCHEME" (atty. docket no. NSC1-66700); U.S. patent application Serial No. 08/_____, entitled "BIDIRECTIONAL PARALLEL SIGNAL INTERFACE" (atty. docket no. NSC1-67000); U.S. patent application Serial No. 08/_____, entitled "LIQUID CRYSTAL
- 30 DISPLAY (LCD) PROTECTION CIRCUIT" (atty. docket no. NSC1-67100); U.S. patent application Serial No. 08/_____, entitled "IN-CIRCUIT EMULATOR STATUS INDICATOR CIRCUIT" (atty. docket no. NSC1-67400); U.S. patent application Serial No. 08/_____, entitled "DISPLAY CONTROLLER CAPABLE OF ACCESSING GRAPHICS DATA FROM A SHARED SYSTEM MEMORY" (atty. docket no. NSC1-67500); U.S. patent application Serial No. 08/_____, entitled
- 35 "INTEGRATED CIRCUIT WITH TEST SIGNAL BUSES AND TEST CONTROL CIRCUITS" (atty. docket no. NSC1-67600); U.S. patent application Serial no. 08/_____, entitled "DECODE BLOCK TEST METHOD AND APPARATUS" (atty. docket no. NSC1-68000).

Having thus described a few particular embodiments of the invention, various alterations, modifications and improvements will readily occur to those skilled in the art. Such alterations,

40 modifications and improvements as are made obvious by this disclosure are intended to be part of this description though not expressly stated herein, and are intended to be within the spirit and scope of the present invention. Accordingly, the foregoing description is by way of example only and not limiting. The invention is limited only as defined in the following claims and equivalents thereto.

CLAIMS

What is claimed is:

1. A method for detecting instruction boundary violations in a pipelined processor comprising the steps of;
5 generating a SPACE value, when a program branch to a branch address occurs, representative of a number of instruction segments between the branch address and a first address in sequential order from said branch address containing data which cannot be executed by said processor,
decrementing said SPACE value, each time instruction segments are retrieved into said processor, by the number of instruction segments which have been retrieved,
10 determining if said SPACE value is zero, and
asserting a LIMIT HIT signal when said SPACE value is zero.
2. A method as set forth in claim 1 further comprising the steps of;
reporting an instruction boundary violation when said LIMIT HIT signal is asserted and it is no longer possible for a branch instruction in a pipeline of the processor to cause instruction flow to branch
15 around said first address before the data retrieved from the address containing non-executable data is to be executed.
3. A method as set forth in claim 2 wherein it is determined that it is no longer possible for a branch instruction in a pipeline of the processor to cause instruction flow to branch around said first address before data retrieved from said first address is to be executed by determining when no
20 unexecuted instructions from memory addresses preceding said first address are in said pipeline.
4. A method as set forth in claim 2 wherein said processor includes a prefetch stage for retrieving instructions from a memory, a decode stage for decoding instructions retrieved from said prefetch stage and an execute stage for executing instructions retrieved from said decode stage, and wherein said reporting step comprises the steps of;
25 determining when there is an insufficient amount of data in the prefetch stage to form an instruction,
determining when the execute stage has taken a branch, and
reporting a limit violation when there is an insufficient amount of data in the prefetch stage to form an instruction, said LIMIT HIT signal is asserted, and said execute stage has not taken a branch.
- 30 5. An apparatus as set forth in claim 4 further comprising the step of;
re-generating said SPACE value each time a branch in program flow occurs.
6. A method as set forth in claim 1 further comprising the step of halting retrieval of instructions when said LIMIT HIT signal is asserted.
7. A method as set forth in claim 6 further comprising the step of resetting said LIMIT HIT
35 signal when a branch occurs.
8. An apparatus for detecting instruction boundary violations in a pipelined processor comprising;
a limit check circuit for generating a SPACE value, when a program branch to a branch address
40 occurs, representative of the number of instruction segments between the branch address and a first

address in sequential order from said branch address containing data which cannot be executed by said processor,

a decrement circuit, coupled to receive said SPACE value, for subtracting a number of instruction segments retrieved from said SPACE value each time instruction segments are retrieved into said processor, and

a comparator coupled to receive said SPACE value and to assert a LIMIT HIT signal when said SPACE value is equal to or less than zero.

9. An apparatus as set forth in claim 8 further comprising a latch for storing said LIMIT HIT signal.

10. An apparatus as set forth in claim 8 further comprising;
a circuit for determining when it is no longer possible that an unexecuted branch instruction exists in the pipeline of the processor, and
a circuit for reporting an instruction boundary violation when said LIMIT HIT signal is asserted and said determining circuit determines that it is no longer possible that an unexecuted branch instruction exists in the pipeline of the processor.

