US00RE45278E

(19) **United States**

(12) **Reissued Patent**

Kasper

(10) **Patent Number: US RE45,278 E**

(45) **Date of Reissued Patent: *Dec. 2, 2014**

(54) **METHOD AND APPARATUS FOR CHANGING MICROCODE TO BE EXECUTED IN A PROCESSOR**

(75) Inventor: **Christian D. Kasper**, Andover, MA (US)

(73) Assignee: **STMicroelectronics, Inc.**, Coppell, TX (US)

(*) Notice: This patent is subject to a terminal disclaimer.

(21) Appl. No.: **12/914,978**

(22) Filed: **Oct. 28, 2010**

Related U.S. Patent Documents

Reissue of:

(64) Patent No.: **7,444,630**
Issued: **Oct. 28, 2008**
Appl. No.: **10/774,994**
Filed: **Feb. 9, 2004**

U.S. Applications:

(63) Continuation of application No. 09/475,927, filed on Dec. 30, 1999, now Pat. No. 6,691,308.

(51) **Int. Cl.**
G06F 9/45 (2006.01)

(52) **U.S. Cl.**
USPC **717/136**; 717/139; 717/140; 717/148; 717/159

(58) **Field of Classification Search**
CPC G06F 8/52; G06F 8/41; G06F 8/443; G06F 9/45508; G06F 9/45516
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,136,696	A *	8/1992	Beckwith et al.	712/240
5,796,974	A	8/1998	Goddard et al.	
5,860,104	A	1/1999	Witt et al.	
5,887,152	A	3/1999	Tran	
5,983,337	A	11/1999	Mahalingaiah et al.	
6,049,672	A *	4/2000	Shiell et al.	717/168
6,101,580	A *	8/2000	Agesen et al.	711/132
6,192,516	B1 *	2/2001	Griesemer	717/139
6,240,506	B1 *	5/2001	Miller	712/213
6,295,644	B1	9/2001	Hsu et al.	
6,397,379	B1 *	5/2002	Yates et al.	717/140
6,502,237	B1 *	12/2002	Yates et al.	717/136
6,629,312	B1 *	9/2003	Gupta	717/136
6,691,308	B1 *	2/2004	Kasper	717/168

(Continued)

OTHER PUBLICATIONS

Lin et al., A compact DSP core with static floating-point unit & its microcode generation, Apr. 2004, 4 pages.*

(Continued)

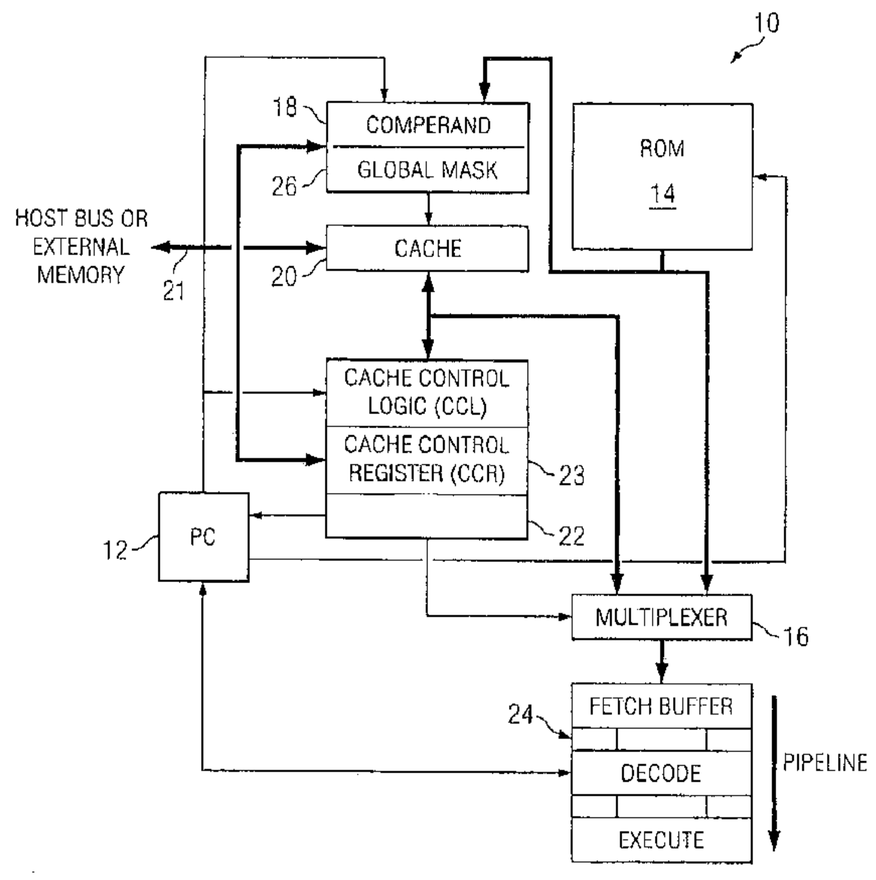
Primary Examiner — Thuy Dao

(74) *Attorney, Agent, or Firm* — Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A Central Processing Unit (CPU) hotpatch circuit compares the run-time instruction stream against an internal cache. The internal cache stores embedded memory addresses with associated control flags, executable instruction codes, and tag information. In the event that a comparison against the current program counter succeeds, then execution is altered as required per the control flags. If no comparison match is made, then execution of the instruction that was accessed by the program counter is executed.

