PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ^{6} : **G06F 9/00, 9/22, 9/30, 9/38**	**A1**	**(11) International Publication Number:**	**WO 95/22101**
		(43) International Publication Date:	17 August 1995 (17.08.95)

(21) International Application Number: PCT/US95/01705

(22) International Filing Date: 8 February 1995 (08.02.95)

(30) Priority Data:
08/193,296 8 February 1994 (08.02.94) US

(71) Applicant: MERIDIAN SEMICONDUCTOR, INC. [US/US]; Suite 145, 17310 Redhill Avenue, Irvine, CA 92714 (US).

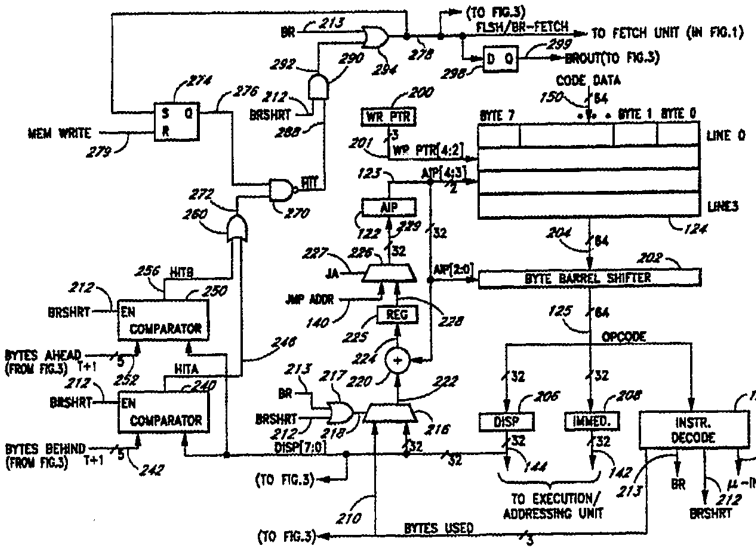
(72) Inventors: KANE, James, A.; 114 Via Orvieto, Newport Beach, CA 92663 (US). WHITTED, Graham, B., III.; 25 Hermosa, Irvine, CA 92720 (US). CHANG, Hsiao-Shih; 1134 North Palo Loma Place, Orange, CA 92669 (US).

(74) Agent: SEWELL, Jerry, T.; Knobbe, Martens, Olson and Bear, 16th floor, 620 Newport Center Drive, Newport Beach, CA 92660 (US).

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

Published
With international search report.

(54) Title: RANDOMLY-ACCESSIBLE INSTRUCTION BUFFER FOR MICROPROCESSOR



(57) Abstract

An instruction buffer circuit performs jumps within an instruction buffer (124), thereby eliminating the need to re-load the instruction buffer when the target instruction is in the instruction buffer. The instruction buffer circuit uses relative offset pointer registers (302, 342) that indicate the number of instruction bytes that fall in front of and behind the read pointer (122) address in the instruction buffer (124). When a jump relative instruction is executed, the relative displacement for performing the jump is compared to the relative offset values to determine whether the target instruction is in the instruction buffer (124). If the target instruction is in the instruction buffer (124), a flush and corresponding re-load of the instruction buffer is inhibited, and the read pointer (122) is bumped to the target instruction. The target instruction is thereby read from the instruction buffer (124) without the delay normally associated with having to re-load the instruction buffer.

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.

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

RANDOMLY ACCESSIBLE INSTRUCTION BUFFER FOR MICROPROCESSOR

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

This invention relates to microprocessors. In particular, this invention relates to instruction queues and buffers that hold instruction data prior to execution by a microprocessor.

DESCRIPTION OF THE RELATED ART

Microprocessors commonly use an instruction buffer to queue instruction data prior to execution. The instruction buffer or "queue" is loaded with lines of instruction data (typically comprising multiple instructions per line) that are fetched either from an external memory or a cache memory. Individual instructions are read or shifted out of the instruction queue for execution on a first-in-first-out basis. The instruction buffer thereby reduces the number of cache reads that are required as instructions are executed in sequential order. Since cache reads typically require at least one clock cycle to perform, a minimum of one clock cycle is avoided when an instruction can be read from the instruction queue. Also, interference with operand accesses is reduced since instructions can be read from the instruction queue while operand (i.e., non-code data) accesses are performed to cache or external memory. The use of an instruction queue can thus significantly increase the performance of a microprocessor.

Two types of instruction queues are commonly used in existing microprocessor designs. The first type is a register-based instruction queue. Register-based instruction queues comprise multiple registers that are connected in a sequential fashion. Instruction data is loaded into the first register of the sequence, and is clocked to the next register in the sequence with successive load operation. Instruction data is thereby shifted in parallel through the instruction queue until the instruction data exits the instruction queue or the queue is flushed.

The second type of prior art instruction queue is a memory based instruction queue. Memory-based queues use a read pointer and a write pointer to address specific memory locations of the queue. Data written to a specific location remains at that location until overwritten (i.e., it is not shifted through the queue). The write pointer is automatically incremented to the next location in the queue when a load is performed, and the read pointer is similarly incremented whenever a read from the queue is performed. The read pointer and the write pointer loop back to the beginning of the queue (i.e., address zero) when incremented beyond the highest queue address. Data is thereby written to and read from the queue on a first-in-first-out basis.

Since existing register-based and memory-based instruction queue designs only allow instruction data to be accessed only on a first-in-first-out basis, they permit a cache access to be avoided only when instructions are executed in sequential order. When a program branch occurs, the instruction queue is flushed and a code-fetch from the cache or the external memory is performed to re-load the instruction queue. The operation of the execution unit of the microprocessor is thus temporarily suspended whenever a program branch occurs.

Forward and backward program branches to instructions that are relatively close (i.e., within approximately 6 instructions) to the branch instruction tend to occur at a high rate at the machine language level. Thus, depending upon the size of the instruction queue, it is not uncommon to have a program branch to an instruction that is currently in the instruction queue. A significant increase in performance can thus be achieved by providing a mechanism for performing a jump to a target instruction within the instruction queue. The present invention relates to an instruction buffer circuit and method that overcomes the above-described limitation in the prior art. Throughout the description that follows, the term "instruction buffer" will refer to a physical memory array or set of sequentially-connected registers in which fetched instruction data is stored (with the term "buffer" being used rather than "queue" to indicate that instruction data can be accessed out-of-order). The term "instruction buffer circuit" will refer to an instruction buffer in combination with control circuitry used for accessing the instruction buffer. The term "branch instruction" will refer to any type of macroinstruction that may pass program control to an instruction address that does not immediately follow the subject instruction. Branch instructions include, for example, subroutine calls, conditional jump instructions, and unconditional jump instructions. "Target branch address" will refer to the program address to which control is passed when a branch is taken. "Target instruction" will refer to the instruction that is the target of a branch (i.e., the instruction at the target branch address).

SUMMARY OF THE INVENTION

The present invention relates to an instruction buffer circuit that can perform a branch to a target instruction in an instruction buffer. The instruction buffer circuit thereby allows program execution to continue without flushing the instruction buffer, and without performing a code-fetch to re-load the instruction buffer.

The instruction buffer circuit comprises an instruction buffer. The instruction buffer is preferably in the form of either an addressable memory array or a plurality of sequentially-connected registers. The instruction buffer circuit further comprises a read pointer for selecting a location of the instruction buffer from which to read code data for execution. The instruction buffer circuit further comprises two relative offset circuits. The first relative offset circuit generates a first relative offset value that indicates the number of valid instruction bytes that are currently in the instruction buffer that fall sequentially ahead of the read pointer address. The second relative offset circuit generates a second relative offset value that indicates the number of valid instruction bytes that are currently in the instruction buffer that fall sequentially behind the read pointer address. The instruction buffer circuit further comprises a compare circuit that compares a relative displacement for a relative jump instruction to the first and second relative offset values, to thereby determine whether the target instruction can be read from the instruction buffer.

During sequential program execution (i.e., when no branches are taken) the read pointer is incremented as instruction bytes are read from the instruction buffer for execution, and the instruction buffer acts as a first-in-first-out buffer.

When a relative branch instruction (i.e., an instruction that uses a relative displacement to specify the target address) that causes a branch is decoded, the compare circuit compares the relative displacement for the jump instruction to the first and second relative offset values. If the relative displacement is positive, indicating a forward jump in memory, the relative displacement is compared with the first relative offset value, which indicates the number of instruction bytes ahead of the read pointer address. If the positive relative displacement is less than or equal to the first relative offset value, indicating that the target instruction is in the instruction buffer, a flush and corresponding re-load of the instruction buffer is inhibited, and the read-pointer is "bumped" to the instruction buffer location that contains the target instruction. On the following clock cycle, the target instruction is read from the instruction buffer and decoded. Thus, the delay normally associated with having to re-load the instruction buffer is eliminated.

If the relative displacement is negative, indicating a backward jump in memory, the relative displacement is compared with the second relative value, which indicates the number of instruction bytes that fall behind the read pointer address. If the negative relative displacement is less than or equal in magnitude to the second relative offset value, indicating that the target instruction is in the instruction buffer, a flush and corresponding re-load of the instruction buffer is similarly inhibited, and the read-pointer is "bumped" to the instruction buffer location that contains the target instruction. On the following clock cycle, the target instruction is read from the instruction buffer and decoded.

To permit self-modifying code, the compare circuit is temporarily disabled following a write to memory, to thereby ensure that the following jump instruction will cause a code-fetch to be performed. This ensures that an unmodified version of a modified instruction will not be executed from the instruction buffer.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a simplified block diagram that illustrates an exemplary embodiment of a pipelined microprocessor. The microprocessor shown will be used to describe an instruction buffer circuit in accordance with the present invention.

Fig. 2 is a block diagram of one embodiment of an instruction buffer circuit in accordance with the present invention.

Fig. 3 is a block diagram of a circuit for generating a BYTES AHEAD relative offset value and a BYTES BEHIND relative offset value for the circuit of Fig. 2.

Fig. 4 is a block diagram of a second embodiment of an instruction buffer circuit in accordance with the present invention.

Fig. 5 is a block diagram of a circuit for generating a DWORDS AHEAD relative offset value and a DWORDS BEHIND relative offset value for the circuit of Fig. 4.

In the drawings, like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit of a reference number identifies the drawing in which the reference number first appears.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Fig. 1 is a high-level block diagram of a pipelined microprocessor 100 that is connected to an external memory 170. The microprocessor 100 shown is an exemplary embodiment of a microprocessor to which the present invention may be applied, and will be used to describe a preferred embodiment of an instruction buffer circuit in accordance with the present invention. It should be understood that the present invention is equally applicable to microprocessors other than the one that will be described herein. Specific widths of busses of the microprocessor 100 are indicated in Fig. 1 where helpful to understanding the preferred embodiment of the instruction buffer circuit that will be described.