11. An apparatus as set forth in claim 9 wherein said processor includes a prefetch stage for retrieving instructions from a memory, a decode stage for decoding instructions and an execute stage for executing instructions, said apparatus comprising;

a circuit for determining if there is a sufficient amount of data in the prefetch stage to form an instruction and asserting an ENOUGH SEGMENTS signal when there is a sufficient amount of data in said prefetch stage,

asserting a BRANCH TAKEN signal when the execute stage takes a branch, and

reporting a limit violation when said ENOUGH SEGMENTS signal is not asserted, said LIMIT HIT signal is asserted, and said BRANCH TAKEN signal is not asserted.

12. A pipelined processor having a prefetch stage for retrieving instructions from a memory, a decode stage for decoding said instructions and an execute stage for executing said decoded instruction comprising;

a limit check circuit in said execute stage for generating, when a program branch to a branch address occurs, a SPACE value by subtracting said branch address from a first address in sequential order from said branch address containing data which cannot be executed by said processor,

a register for storing said SPACE value,

a decrement circuit, coupled to said register to receive said SPACE value, for subtracting from said SPACE value, each time instructions are retrieved into said processor, the number of instruction segments retrieved and rewriting said SPACE value in said register, and

a comparator coupled to receive said SPACE value and to assert a LIMIT HIT signal when said SPACE value is equal to or less than zero,

a latch which is set by said LIMIT HIT signal, and

means for halting retrieval of instructions from said memory when said latch is set.

13. An apparatus as set forth in claim 13 wherein said prefetch stage includes first and second prefetch buffers, each comprising a plurality of storage positions for storing instruction segments retrieved from said memory, said first buffer providing instruction segments to said decode stage for decoding and said second buffer temporarily holding instruction segments to be loaded into said first buffer, said prefetch stage further including a tag bit corresponding to each storage position in each of

said buffers, said tag bits indicating whether said corresponding storage position contains a valid instruction segment to be decoded, said apparatus further comprising;

means in said execute stage for generating a BRANCH TAKEN signal which is asserted when the execute stage takes a branch,

- 5 means for generating an ENOUGH SEGMENTS signal which is asserted when there are enough segments in said first buffer to complete the instruction currently in the decode stage,

a first AND gate coupled to receive said LIMIT HIT signal from said latch and said tag bits corresponding to said storage positions in said second buffer, said first AND gate having an output signal which is asserted when said LIMIT HIT signal is asserted and all of said tag bits corresponding to said

- 10 storage positions in said second buffer are invalid, and

a second AND gate coupled to receive output signal from said first AND gate at a first input, said BRANCH TAKEN signal from said execute stage at a second input and said ENOUGH SEGMENTS signal at a third input and having an output signal which is asserted when said output signal of said first AND gate is asserted and said BRANCH TAKEN and ENOUGH SEGMENTS signals are not asserted.