31 Claims, 4 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,704,926	B1 *	3/2004	Blandy et al.	717/148
6,718,539	B1 *	4/2004	Cohen et al.	717/136
6,851,109	B1 *	2/2005	Alexander et al.	717/148
7,036,118	B1 *	4/2006	Ulery et al.	717/159
7,134,119	B2 *	11/2006	Nevill	717/139
7,254,806	B1 *	8/2007	Yates et al.	717/136

OTHER PUBLICATIONS

A. K. Tirrell, A study of the application of compiler techniques to the generation of micro-code, Aug. 1974, 18 pages.*

IBM Technical Disclosure Bulletin, published Oct. 1, 1993.

* cited by examiner

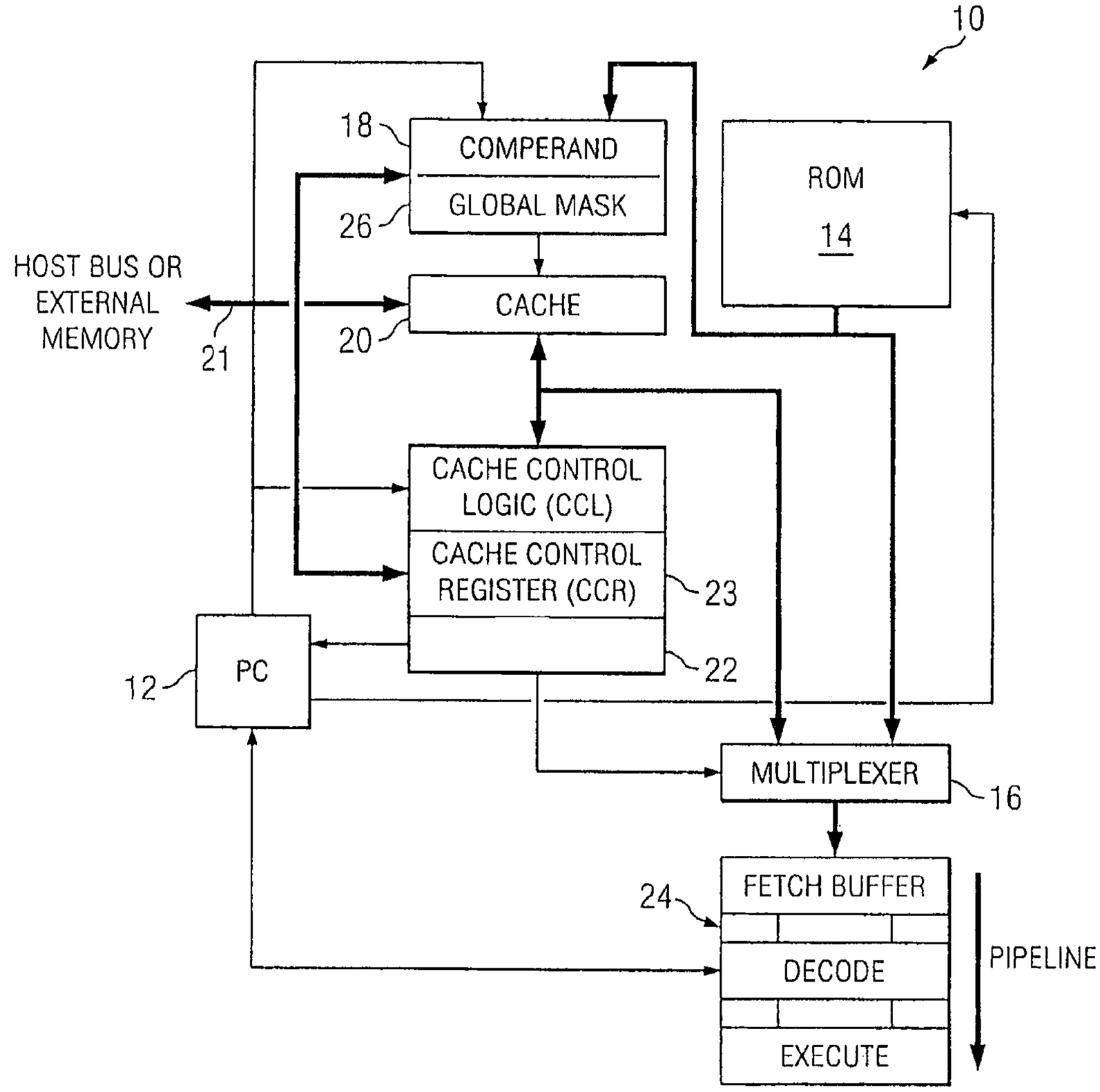

10
18
COMPERAND
GLOBAL MASK
26
ROM
14
HOST BUS OR
EXTERNAL
MEMORY
21
CACHE
20
CACHE CONTROL
LOGIC (CCL)
CACHE CONTROL
REGISTER (CCR)
23
22
12
PC
MULTIPLEXER
16
FETCH BUFFER
24
DECODE
EXECUTE
PIPELINE