Referring to Fig. 1, the microprocessor 100 includes an execution/addressing unit 110, an instruction control unit (ICU) 120, and a cache/bus unit 130. The ICU 120 has an instruction buffer 124, a 32-bit advanced instruction pointer (AIP) register 122, an instruction decode circuit 126, and a fetch unit 128. The AIP register 122 is connected to the instruction buffer 124 by a bus 123. The instruction buffer 124 is connected to the instruction decode circuit 126 by a bus 125. The cache/bus unit 130 has a cache memory (cache) 132.

The execution/addressing unit 110 is connected to the ICU 120 by a jump-address (JMP ADDR) bus 140, a 32-bit immediate operand bus 142, a displacement (DISP) bus 144, and a micro-instruction (μ-INSTRUCTION) bus 146. The execution/addressing unit 110 is connected to the bus/cache unit 130 by a data bus 148 and a 32-bit address bus 152. The ICU 120 is connected to the bus/cache unit 130 by the microinstruction bus 146, a 64-bit code data bus 130, and two data available (DAV) lines 151. The bus/cache unit 130 is connected to an external memory 170 by an address/control (ADDR/CNTRL) bus 160 and a data bus 162.

1. General Operation of Microprocessor

The general operation of the microprocessor 100 will now be described. This description will provide the necessary foundation for describing the preferred embodiment of the instruction buffer 124.

Referring to Fig. 1, on memory access command cycles the execution/addressing unit 110 generates an address for performing either a code-fetch (i.e., a read of instruction data) or an operand access (i.e., a read or write of operand data). The address generated by the execution/addressing unit 110 is provided to the bus/cache unit 130 on the 32-bit address bus 152. The ICU 120 provides a corresponding memory access command (i.e., a fetch request or an operand access request) to the bus/cache unit 130 on the microinstruction bus 146. The memory access command is in the form of a microinstruction field that specifies the access type (i.e., code-fetch, operand read, operand write), and certain parameters for performing the access. The bus/cache unit 130 performs the memory access command by accessing the cache 132 and/or performing an access on the external busses 160, 162. For operand accesses, operand data is passed between the bus/cache unit 130 and the execution/addressing unit 110 on the data bus 148. For code-fetches (hereinafter "fetches"), the bus/cache unit 130 returns the requested instruction data on the code data bus 150.

During each transfer cycle in which code data is returned to the ICU 120, the bus/cache unit 130 places either 4 bytes or 8 bytes of code data on the code data bus 150. The data available (DAV) lines 151 indicate on each clock cycle whether 0, 4 or 8 bytes of code data are being transferred, with a high value on DAV[1] indicating a valid 4-byte value on the upper 32 bits of the code data bus 150, and high value on DAV[0] indicating a valid 4-byte value on the lower 32 bits of the code data bus 150.

Code data returned on the bus 150 is placed in the instruction buffer 124. The instruction buffer 124 holds the code data until it is either overwritten or the instruction buffer 124 is flushed. The least significant five bits of the AIP (advanced instruction pointer) 122 are used as a read pointer for reading instruction data from the instruction buffer 124, as will be described in detail with reference to Fig. 2. Instructions referenced by the AIP 122 are read from the instruction buffer 124 and are passed to the instruction decode circuit 126. The instruction decode circuit 126 decodes individual instructions and generates microinstructions on the bus 146.

The microprocessor 100 has a variable-length instruction format, which includes an opcode of either 1 or 2 bytes, an immediate operand data field of 0, 1, 2 or 4 bytes, and a displacement field of 0, 1, 2 or 4 bytes. Circuitry (Fig. 2) of the ICU 120 extracts any immediate operand data and displacement data included within each instruction. Immediate operand data is passed to the execution/addressing unit 110 on the bus 142. Displacement data is passed to the execution/addressing unit 110 on the displacement (DISP) bus 144. The displacement values are used by the execution/addressing unit 110 to generate addresses on the bus 152.

Of particular importance to the present invention is a type of instruction that will hereinafter be referred to as a "branch short relative" instruction. Branch short relative instructions are characterized by a one-byte opcode and a one-byte displacement field (DISP[7:0]). This displacement field specifies a relative displacement value that can range from -128 to +127 (with DISP[7] acting as a sign bit and negative numbers being represented in standard two's complement format). This relative displacement value specifies the target branch address relative to the next sequential instruction. For example, a jump (JMP) short relative instruction with DISP[7:0] = 15_{10} specifies a jump forward in memory by 15 byte locations relative to the next sequential instruction.

A variety of branch-short-relative-type instructions are included within the instruction set of the microprocessor 100. For example, the microprocessor 100 has a JMP short relative absolute instruction and a jump short relative conditional instruction. The microprocessor 100 also has a loop short relative instruction that causes the microprocessor 100 to decrement a count and then perform a short jump if certain conditions are met.

In the preferred embodiment of the microprocessor 100, the branch short relative instruction is the only type of branch instruction that can potentially reference a target instruction that is in the instruction buffer 124.

The fetch unit 128 controls the generation of fetch requests. Whenever a fetch request is generated, the ICU 120 provides a displacement value on the bus 144. The execution/addressing unit 110 uses this displacement value to generate a fetch address on the bus 152.

The fetch unit 128 generates two types of fetch requests. The first type is a pre-fetch request, wherein the next sequential 16-byte line of code in the memory 170 (relative to the line of code currently being executed) is requested. The second type is a branch fetch request. A branch fetch request may be generated if the microprocessor 100 takes a branch (as the result of a call, interrupt, conditional jump instruction, unconditional jump instruction, loop instruction, etc.) that requires execution to begin at a new program address.

The bus/cache unit 130 is designed to perform all pre-fetches as 16-byte aligned reads from the memory 170 (or the cache 132). Thus, for example, if the execution/addressing unit 110 issues a pre-fetch address of 00005553_{16} on the bus 152, the bus/cache unit 130 will return the 16 instruction bytes from 00005550_{16} to $0000555F_{16}$ in the memory 170. The bus/cache unit 130 is designed to perform branch fetches in a slightly different manner. If the branch fetch address (i.e., the target branch address) falls in the first doubleword (i.e. four bytes) of a 16-byte line, the bus/cache unit 130 returns the entire 16-byte line. If the branch fetch address falls in the second doubleword of a 16-byte line, the bus/cache unit 130 returns the second, third and fourth doublewords of the line. If the branch fetch address falls in the third doubleword of a 16-byte line, the bus/cache unit 130 returns the third and fourth doublewords of the line. And if the branch fetch address falls in the fourth doubleword of a 16-byte line, the bus/cache unit 130 returns only the fourth doubleword of the line.

As will be described in detail, tne present invention inhibits the generation of a branch fetch request on a branch short relative instruction if the target instruction can be read from the instruction buffer 124.

2. Overview of Present Invention

As described above, prior art microprocessors flush the instruction buffer and generate a branch fetch request whenever a program branch is taken, even if the target instruction is in the instruction buffer 124 at the time the branch is taken. Program execution is thereby temporarily suspended while the new fetch data is read from the memory or cache of the system. The present invention solves this problem by including a mechanism to inhibit the generation of an instruction buffer flush and a corresponding branch fetch request when a target instruction can be read from the instruction buffer 124.

Referring to Fig. 1, the microprocessor 100 handles relative branch instructions as follows. During a decode clock cycle for a branch instruction, the instruction decode circuit 126 determines whether a branch will be taken as a result of the branch instruction. If the decode circuit 126 determines that a branch will be taken, the AIP 122 is incremented or decremented by adding the displacement (DISP) value for the instruction to the AIP 122. The five least significant bits of the AIP register 122 (which serve as the read pointer) are thereby "bumped" to point to the instruction buffer 124 location from which the target instruction will be read.

If the branch instruction is a short relative instruction, a comparison circuit (Figs. 2 and 4) determines whether the target instruction can be read from the instruction buffer 124 on the following clock cycle. If so, a "hit" signal is generated which inhibits the generation of a flush and corresponding branch fetch. The target instruction is then read from the instruction buffer 124 on the following clock cycle, and execution continues without delay. The delay heretofore associated with having to re-load the instruction buffer 124 in this event is thereby avoided.

If the target instruction for a branch short relative instruction cannot be read from the instruction buffer 124 on the following cycle, or if the relative branch instruction decoded is not a branch short relative instruction (i.e., does not use a 1 byte relative displacement), the ICU 120 flushes the instruction buffer 124 and generates a branch fetch request. The ICU 120 also provides a displacement value to the execution/addressing unit 110 on the displacement bus 144. The execution/addressing unit 110 uses the displacement value to calculate the branch fetch address for performing the branch fetch. The bus/cache unit 130 uses the branch fetch address to perform a fetch to re-load the instruction buffer 124. Once the instruction buffer 124 is re-loaded with the target instruction, execution resumes.

For certain types of program branches, the branch address, or a value used to calculate the branch address, is read from the memory 170 (or cache 132). "Return" and "jump indirect" are examples of instructions that require such a memory access. For this type of branch instruction the execution/addressing unit 110 provides the jump address to the ICU 120 on the jump address (JMP ADDR) bus 140, and the ICU 120 loads the jump address into the AIP 122. A flush and corresponding branch fetch request are always generated for this type of branch instruction. A flush and branch fetch are also generated whenever an absolute jump instruction is executed.

Since the preferred embodiment of the microprocessor 100 permits self-modifying code, the ability of the instruction buffer circuit to effect a jump within the instruction buffer 124 is temporarily disabled following a memory write operation and until the next flush of the instruction buffer 124 occurs. This ensures that the instruction buffer 124 will be flushed and re-loaded upon the first branch that follows a memory write.

3. Description of the Instruction Buffer Circuit

Figs. 2 and 3 illustrate a preferred embodiment of the instruction buffer circuit. The circuit shown in Fig. 2 will initially be described.

Referring to Fig. 2, the instruction buffer 124 is in the form of a 32-byte addressable memory array arranged as four lines of eight bytes each. As will be described with reference to Fig. 4, the instruction buffer 124 may alternatively be in the form of a plurality of registers sequentially connected in parallel. The code data bus 150 is connected as a data input to the instruction buffer 124. The output of a 5-bit write pointer (WR PTR) register 200 is connected as a first address input to the instruction buffer 124 by a three-bit bus 201. The three lines of the bus 201 correspond to WR PTR[4:2] (i.e., the three most significant bits of the write pointer 200). A bus 123 is connected to the output lines of the AIP register 122. The bit lines AIP[4:3] on the bus 123 are connected as a second address input to the instruction buffer 124.

The data output of the instruction buffer 124 is connected to a byte barrel shifter 202 by a 64-bit bus 204. The byte barrel shift 202 rotates the data appearing on the bus 204 to the right by N bytes, where N may range from 0 to 7. The number N is specified by the bits lines AIP[2:0] of the bus 123. As will be recognized by one skilled in the art, the byte barrel shifter 202 can be implemented using a combination of multiplexers.