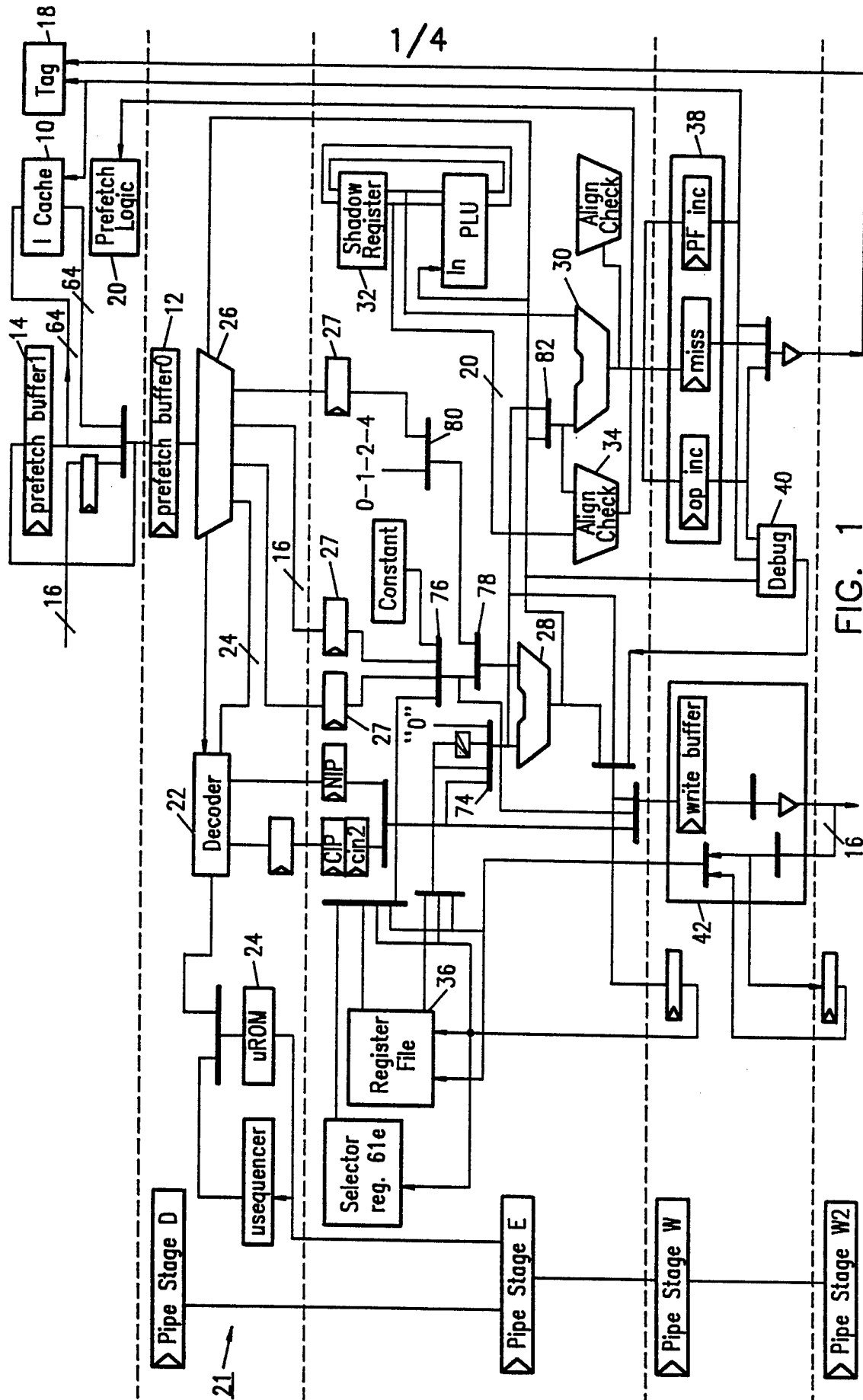


FIG. 1

2/4

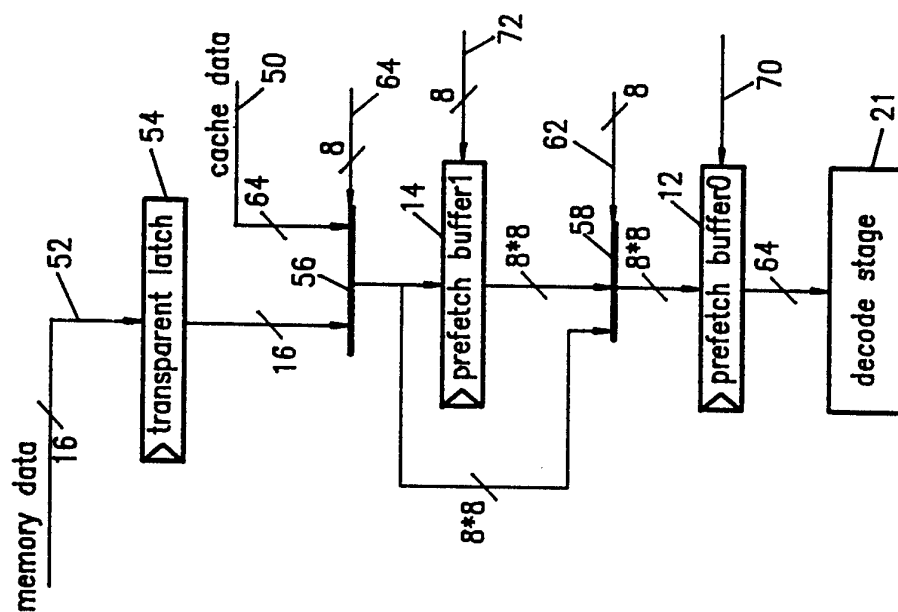


FIG. 2

3/4

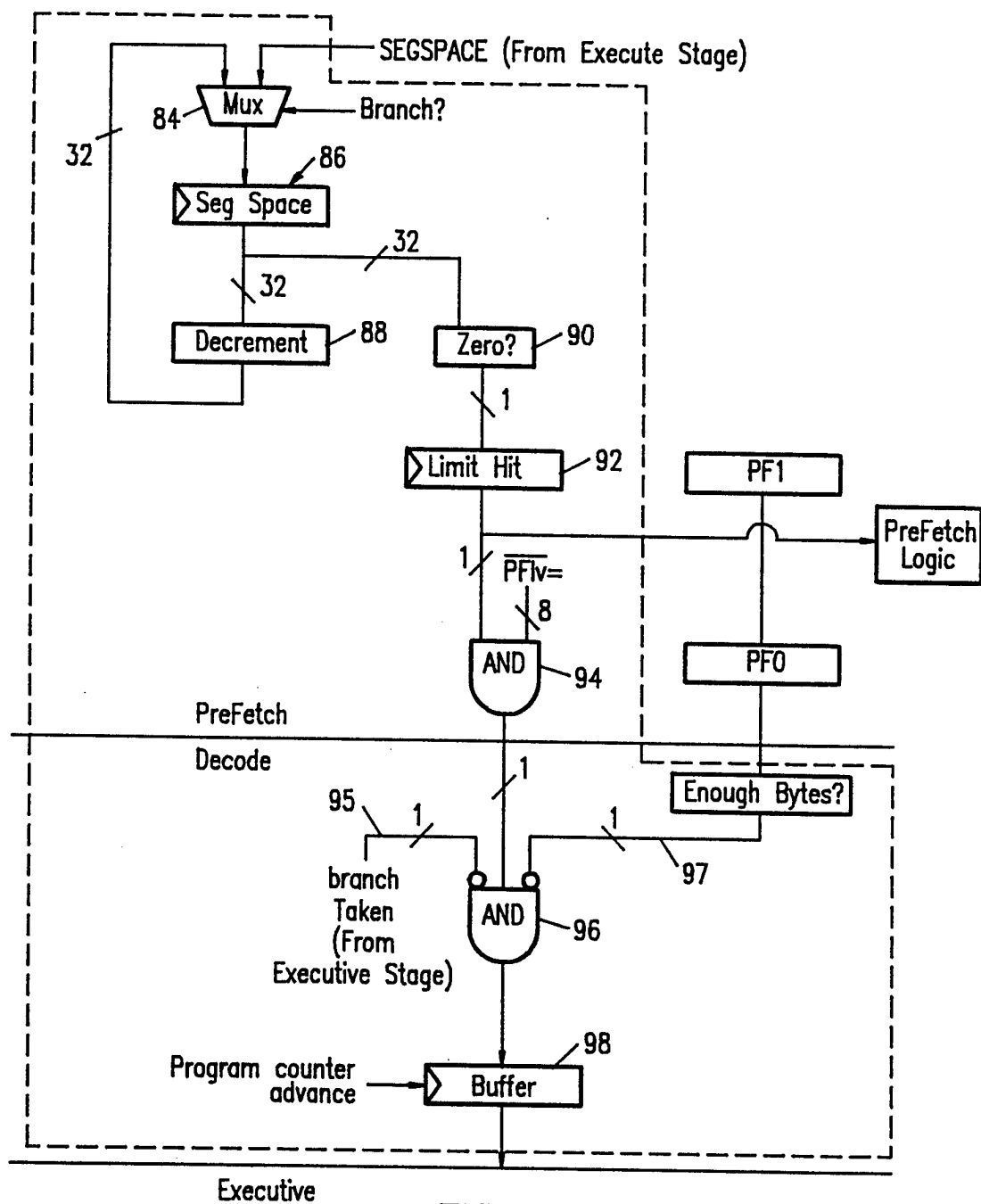


FIG. 3

4/4

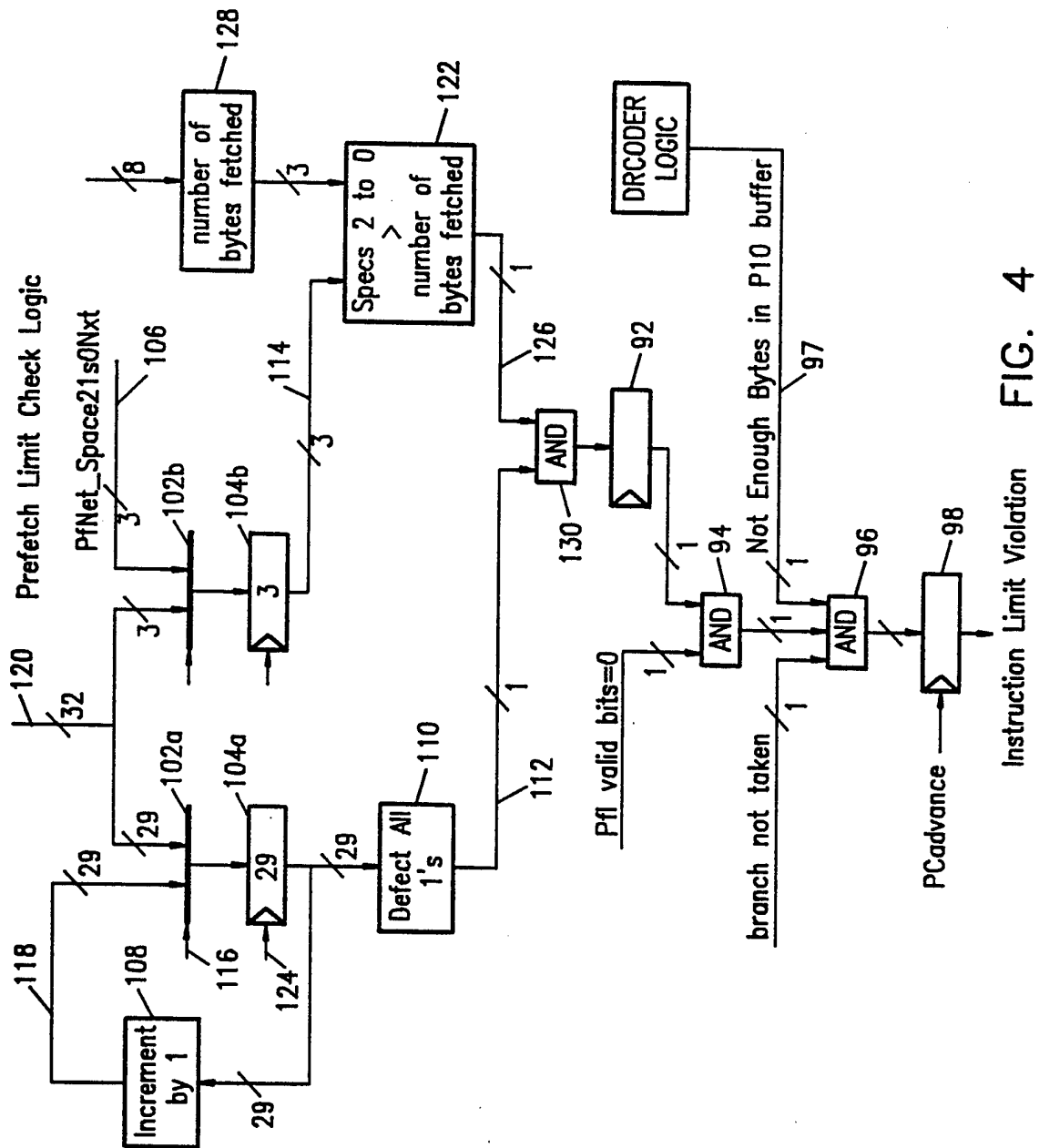


FIG. 4

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 96/06146

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/38

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 32, no. 5A, October 1989, page 109 XP000048854 "CPU IN-PAGE ONLY INSTRUCTION PREFETCHER" see the whole document ---	1
A	EP,A,0 036 093 (TOKYO SHIBAURA ELECTRIC CO) 23 September 1981 see claim 1 ---	1,8,12
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 26, no. 7B, December 1983, NEW YORK US, pages 3941-3944, XP000577315 B. D. HERRMAN ET AL.: "Installing an Instruction-Buffer on a Processor Memory Interface" see page 3944 ---	1

-/--

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

- "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 when the document is taken alone
- "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

Date of the actual completion of the international search

14 August 1996

Date of mailing of the international search report

05.09.96

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+ 31-70) 340-3016

Authorized officer

Daskalakis, T

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 96/06146

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 27, no. 11, April 1985, NEW YORK US, page 6504 XP002010861 "Instruction Prefetch Exception Reporting" -----	2,3,10

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 96/06146

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0036093	23-09-81	JP-A- 56118150	17-09-81
		JP-A- 56127247	05-10-81
		JP-A- 56127248	05-10-81
		DE-A- 3176840	15-09-88
		EP-A- 0139080	02-05-85
		US-A- 4616331	07-10-86
		US-A- 4734849	29-03-88