The output of the byte barrel shifter 202 is connected by the 64-bit bus 125 to a displacement (DISP) register 206, an immediate (IMMED) register 208, and the instruction decode circuit 126. The outputs of the DISP register 206 and the IMMED register 208 are connected to the execution/addressing unit 110

(Fig. 1) by the busses 144 and 142 respectively. The instruction decode circuit 126 has four outputs that are shown. The first output is the microinstruction bus 146 of Fig. 1. The second output is a branch short (BRSHRT) signal line 212 which becomes active during a decode cycle for a branch short relative instruction if a branch will be taken. If the branch short relative instruction falls across a line boundary of the instruction buffer 124, the BRSHRT signal line 212 becomes active during the second of two decode cycles. The third output is a branch (BR) output line 213 that goes high whenever a branch occurs as the result of something other than a branch short relative instruction. For example, the BR line 213 will go high upon the decode of a branch that results from an absolute jump instruction, a call instruction, a return instruction, or a relative jump instruction that uses a displacement value of two or more bytes. The BR line 213 thus goes high only for branches to target instructions that cannot (or are highly unlikely to) be found in the instruction buffer 124. The fourth output is a BYTES USED bus 210, that indicates the number of instruction bytes used during the current clock cycle.

The BYTES USED bus 210 is connected as a first data input to a multiplexer 216. The displacement (DISP) bus 144 is connected as a second data input to the multiplexer 216. The multiplexer 216 has a control input that is connected to the output of an OR gate 217 by a line 218. The OR gate 217 has a first input connected to the BRSHRT line 212 and a second input connected to the BR line 213. The output of the multiplexer 216 is connected as a first input to a binary adder 220 by an 8-bit bus 222. The bus 123 is connected as the second input to the adder 220. The output of the adder 220 is connected as an input to a register 225 by a bus 224. The output of the register 225 is connected as a first data input to a multiplexer 226 by a bus 228. The jump address (JMP ADDR) bus 140 (Fig. 1) is connected as a second data input to the multiplexer 226. The multiplexer 226 has a control input that is connected to a jump address (JA) signal line 227 that becomes active when the execution/addressing unit 110 provides a jump address to the ICU 120 on the JMP ADDR bus 140. The output of the multiplexer 226 is connected as a data input to the AIP register 122 by a bus 229.

The lines DISP[7:0] of the bus 144 are connected as a first data input to a comparator 240. The second data input to the comparator 240 is connected to a BYTES BEHIND circuit (Fig. 3) by a 5-bit bus 242. The BYTES BEHIND circuit provides a relative offset value BYTES BEHIND_{T+1} on the bus 242. As will be described in greater detail with reference to Fig. 3, the value BYTES BEHIND is equal to the number of valid instruction bytes in the instruction buffer 124 that currently fall behind the instruction byte referenced by AIP[4:0]. The value BYTES BEHIND_{T+1} is thus equal to the number of valid bytes that will exist behind AIP[4:0] on the following clock cycle T+1. The comparator 240 has an enable input that is connected to the BRSHRT line 212.

The lines DISP[7:0] of the bus 144 are also connected as a first data input to a comparator 250. The second data input to the comparator 250 is connected to a BYTES AHEAD circuit (Fig. 3) by a 5-bit bus

252. The bus 252 specifies a value BYTES AHEAD_{T+1}, which is equal to the number of valid instruction bytes that will fall ahead of AIP[4:0] in the instruction buffer 124 on the following clock cycle T+1. The comparator 250 has an enable input that is connected to the BRSHRT line 212.

The output of the comparator 240 is connected as a first input to an OR gate 260 by a HITA signal line 246. The output of the comparator 250 is connected as a second input to the OR gate 260 by a HITB signal line 256. The output of the OR gate 260 is connected as a first input to a NAND gate 270 by a line 272. The second input to the NAND gate 270 is connected to the output (Q) of an R-S flip-flop 274 by a line 276. The set (S) input of the R-S flip-flop 274 is connected to the output of an OR gate 291 by a flush/branch-fetch (FLSH/BR-FETCH) signal line 278. The reset (R) input of the flip-flop 274 is connected to a MEM WRITE signal line 279 that goes high whenever a write to the memory 170 (Fig. 1) and/or the cache 132 occurs.

The output of the NAND gate 270 is connected as a first input to an AND gate 290 by a $\overline{\text{HIT}}$ signal line 288. The BRSHRT signal line 212 is connected as the second input to the AND gate 290. The output of the AND gate 290 is connected as a first input to an OR gate 294 by a line 292. The BR line 213 (from the instruction decode circuit 126) is connected as a second input to the OR gate 294. The output of the OR gate is connected to the FLSH/BR-FETCH signal line 278. The FLSH/BR-FETCH signal line 278 is connected to the fetch unit 128 (Fig. 1). The FLSH/BR-FETCH signal line 278 is also connected to the circuit shown in Fig. 3. The FLSH/BR-FETCH signal line 278 is also connected as an input to a flip-flop 298. The output of the flip-flop 298 is connected to a BROUT signal line 299.

The operation of the circuit of Fig. 2 will now be described. As noted above, code data is provided to the instruction buffer 124 on the code data bus 150 either four bytes at-a-time or eight bytes at-a-time. Thus, on a given clock cycle, 0, 4 or 8 bytes may be loaded into the instruction buffer 124. Instructions fetched from byte locations in the memory 170 are loaded into corresponding byte locations of the instruction buffer 124, with the five least significant bits of the memory 170 address specifying the instruction buffer 124 location. Instructions are loaded with the opcode byte (or bytes) falling at the lowest byte address in the instruction buffer 124. For example, a three-byte instruction loaded into the BYTE0, BYTE1 and BYTE2 locations in LINE0 will have an opcode at the BYTE0 location of LINE0. Instructions can fall on any byte boundary in the instruction buffer 124, and can fall across one or more of the 8-byte lines LINE0-LINE3.

The write pointer 200 (WR PTR) is a 5-bit register that provides a write address for loading the instruction buffer 124. Loads are performed using WR PTR[4:2] (i.e., write pointer bits 4, 3 and 2) as the write address. All loads of the instruction buffer 124 are thus performed on four-byte boundaries (i.e., all loads start at either BYTE0 or BYTE4 of one of the four lines LINE0-LINE3). As loads are performed, the write pointer 200 is incremented by the number of bytes being loaded (either four or eight), as indicated by the DAV lines 151 (Fig. 1). The write pointer 200 automatically loops back to LINE0 when the 5-bit value is incremented beyond its maximum value of 31_{10}. Data written to the instruction buffer 124 may be read out until either the data is overwritten or the instruction buffer 124 is flushed. If a branch occurs that causes the instruction buffer 124 to be flushed, the write pointer 200 is automatically loaded with the five least significant bits of the target branch address. The circuitry for incrementing and loading the write pointer is omitted to simplify the figure.

The five least significant bits of the 32-bit AIP register 122 are used as a read pointer for reading instructions from the instruction buffer 124. All reads are performed as 8-byte aligned accesses, with the bits AIP[4:3] used to address one of the four 8-byte lines LINE0-LINE3. Whenever a read is performed, the 8 bytes of instruction data read from the addressed line are passed through the byte barrel shifter 202. The byte barrel shifter 202 rotates the 8 bytes of instruction data such that the opcode (if any) of the next instruction to be executed falls in the right-most (i.e., least significant) byte position on the bus 125. The byte barrel shifter 202 thereby aligns the instructions so that the opcode, immediate, and displacement fields can be extracted. Displacement and immediate fields, if any, are loaded into the registers 206 and 208 respectively. In the preferred embodiment, 32-bit shifters (not shown) are also used to align the displacement and immediate values before the values are loaded into the registers 206, 208. Opcodes (and other fields that require decoding) are passed to the instruction decode circuit 126.

On every clock cycle the instruction decode circuit 126 generates a BYTES USED value on the bus 210. The BYTES USED value may range from 0 to 8, and indicates the number of bytes of the 8-byte line that are used during the current clock cycle. For example, if the next instruction to be executed is five bytes long, and the first two instruction bytes fall in the line currently referenced by AIP[4:3], BYTES USED will be 2 for the current clock cycle.

4. Generation of Next AIP Value

During each clock cycle the circuit of Fig. 2 generates a 32-bit value AIP_{T+1} that will be clocked into the AIP register 122 at the end of the clock cycle as the next AIP value. The method used to generate this value depends upon the particular type of instruction decoded (if any) by the instruction decode circuit 126.

On clock cycles for which both the BRSHRT signal line 212 and the BR signal line 213 are low (indicating that no branches will be taken on the immediately-following execution cycle), the multiplexer control line 218 is low, causing the multiplexer 216 to select the BYTES USED bus 210. The BYTES USED value selected by the multiplexer 216 is added to the current AIP value on the bus 123 by the adder 220. The output of the adder 220 is clocked into the register 225 (clock not shown). The jump address (JA) line 227 will be low on the following clock cycle, causing the multiplexer 226 to select the output of the register 225 on the bus 228 as the next AIP value. Thus, during sequential program execution (i.e., execution without program branches), $AIP_{T+1} = AIP_T + BYTES\ USED_T$. Note that when the read pointer bits AIP[4:0] exceed the maximum instruction buffer address of 31_{10}, the read pointer automatically loops back to address zero of the instruction buffer 124, causing LINE0 to be read on the following decode cycle. Thus, the instruction buffer 124 acts as a first-in-first-out (FIFO) buffer (i.e., a queue) during sequential program execution.

When a branch instruction that causes a branch is decoded, the instruction decode circuit 126 asserts either the BRSHRT signal line 212 or the BR signal line 213, depending upon the type branch instruction decoded. If the branch instruction is a relative branch instruction, the DISP bus 144 specifies the relative displacement value for performing the jump. If the branch instruction is an absolute jump instruction, the DISP bus 144 specifies the absolute target branch address to which the jump will be performed.

The high value on either the BRSHRT line 212 or the BR line 213 causes the multiplexer select line 218 to go high, thereby causing the multiplexer 216 to select the DISP bus 144. If the branch instruction is a relative (long or short) branch instruction, the adder 220 adds the displacement value on the DISP bus 144 to the current AIP value on the bus 123 to generate the next AIP value. If the branch instruction is an absolute branch instruction, the adder 220 passes the displacement value appearing on the bus 222 through to its output without adding it to the current AIP value on the bus 123 (circuitry for implementing pass-through function of the adder 220 not shown). In either case, the output of the adder 220 is clocked into the register 225.

On the following clock cycle the multiplexer 226 selects either the output of the register 225 or the value (if any) on the JMP ADDR bus 140. If a jump address is being provided by the execution/addressing unit 110 on the current clock cycle, the jump address (JA) line 227 will be high, causing the multiplexer 226 to select the JMP ADDR bus 140 as the source of the next AIP value. Otherwise, the multiplexer 226 will select the output of the register 225 as the next AIP value.

The five least significant bits of the value loaded into the AIP register 122 following a branch specify the instruction buffer 124 address where the target instruction will be read from. If the branch is to an instruction not currently in the instruction buffer 124, the instruction buffer 124 must be flushed and re-loaded before the target instruction can be read from the instruction buffer and executed. If the branch is a relative short branch to an instruction that can be read from the instruction buffer 124 on the following clock cycle (i.e., the target instruction is or will be in the instruction buffer 124, and the internal jump mechanism is currently enabled), the 8-byte line containing the target instruction is read from the instruction buffer 124 on the following clock cycle and the target instruction is executed.

5. Comparison Circuit

Whenever a branch short relative instruction is decoded, a comparison circuit (comprising the comparators 240 and 250 and the OR gate 260) is used to determine whether or not the target instruction can be read from the instruction buffer 124. The comparators 240 and 250 compare DISP[7:0] to the BYTES BEHIND and BYTES AHEAD values respectively. Since it must be determined whether or not the target instruction will be in the instruction buffer 124 during the following clock cycle (i.e., the clock cycle following the decode of the branch short relative instruction), the comparators 240 and 250 compare DISP[7:0] to BYTES BEHIND_{T+1} and BYTES AHEAD_{T+1}. The comparisons thus take into account any code data that is loaded into the instruction buffer 124 during the current clock cycle (which may include the target instruction, or may overwrite the target instruction in the instruction buffer 124).

The comparator 240 generates a signal HITA on the line 246 according to the following logic equation:

$$\text{HITA} = \text{BRSHRT and } \overline{\text{DISP[7]}} \text{ and } |\text{DISP[7:0]}| \le \text{BYTES BEHIND}_{T+1}$$

Thus, the HITA line 246 becomes active if a branch short relative instruction is decoded that has a negative relative displacement value that is less than or equal (in magnitude) to BYTES BEHIND_{T+1}. Since BYTES BEHIND_{T+1} is equal to the number of valid bytes that will be in the instruction buffer 124 on the following cycle that will fall behind the address AIP[4:0]_{T+1}, the HITA signal line 246 will go high if a backward branch is taken to a target instruction that can be read from the instruction buffer 124.

The comparator 250 generates a logic signal HITB on the line 256 according to the following logic equation:

$$\text{HITB} = \text{BRSHRT and DISP[7] and (DISP[7:0]} \le \text{BYTES AHEAD}_{T+1})$$

Thus, the HITB line 256 becomes active if a branch short relative instruction is decoded that has a positive relative displacement value that is less than or equal to BYTES AHEAD_{T+1}. Since BYTES AHEAD_{T+1} is equal to the number of valid bytes that will be in the instruction buffer 124 on the following cycle that will fall ahead of the address AIP[4:0]_{T+1}, the HITB signal line 256 will go high if a forward branch is taken to a target instruction that can be read from the instruction buffer 124.

Assuming for purposes of illustration that the line 276 is currently high, the $\overline{\text{HIT}}$ signal line 288 will go low if either the HITA or the HITB signal line goes high. Thus, the $\overline{\text{HIT}}$ signal line 288 goes low only if a relative short jump is taken to an instruction that can be read from the instruction buffer 124.

The output of the AND gate 290 on the line 292 goes high on clock cycles for which a jump short relative instruction that causes a branch is decoded and no hit occurs. The FLSH/BR-FETCH signal line 278 thus goes high if either a non-branch-short-relative branch is decoded, or a branch short relative branch is decoded to a target instruction that cannot be read from the instruction buffer 124. Thus, the FLSH/BR-FETCH signal line 278 goes high whenever a branch is taken to a target instruction that cannot be read from the instruction buffer 124. The FLSH/BR-FETCH signal line 278 is connected to the fetch unit 128 (Fig. 1), and initiates a branch fetch to re-load the instruction buffer 124 when high. A high value on the FLSH/BR-FETCH signal line 278 also causes the BYTES AHEAD and BYTES BEHIND offset values to be reset, as discussed below with reference to Fig. 3. A high value on the FLSH/BR-FETCH signal line 278 also generates a flush of the instruction buffer 124.

The FLSH/BR-FETCH signal on the line 278 is delayed by one clock cycle by the flip-flop 298 (clock not shown) to produce the branch outside (BROUT) signal on the line 299. The BROUT signal line thus goes high on the execution cycle for any branch instruction that causes a branch outside the instruction buffer 124. As will be discussed with reference to Fig. 3, the BROUT signal is used for generating the BYTES AHEAD and BYTES BEHIND relative offsets.

The operation of the circuit comprising the R-S flip-flop 274 and the NAND gate 270 will now be described. This circuit has the purpose of preventing branches within the instruction buffer 124 if a memory write has been performed and a flush of the instruction buffer 124 has not been performed since the memory write. The circuit thereby allows the microprocessor 100 to implement code modification, wherein writes are performed to the memory 170 (Fig. 1) to modify individual instructions.

Referring to Fig. 2, when a memory write is performed, the MEM WRITE signal line 279 goes high, causing the output line 276 of the R-S flip-flop to go low. The low value on the line 276 masks any HIT signals appearing on the line 272, thereby preventing a jump within the instruction buffer 124. This guarantees that an instruction modified by the write operation will be executed out of the cache 132 (or the memory 170), and thus eliminates the possibility that the unmodified version of a modified instruction will be executed from the instruction buffer 124. (Note: the preferred embodiment of the microprocessor 100 requires that a jump be performed before a modified instruction is executed).

The first jump instruction to follow the memory write causes the FLSH/BR-FETCH line 278 to go high, even if the target instruction is in the instruction buffer 124. The high value on the FLSH/BR-FETCH line 278 causes a flush of the instruction buffer 124, and causes a branch fetch to be generated. The high level on the FLSH/BR-FETCH line 278 also causes the output of the R-S flip flop 274 to go high, to thereby re-enable jumps within the instruction buffer 124.

As will be apparent to one skilled in the art, use of the circuit comprising the R-S flip-flop 274 and the NAND gate 270 is desirable only if the microprocessor to which the present invention is applied supports code modification. It will further be apparent that the MEM WRITE signal line 279 can be appropriately qualified for certain microprocessor designs to reset the flip-flop 274 only upon certain types of write operations that can affect code. For example, since stack operations are not normally used as a means for modifying code, the MEM WRITE signal line 279 can be qualified such that it only becomes active for non-stack memory writes.

6. Circuit for Generating Relative Offset Values

The circuit for generating BYTES AHEAD$_{T+1}$ and BYTES BEHIND$_{T+1}$ will now be described. Referring to Fig. 3, the BYTES AHEAD$_{T+1}$ value is generated as the output of a 3-input adder 300. The output lines 252 of the adder 300 are connected as an input to a register 302. One input of the adder 300 is connected to the output of a multiplexer 304 by a bus 306. The multiplexer 304 has a control input that is connected to the branch outside (BROUT) signal line 299. The multiplexer 304 has a first data input that is connected to a two's complement circuit 310 by a bus 312. The two's complement circuit 310 has an input that is connected to a 5-bit bus 314. Bits 3 and 4 of the bus 314 are tied to zero (i.e., tied low). Bits[2:0] of the bus 314 are connected to bits[2:0] of the bus 229 (Fig. 2), which represent AIP[2:0]$_{T+1}$ (i.e., the three least significant bits of the next AIP value). The multiplexer 304 has a second data input that is connected to the output of the register 302 by the bus 313.

A second input to the adder 300 is connected to the output of a multiplexer 318 by a bus 320. The multiplexer 318 has a control input that is connected to the data available lines DAV[0:1] (Fig. 1). The multiplexer 318 has a first data input of zero on a bus 322 (i.e., all lines of the bus 322 are tied to zero). The multiplexer 318 has a second data input of "4" on a bus 324 (i.e., bits [3:0] = 0100_2 on the bus 324). The multiplexer 318 has a third data input of "8" on a bus 326 (i.e., bits [3:0] = 1000_2 on the bus 326).

A third input of the adder 300 is connected to the output of a two's complement circuit 330 by a bus 332. The input of the two's complement circuit 330 is connected to the output of a multiplexer 334 by a bus 336. The multiplexer 334 has a first data input of zero on a bus 328. The multiplexer 334 has a second data input that is connected to the BYTES USED bus 210 (Fig. 2). The multiplexer 334 has a third data input that is connected to a DISP[7:0]$_{T-1}$ bus 337, which provides a value equal to DISP[7:0] (on the bus 144 of Figs. 1 and 2) from the previous clock cycle T-1 (circuitry for delaying DISP[7:0] not shown). The multiplexer 334 has a first control input that is connected to a $\overline{\text{HIT}}_{T-1}$ signal line 335, which is generated by delaying the $\overline{\text{HIT}}$ signal line 288 (Fig. 2) by one clock cycle (circuitry for delaying the $\overline{\text{HIT}}$ signal not shown). The multiplexer 334 has a second control input connected to the BROUT signal line 299 (Fig. 2).

The BYTES BEHIND$_{T+1}$ value is generated as the output of a 3-input adder 340. The output lines 242 of the adder 340 are connected as an input to a register 342. One input of the adder 340 is connected to the output of the multiplexer 334 by the bus 336. A second input of the adder 340 is connected to the output of a two's complement circuit 346 by a bus 348. The input of the two's complement circuit is connected to the output of a multiplexer 350 by a bus 352. The multiplexer 350 has a first data input of zero on a bus 354, and a second data input of "16" (i.e., 16_{10}) on a bus 356. The multiplexer 350 has a control input that is connected to a PRE-FETCH signal line 358 that goes high for one clock cycle when a pre-fetch is initiated by the bus/cache unit 130 (Fig. 1).

A third input of the adder 340 is connected to the output of a multiplexer 362 by a bus 364. The multiplexer 362 has a first data input that is connected to a 5-bit bus 368. Bit[4] of the bus 368 is tied to zero (i.e., tied low), and bits[3:0] of the bus 368 are connected to the JMP ADDR[3:0] lines of the bus 140 (Fig. 1 and 2). The multiplexer 362 has a second data input that is connected to the output of the register 342 by a bus 370. The multiplexer 362 has a control input that is connected to the BROUT signal line 299.

7. Calculation of BYTES AHEAD Offset

The operation of the circuit used to calculate BYTES AHEAD_{T+1} will now be described. Referring to Fig. 3, the register 302 holds the value BYTES AHEAD_T, which is the BYTES AHEAD value for the current clock cycle. At the end of each clock cycle the output of the adder 300 is clocked into the register 302 (clock not shown) as the new BYTES AHEAD value. Thus, the value on the bus 252 represents the BYTES AHEAD value for the following clock cycle. The adder generates the BYTES AHEAD_{T+1} value by adding the three values on the busses 306, 320 and 332.

Referring to the multiplexer 304, when the BROUT signal line 299 is low the multiplexer 304 selects the feedback path 313. As discussed above, the BROUT signal line 299 will become high only if a branch is taken to a target instruction that cannot be retrieved from the instruction buffer 124. Thus, during all other clock cycles (including execution cycles of branch short relative instructions that cause jumps within the instruction buffer 124), the current BYTES AHEAD value is added in as one of the three components for generating the next BYTES AHEAD value. When the BROUT signal line 299 is high, the multiplexer 304 selects the bus 312, which has a value that is the two's complement of the value $0,0,\text{AIP}[2:0]_{T+1}$. The value $\text{AIP}[2:0]_{T+1}$ is thereby subtracted from the values appearing on the busses 320 and 332.

Referring to the multiplexer 318, the data available lines DAV[1:0] 151 specify the number of instruction bytes being loaded into the instruction buffer 124 on the current clock cycle. When the DAV lines 151 indicate that no instruction bytes are being loaded into the instruction buffer 124 (i.e., DAV[1:0] - 00_2), the multiplexer 318 outputs a zero on the bus 320. When the DAV lines 151 indicate that four instruction bytes are being loaded into the instruction buffer 124 (i.e., DAV[1:0] - 10_2 or DAV[1:0] - 01_2), the multiplexer 318 outputs the value 4 on the bus 320. When the DAV lines 151 indicate that eight instruction bytes are being loaded into the instruction buffer 124 (i.e., DAV[1:0] - 11_2), the multiplexer 318 outputs the value 8 on the bus 320. Thus the value on the bus 320 indicates the number of bytes being loaded into the instruction buffer 124 on the current clock cycle.

Referring to the multiplexer 334, the control lines BROUT 299 and $\overline{\text{HIT}}_{T-1}$ 355 are used to select between the zero bus 328, the BYTES USED bus 210 and the DISP[7:0]$_{T-1}$ bus 337 according to Table 1. The output of the multiplexer 334 is passed through the two's complement circuit 330 to effect a subtraction of the value selected by the multiplexer 334.

BROUT	$\overline{\text{HIT}}_{T-1}$	MUX OUTPUT
0	1	BYTES USED
0	0	DISP[7:0]$_{T-1}$
1	0	(does not occur)
1	1	0

TABLE 1

Referring to Table 1, if the BROUT signal line 299 is low (inactive) during the current clock cycle (indicting that a branch outside the instruction buffer 124 is not currently being executed) and the $\overline{\text{HIT}}_{T-1}$ signal line 335 is high (inactive), indicating that a hit for a branch short relative instruction did not occur during the previous clock cycle, the multiplexer 334 (MUX) selects the BYTES USED bus 210. Thus, when no branches are being taken, the BYTES AHEAD$_{T+1}$ value is generated by subtracting the number of bytes used from the sum of the current BYTES AHEAD value plus the number of bytes being loaded into the instruction buffer 124.

Referring to Table 1, if the BROUT signal line 299 is low (inactive) and the $\overline{\text{HIT}}_{T-1}$ signal line 335 is low (active) during the current clock cycle (indicating that an instruction buffer 124 hit occurred on the previous clock cycle), the DISP[7:0]$_{T-1}$ bus 337 is selected by the multiplexer 334. Thus, the relative displacement DISP[7:0] for the branch short relative instruction being executed is subtracted from the sum of BYTES AHEAD$_T$ and the number of bytes being loaded into the instruction buffer 124 on the current clock cycle. If the relative displacement DISP[7:0] is positive (indicating a forward jump in the instruction buffer 124), the next BYTES AHEAD value decreases as a result of the jump, indicating that fewer instructions will be ahead of the read pointer AIP[4:0] following the jump. If the relative displacement DISP[7:0] is negative (indicating a backward jump in the instruction buffer 124), the next BYTES AHEAD value increases as a result of the jump, indicating that a greater number of instructions will be ahead of the read pointer AIP[4:0] following the jump.

Referring to Table 1, if the BROUT signal line 299 is high (active) and the $\overline{\text{HIT}}_{T-1}$ signal line 335 is high (inactive) during the current clock cycle, indicating that a branch outside the instruction buffer 124 is being executed, the multiplexer 334 outputs a zero. During the same clock cycle the multiplexer 304 outputs the two's complement of $AIP[2:0]_{T+1}$. Thus, in the event of a jump outside the instruction buffer 124, the new BYTES AHEAD value is generated by subtracting $AIP[2:0]_{T+1}$ from the number of instruction bytes being loaded into the instruction buffer 124. If no instruction bytes are being loaded into the instruction buffer 124 on the current clock cycle, the BYTES AHEAD value will initially be negative, and will become positive as the corresponding branch fetch is performed. Thus, for example, if a branch outside the instruction buffer 124 is taken to a jump address of $xxxxxx02_{16}$ (x = "don't care"), and no instruction bytes are loaded during the current cycle, BYTES AHEAD will initially be -2. If 8 bytes of code from the branch fetch are loaded into the instruction buffer 124 on the following clock cycle, the BYTES AHEAD value will be incremented to 6, indicating that six instruction bytes are ahead of the new read pointer value of AIP[4:0] = 00010_2.

Note that in Table 1, the combination 1,0 does not occur because BROUT and $\overline{HIT}_{T-1}$ cannot be active during the same cycle.

8. Calculation of BYTES BEHIND Offset

The BYTES BEHIND relative offset is similarly calculated as the summation of three components. Referring to the multiplexer 362, when the BROUT signal line 299 is low the multiplexer 362 selects the current BYTES BEHIND value on the bus 370. Thus, the current BYTES BEHIND value is used as a first component of the next BYTES BEHIND value if no branch outside the instruction buffer 124 is currently being executed. When the BROUT signal line 299 is high, indicating that a branch outside the instruction buffer 124 is being executed, the multiplexer 362 selects the bus 368.

A second component of the addition is provided as the output of the multiplexer 334 on the bus 336. The value selected by the multiplexer 334 is routed to the adder 340 without being passed through the two's complement circuit 330. Thus, on clock cycles for which BYTES AHEAD is decremented by BYTES USED, BYTES BEHIND is incremented by BYTES USED. On clock cycles for which the relative displacement DISP[7:0] is subtracted BYTES AHEAD, DISP[7:0] is added to BYTES BEHIND.

The third component of the addition is provided as the two's complement of the output of the multiplexer 350. When the pre-fetch line 358 is high, indicating that a pre-fetch is being initiated by the bus/cache unit 130 (Fig. 1), the multiplexer 350 selects the "16" bus 356. Thus, 16_{10} is subtracted from the current BYTES BEHIND value whenever a pre-fetch is initiated, to indicate that 16 bytes of instruction data in the instruction buffer 124 will be overwritten during the subsequent clock cycles as the pre-fetch is performed. When the PRE-FETCH line 358 is low, the zero bus 354 is selected, resulting in a zero on the bus 348.

When the BROUT signal line 299 is high, indicating a branch to a target instruction outside the instruction buffer 124, the PRE-FETCH line 358 will be low (i.e., pre-fetches are blocked when jumps outside the instruction buffer 124 occur). Thus, the next BYTES BEHIND value following the branch will be AIP[3:0]$_{T+1}$. For example, if a jump to a jump address of xxxxxx15$_{16}$ is executed, the new BYTES BEHIND value will be 5, indicating that 5 of the 16 instruction bytes (in the 16-byte line to be fetched from xxxxxx10$_{16}$ to xxxxxx1F$_{16}$) will fall behind the new read pointer value of AIP[4:0] = 15$_{16}$. Since this new BYTES BEHIND value represents the number of bytes that will fall behind the read pointer once the fetch data has been loaded into the instruction buffer 124, logic (not shown) is included to inhibit the BYTES BEHIND comparison until the last clock cycle of the branch fetch.

9. Alternative Embodiments of Instruction Buffer Circuit

An alternative embodiment of the instruction buffer circuit of Fig. 2 will now be described with reference to Fig. 4. This embodiment that will be described illustrates two variations that can be made to the instruction buffer circuit. The first variation involves the use of a register-based instruction buffer in place of the randomly-accessible-memory based instruction buffer 124 of Fig. 2. The second variation, which can be made independently from the first variation, is the replacement of the BYTES AHEAD and BYTES BEHIND relative offset values with relative offset values that indicate the numbers of doublewords (i.e., four-byte values) ahead and behind the instruction currently referenced by the AIP. In describing the embodiment shown in Fig. 4, like reference numbers will be used to refer to elements that are functionally similar to the elements shown in Fig. 2.

Referring to Fig. 4, the instruction buffer 124 is shown as a register-based buffer comprising four 8-byte registers 124a, 124b, 124c and 124d. The code data bus 150 is connected to the register 124a. The register 124a is connected to the register 124b by a bus 400. The register 124b is connected to the register 124c by a bus 402. The register 124c is connected to the register 124d by a bus 404. Each least significant byte of the registers 124a-124d is connected as an input to a first 4:1 multiplexer 408a. The second byte of each of the registers 124a-124d is connected as an input to a second 4:1 multiplexer (not shown). The third byte of each of the registers 124a-124d is connected as an input to a second 4:1 multiplexer (not shown). The fourth byte of each of the registers 124a-124d is connected as an input to a third 4:1 multiplexer (not shown). The fifth byte of each of the registers 124a-124d is connected as an input to a fourth 4:1 multiplexer (not shown). The sixth byte of each of the registers 124a-124d is connected as an input to a fifth 4:1 multiplexer (not shown). The seventh byte of each of the registers 124a-124d is connected as an input to a seventh 4:1 multiplexer (not shown). The eighth byte of each of the registers 124a-124d is connected as an input to an eighth 4:1 multiplexer 408h. The outputs of the eight 4:1 multiplexers (e.g., 408a and 408h) are each connected to 8 of the bit lines of the 64-bit bus 204. The 64-bit bus is connected to the byte barrel shifter 202, as in Fig. 2.

The multiplexers (e.g., 408a and 408h and the other multiplexers, not shown) each have respective control inputs connected to a control logic circuit 409 by respective pairs of select lines (e.g., 410a, 410b and 410h). The control logic circuit 409 has a first input that is connected to a 3-bit DWORDS AHEAD bus 412. The DWORDS AHEAD bus 412 provides a DWORDS (doublewords) AHEAD$_{T+1}$ value that indicates the number of doublewords that will be in the instruction buffer 124 on the following clock cycle that fall ahead of the doubleword selected by the multiplexers (e.g., 408a and 408h). The control logic circuit 409 has a second input that is connected to the AIP[2:0] lines of the bus 123 (Fig. 2). The control logic circuit 409 has a third input that is connected to the DISP[7:0] lines of the bus 144.

The comparator 240 now has a first input that is connected to the DISP[7:2] lines of the DISP bus 144, and a second input that is connected to a DWORDS (doublewords) BEHIND bus 416, that provides a DWORDS BEHIND$_{T+1}$ relative offset value. The DWORDS BEHIND value is equal to the number of doublewords in the instruction buffer 124 that are behind the doubleword selected by the multiplexer 408. Thus, DWORDS BEHIND$_{T+1}$ is the DWORDS BEHIND value for the following clock cycle. The comparator 250 now has a first input that is connected to the DISP[7:2] lines of the bus 144, and a second input connected to the DWORDS AHEAD bus 412. The instruction buffer circuit of Fig. 4 is otherwise substantially identical to the circuit of Fig. 2.

Instruction data loaded into the instruction buffer 124 from the code data bus 150 is loaded into the register 124a. With successive load operations the code data is shifted to the next register (124b, 124c and 124d) in sequence. Code data in the register 124d is overwritten in the instruction buffer 124 with the following load operation. On every clock cycle, the multiplexers (e.g., 408a and 408h) select eight bytes of contiguous code data from the registers 124a-124d as the source of the code data to be decoded during the current clock cycle. The control logic circuit 409 controls each of the multiplexers (e.g., 408a and 408h) based on the DWORDS AHEAD$_{T+1}$, AIP[2:0] and DISP[7:0] values. The control logic circuit 409 keeps track of the current location within the instruction buffer 124 from which instruction data is being read. In the event of a hit, the control logic circuit 409 uses DISP[7:0] value on the bus 144 to determine the registers 124a-124d and the byte locations within the registers from which to perform the next read. The DWORDS AHEADT+1 input on the bus 412 allows the control logic 409 to keep track of the status of the instruction buffer 124 (empty, full, partially full, etc).

The control logic circuit 409 can advantageously be designed to read code data during sequential program execution from the registers 124b and 124c (when possible) once the instruction buffer 124 becomes full to thereby maintain a least two doublewords ahead and two doublewords behind the current point of execution. This assures that a branch forward by 8 bytes or less or a branch backward by 8 bytes or less can be performed without re-loading the instruction buffer 124 (provided that the comparison circuit is currently enabled). The use of the eight separate multiplexers (e.g., 408a and 408h) advantageously allows code data to read from the instruction buffer 124 on any byte boundary.

When a branch short relative instruction is decoded, the comparators 240 and 250 of Fig. 4 compare DISP[7:2] to DWORDS AHEAD$_{T+1}$ and DWORDS BEHIND$_{T+1}$ to generate the HITA and HITB signals on the lines 246 and 256 respectively according to the following equations:

$$\text{HITA} = \text{BRSHRT and } \overline{\text{DISP[7]}} \text{ and } |\text{DISP[7:2]}| \le \text{DWORDS BEHIND}_{T+1}$$

$$\text{HITB} = \text{BRSHRT and DISP[7] and } (\text{DISP[7:2]} \le \text{DWORDS AHEAD}_{T+1})$$

These comparisons advantageously result in a reduction in logic over the comparisons performed for the circuit of Fig. 2. The HITA and HITB signal lines 246 and 256 are used to generate the FLSH/BR-FLUSH signal on the line 278 and the BROUT signal on the line 299 in the same manner described above for Fig. 2.

Fig. 5 illustrates a circuit for generating the DWORDS AHEAD and DWORDS BEHIND values used by the circuit of Fig. 4. The circuit is identical to the circuit of Fig. 3 with the following exceptions. The registers 302 and 342 now hold 3-bit DWORDS AHEAD and DWORDS BEHIND values rather than 5-bit BYTES AHEAD and BYTES BEHIND values. The multiplexer 304 now selects between the output of the register 302 and a zero bus 512. The multiplexer 318 now selects between the zero bus 322, a "one" bus 524 and a "two" bus 526, corresponding to the three possible quantities of doublewords that may be loaded into the instruction buffer 124 during a given clock cycle. The multiplexer 334 now has a data input that is connected to the output of a doubleword boundary-cross logic circuit 515 by a bus 510. The circuit 515 has a first input that is connected to the AIP[2:0] lines of the bus 123, and a second input that is connected to the BYTES USED bus 210. The circuit 515 outputs a value that is equal to the number of doubleword boundaries crossed during the current cycle. For example, for a current AIP value of xxxxxx02$_{16}$ and a BYTES USED value of 5, the circuit 515 will output a "1" on the bus 510, indicating that a single doubleword boundary is crossed. The multiplexer 334 now has an input DISP[7:2]$_{T-1}$ on the bus 337. The multiplexer 350 now selects between the zero bus 354 and a "4" bus 556 ("4" corresponding to the number of doublewords fetched whenever a pre-fetch is performed). The multiplexer 362 now selects between a zero bus 528 and the feedback path 370.

The operation of the circuit of Fig. 5 is analogous to the operation of the circuit of Fig. 3. Whenever an instruction buffer load is performed, the multiplexer 318 is controlled to add either 1 or 2 to the current DWORDS AHEAD value, corresponding to the number of doublewords being loaded into the instruction buffer 124. When an instruction is executed that does not cause a jump, the multiplexer 334 selects the output of the circuit 515 to decrement DWORDS AHEAD and increment DWORDS BEHIND by the number of doubleword boundaries crossed. When a branch within the instruction buffer is executed, the multiplexer 334 selects the bus 337 to subtract DISP[7:2] from the current DWORDS AHEAD value and add DISP[7:2] to the current DWORDS BEHIND value. When a pre-fetch is initiated, the multiplexer 350 selects the "4" bus 556 to increment DWORDS BEHIND by four. When a branch outside the instruction buffer 124 is executed, the multiplexers 304, 318, 334 350 and 362 all select their respective zero busses 512, 322, 328, 354 and 568 to reset DWORDS AHEAD and DWORDS BEHIND to zero.

The circuits and methods that have been described for performing a branch within an instruction buffer have been presented by way of example only, and not limitation. Thus, the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

WHAT IS CLAIMED IS:

1. An instruction buffer circuit that uses relative offset values to perform a jump within an instruction buffer without performing a fetch, comprising:

an instruction buffer;

a read pointer that provides an address for reading instruction data from said instruction buffer;

a first relative offset circuit that generates a first relative offset value, said first relative offset value indicating a number of instruction bytes in said instruction buffer that are ahead of said address provided by said read pointer;

a second relative offset circuit that generates a second relative offset value, said second relative offset value indicating a number of instruction bytes in said instruction buffer that are behind said address provided by said read pointer;

a compare circuit that compares a relative displacement value for a branch instruction to said first relative offset value or to said second relative offset value, to thereby determine whether a target instruction for said branch instruction can be read from said instruction buffer; and

a modify circuit that modifies said read pointer to point to said target instruction when said target instruction can be read from said instruction buffer.

2. The instruction buffer circuit as defined in Claim 1, wherein said instruction buffer acts as a first-in-first-out instruction buffer during sequential program execution.

3. The instruction buffer circuit as defined in Claim 1, further comprising an inhibit circuit that inhibits said compare circuit after a memory write operation.

4. A method of branching to a target instruction in an instruction buffer to avoid having to re-load said instruction buffer, said instruction buffer having a read pointer that provides an instruction buffer address for reading instruction data from said instruction buffer, said method comprising the steps of:

generating a relative offset value that indicates the number of instruction bytes in said instruction buffer that fall ahead of said instruction buffer address provided by said read pointer;

comparing said relative offset value to a relative displacement value for a branch instruction to determine whether a target instruction of said branch instruction is in said instruction buffer; and

incrementing said read pointer to the instruction buffer address of the target instruction without flushing said instruction buffer when said target instruction is in said instruction buffer to thereby effect a forward branch within said instruction buffer.

5. The method as defined in Claim 4, wherein said step of incrementing said read pointer is performed by adding said relative displacement value to the current read pointer value.

6. The method as defined in Claim 4, further comprising the steps of:

generating a second relative offset value that indicates the number of instruction bytes in said instruction buffer that fall behind said instruction buffer address provided by said read pointer;

comparing said second relative offset value to said relative displacement value for a branch instruction to determine whether a target instruction of said branch instruction is in said instruction buffer; and

decrementing said read pointer to the instruction buffer address of the target instruction when said target instruction is in said instruction buffer to thereby effect a backward branch within said instruction buffer.

7. An instruction buffer circuit for a microprocessor, comprising:

an addressable instruction array that comprises a plurality of storage locations that store a plurality of instruction bytes, said storage locations corresponding to sequential address locations in a memory from which said instruction bytes were transferred;

a current location pointer that points to one of said storage locations in said instruction array for a current instruction byte;

a circuit that generates a first relative value indicator that indicates the number of storage locations in said instruction array that correspond to instruction bytes from sequential address locations in said memory ahead of said current instruction byte;

a circuit that generates a second relative value indicator that indicates the number of storage locations in said instruction array that correspond to instruction bytes from sequential address locations in said memory behind said current instruction byte;

a comparison circuit that compares a relative displacement value in a branch instruction executed by said microprocessor with said first and second relative value indicators to determine whether a next instruction byte at an address resulting from said branch instruction is currently stored in said instruction array; and

a selection circuit that selects a byte from said instruction array as said next instruction byte when said comparison circuit indicates that said next instruction byte is currently stored in said instruction array.

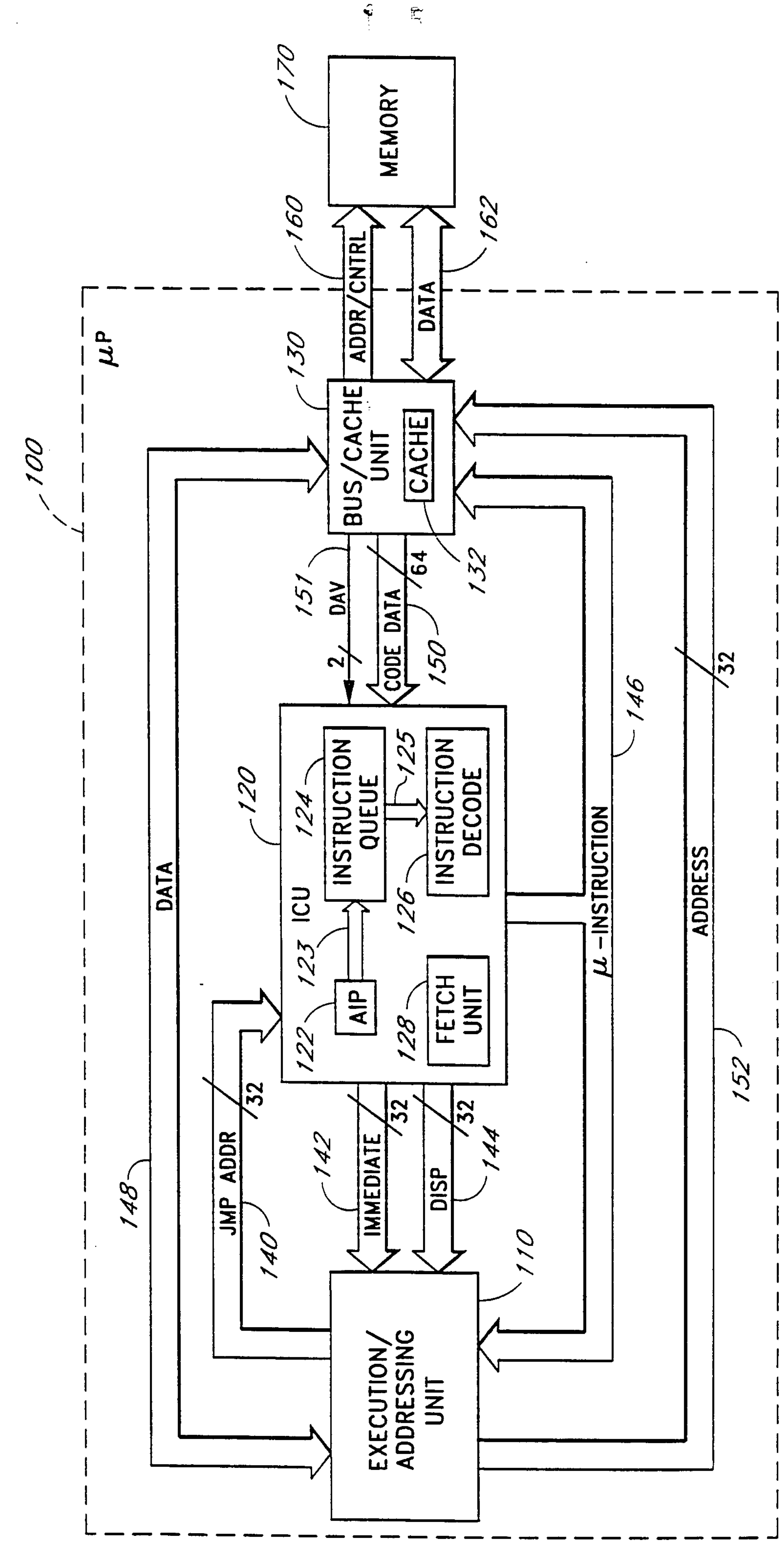

170
MEMORY
160
ADDR/CNTRL
162
DATA
μP
100
130
BUS/CACHE UNIT
CACHE
132
151
DAV
2
CODE DATA
64
150
120
ICU
124
INSTRUCTION QUEUE
125
INSTRUCTION DECODE
126
123
122
AIP
128
FETCH UNIT
146
μ-INSTRUCTION
ADDRESS
32
152
DATA
148
JMP ADDR
32
140
142
IMMEDIATE
32
DISP
32
144
110
EXECUTION/ ADDRESSING UNIT

FIG.1

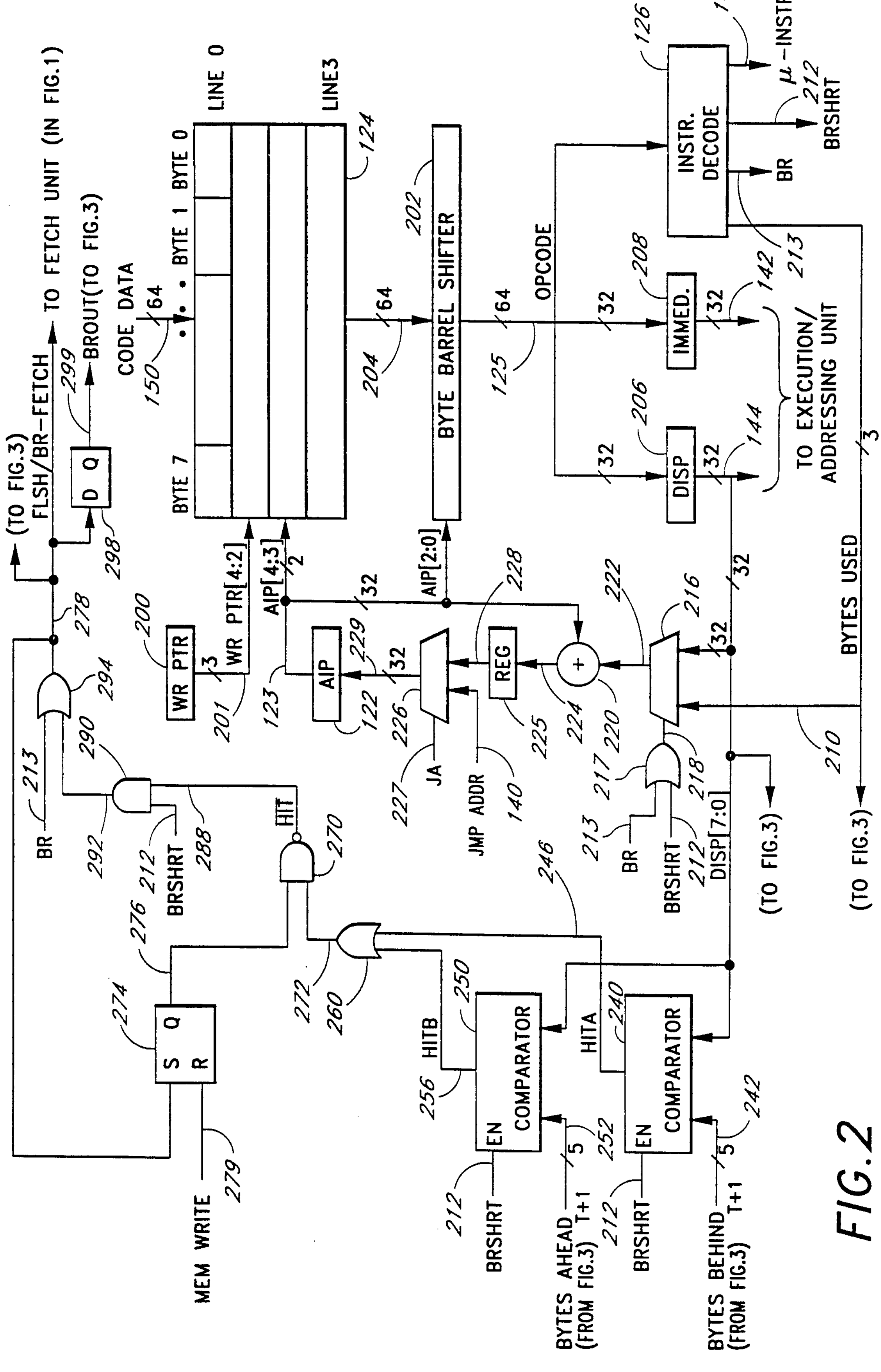

(TO FIG.3)
FLSH/BR-FETCH
TO FETCH UNIT (IN FIG.1)
299
BROUT(TO FIG.3)
CODE DATA
150
64
BYTE 7
BYTE 1
BYTE 0
LINE 0
LINE3
124
D
Q
298
278
200
WR PTR
3
WR PTR[4:2]
201
AIP[4:3]
2
AIP[2:0]
32
AIP
123
122
229
226
228
REG
225
224
+
222
216
220
217
218
64
204
BYTE BARREL SHIFTER
202
125
OPCODE
32
IMMED.
208
142
DISP
206
144
TO EXECUTION/
ADDRESSING UNIT
INSTR.
DECODE
126
146
μ-INSTR.
212
BRSHRT
BR
213
3
BYTES USED
210
(TO FIG.3)
DISP[7:0]
(TO FIG.3)
JA
227
JMP ADDR
140
BR
213
294
290
292
BRSHRT
212
288
HIT
270
276
246
272
260
250
HITB
COMPARATOR
EN
256
252
5
HITA
240
COMPARATOR
EN
242
5
274
S
Q
R
MEM WRITE
279
BRSHRT
212
BYTES AHEAD
(FROM FIG.3)
T+1
BRSHRT
212
BYTES BEHIND
(FROM FIG.3)
T+1

FIG.2

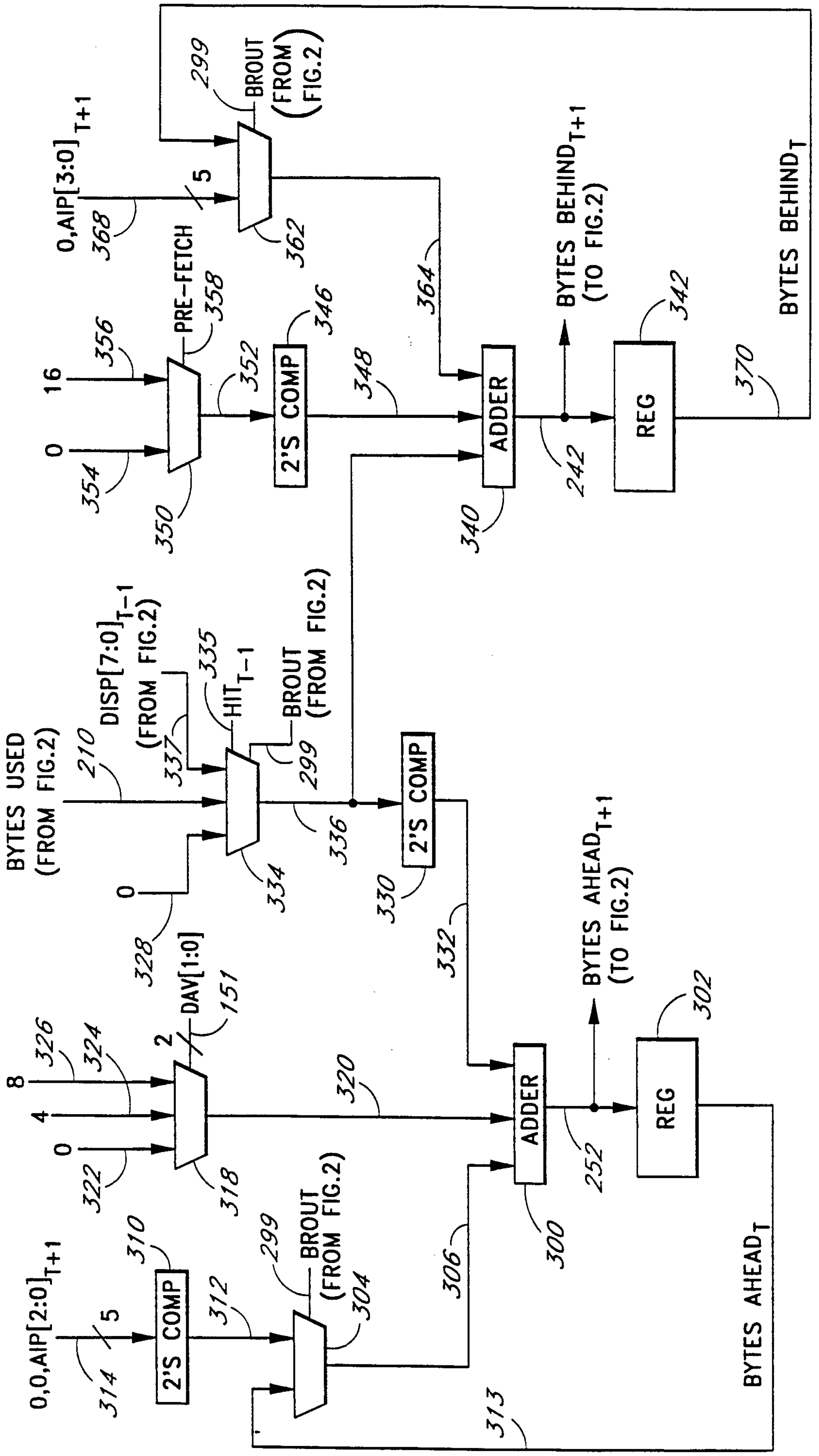

0,AIP[3:0]$_{T+1}$
368
5
299
BROUT (FROM FIG.2)
362
364
BYTES BEHIND$_{T+1}$ (TO FIG.2)
BYTES BEHIND$_T$
370
342
REG
242
ADDER
340
16
356
0
354
PRE-FETCH
358
350
352
2'S COMP
346
348
BYTES USED (FROM FIG.2)
210
DISP[7:0]$_{T-1}$ (FROM FIG.2)
337
HIT$_{T-1}$
335
BROUT (FROM FIG.2)
299
0
328
334
336
2'S COMP
330
332
BYTES AHEAD$_{T+1}$ (TO FIG.2)
302
REG
252
ADDER
300
8
326
4
324
0
322
DAV[1:0]
2
151
318
320
0,0,AIP[2:0]$_{T+1}$
5
314
2'S COMP
310
312
BROUT (FROM FIG.2)
299
304
306
313
BYTES AHEAD$_T$

FIG. 3

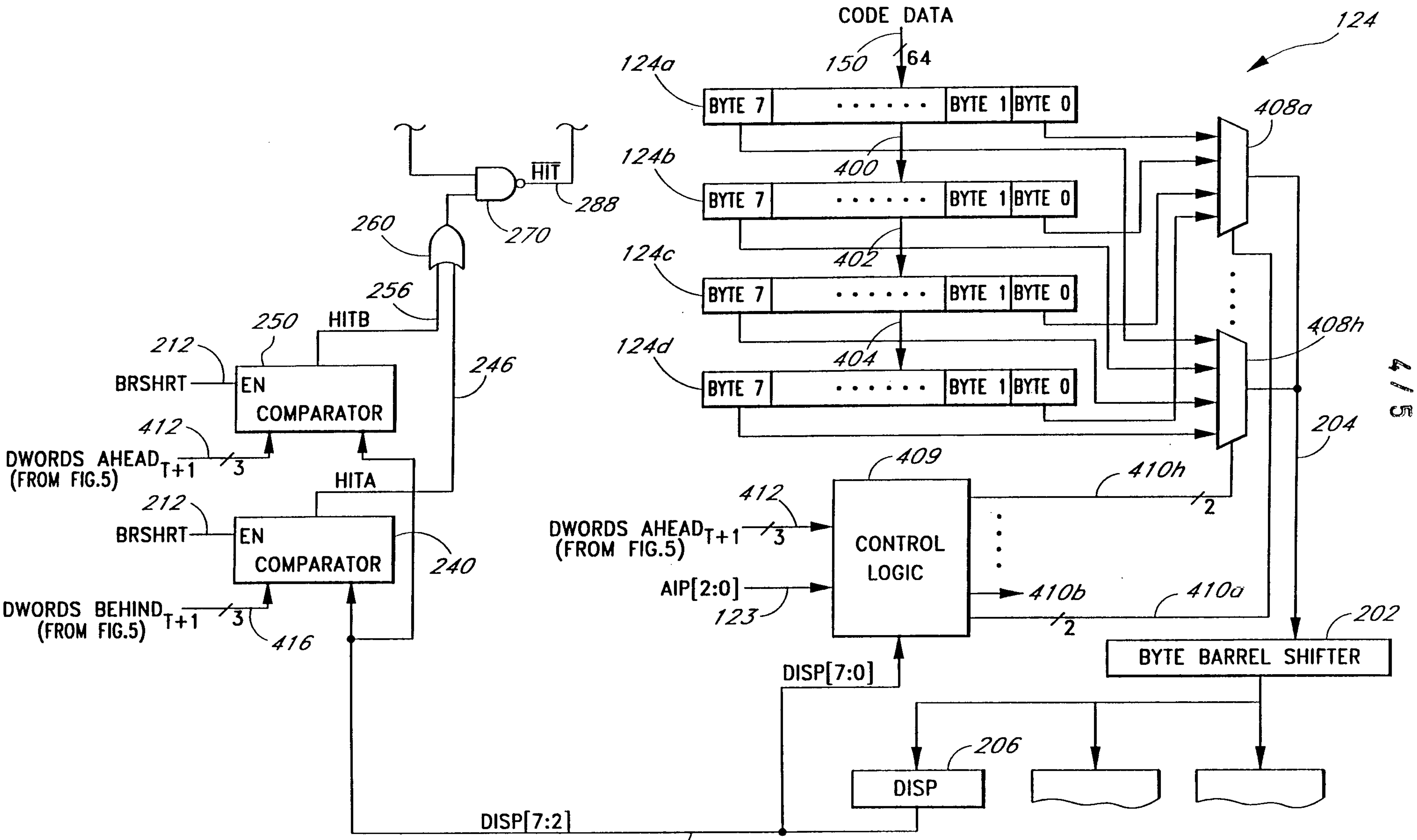

124
CODE DATA
150
64
124a
BYTE 7
BYTE 1
BYTE 0
400
124b
402
124c
404
124d
408a
408h
204
409
410h
2
412
DWORDS AHEAD$_{T+1}$
(FROM FIG.5)
3
CONTROL LOGIC
AIP[2:0]
123
410b
410a
2
202
BYTE BARREL SHIFTER
DISP[7:0]
206
DISP
DISP[7:2]
144
HIT
288
270
260
256
250
HITB
212
BRSHRT
EN
COMPARATOR
246
412
DWORDS AHEAD$_{T+1}$
(FROM FIG.5)
3
HITA
212
BRSHRT
EN
COMPARATOR
240
DWORDS BEHIND$_{T+1}$
(FROM FIG.5)
3
416

FIG. 4

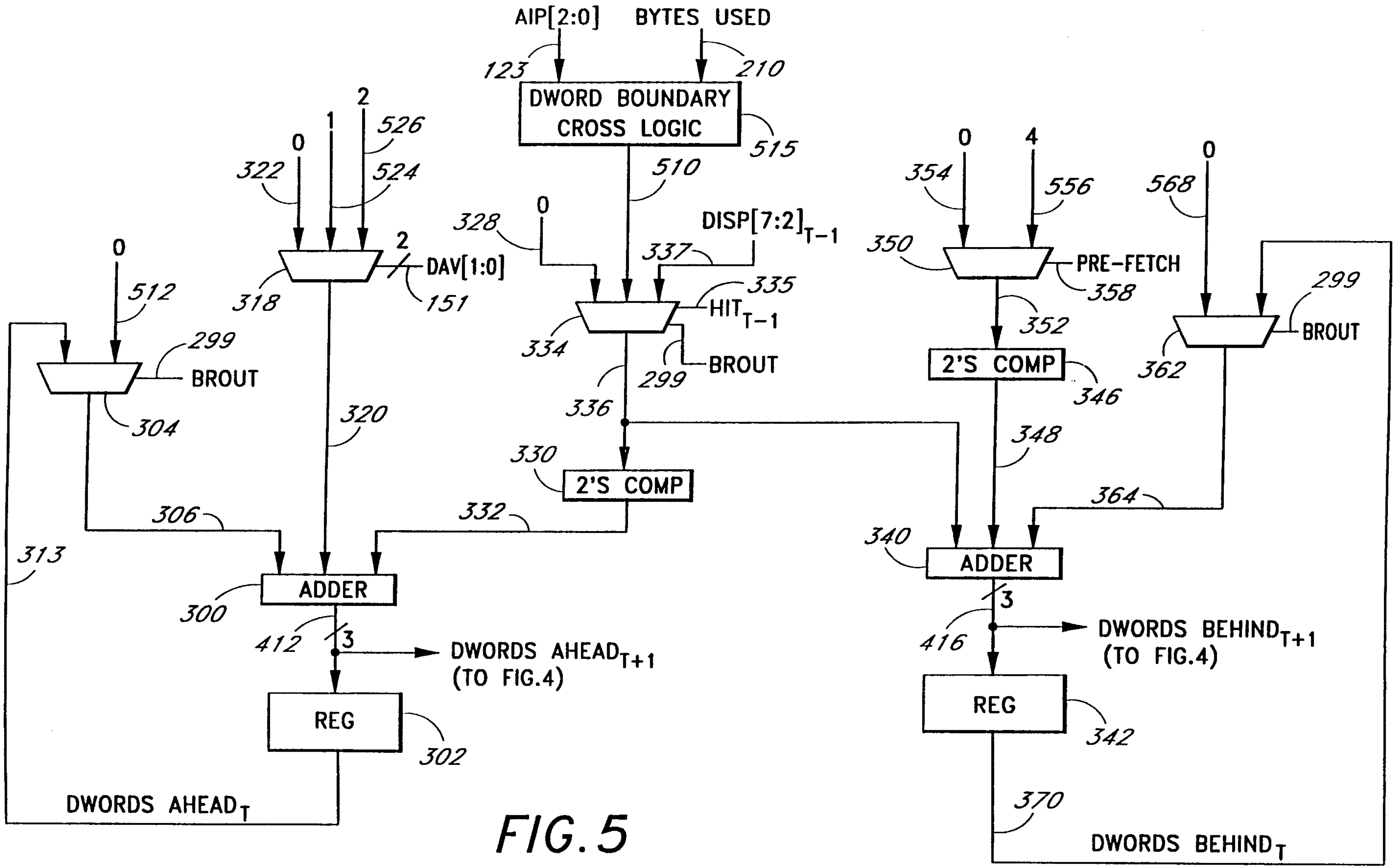

AIP[2:0]
BYTES USED
123
210
DWORD BOUNDARY CROSS LOGIC
515
510
0
1
2
526
524
322
2
DAV[1:0]
151
318
0
512
299
BROUT
304
320
328
0
DISP[7:2]$_{T-1}$
337
335
HIT$_{T-1}$
334
299
BROUT
336
330
2'S COMP
332
306
313
ADDER
300
412
3
DWORDS AHEAD$_{T+1}$
(TO FIG.4)
REG
302
DWORDS AHEAD$_T$
0
4
354
556
350
PRE-FETCH
358
352
2'S COMP
346
348
0
568
299
BROUT
362
364
340
ADDER
3
416
DWORDS BEHIND$_{T+1}$
(TO FIG.4)
REG
342
370
DWORDS BEHIND$_T$

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/01705

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 9/00,9/22,9/30,9/38
US CL : 395/375, 400, 425, 800
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)

U.S. : 395/375, 400, 425, 800

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 practicable, search terms used)

APS
search terms: instruction or prefetch buffer, pointer, branch or jump, target of branch

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US, A, 4,449,184, (POHLMAN, III ET AL). 15 MAY, 1984, WHOLE DOCUMENT	1-7
Y, P	US, A, 4,992,932 (OHSHIMA) 12 FEBRUARY, 1991, WHOLE DOCUMENT	1,4-7
A	US, A, 4,363,091 (POHLMAN, III ET AL), 07 DECEMBER, 1982, WHOLE DOCUMENT	1-7
A	US, A, 4,200,927 (HUGHES ET AL) 29 APRIL, 1980, WHOLE DOCUMENT	1,4,7
A	US, A, 5,226,126 (McFARLAND ET AL), 06 JULY, 1993, COLUMNS 5-8	1,4,7

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be part of particular relevance		
"E"	earlier document published on or after the international filing date	"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
"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)	"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
"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	"&"	document member of the same patent family

Date of the actual completion of the international search 04 APRIL 1995	Date of mailing of the international search report 31 MAY 1995
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer KRISHNA MALYALA Telephone No. (703) 305-9673