FIG. 1

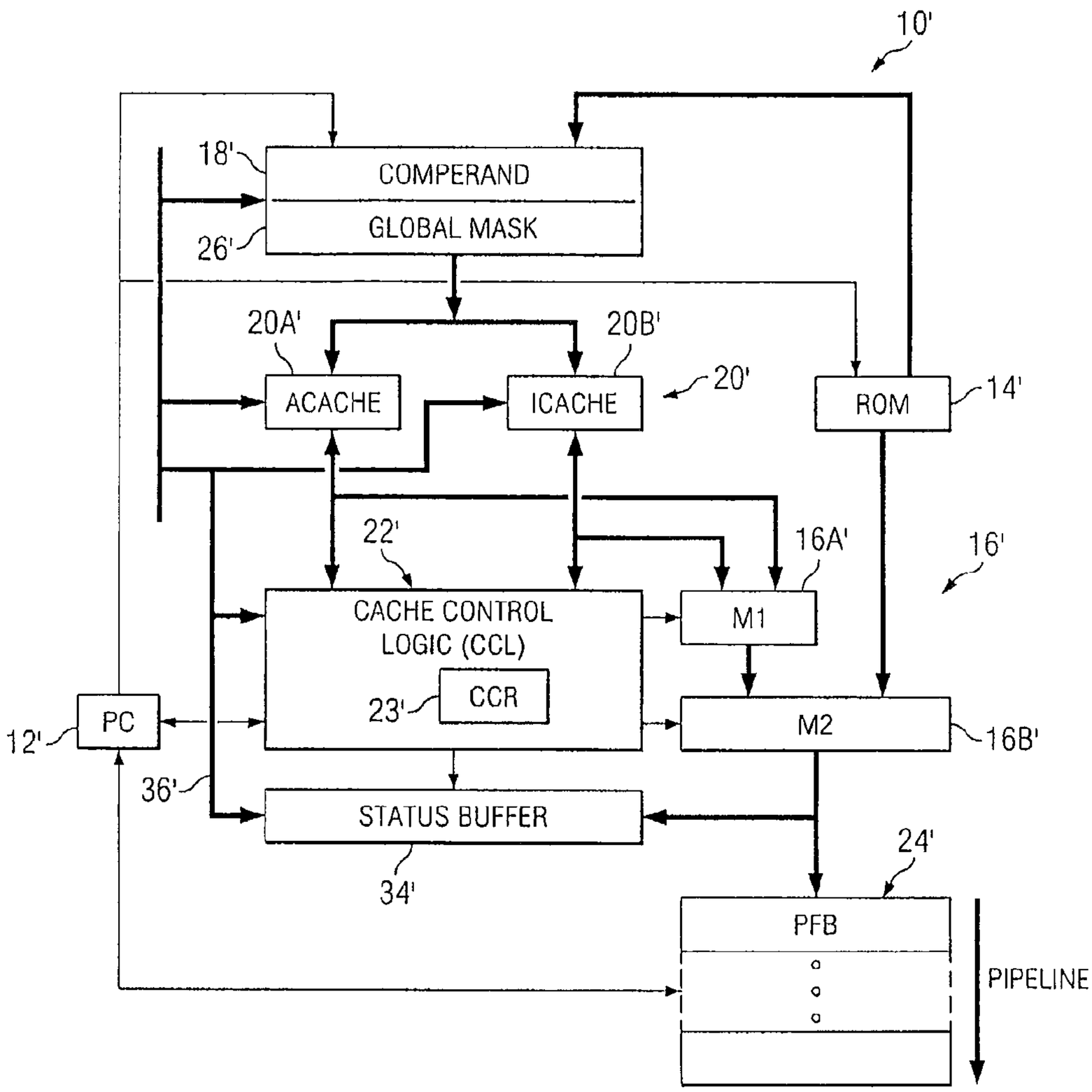

10'
18'
COMPERAND
GLOBAL MASK
26'
20A'
20B'
ACACHE
ICACHE
20'
ROM
14'
22'
16A'
16'
CACHE CONTROL LOGIC (CCL)
M1
23'
CCR
PC
12'
M2
16B'
36'
STATUS BUFFER
24'
34'
PFB
PIPELINE

FIG. 2

CONTROL FLAGS	FIELD	OPERATION	OTHER	EXPLANATION
1.	V			VALID
2.	A			ADDRESS
3.	O			OPCODE
4.	G			GLOBAL
5.		I		INSERT
6.		M		MATCH
7.		B		BLOCK ASSIGNMENT
8.		X		DELETE
9.			E	EXTERNAL ROM
10.			H	HALT
11.			L	LOCK
12.			Q	GENERATE TRAP

FIG. 3

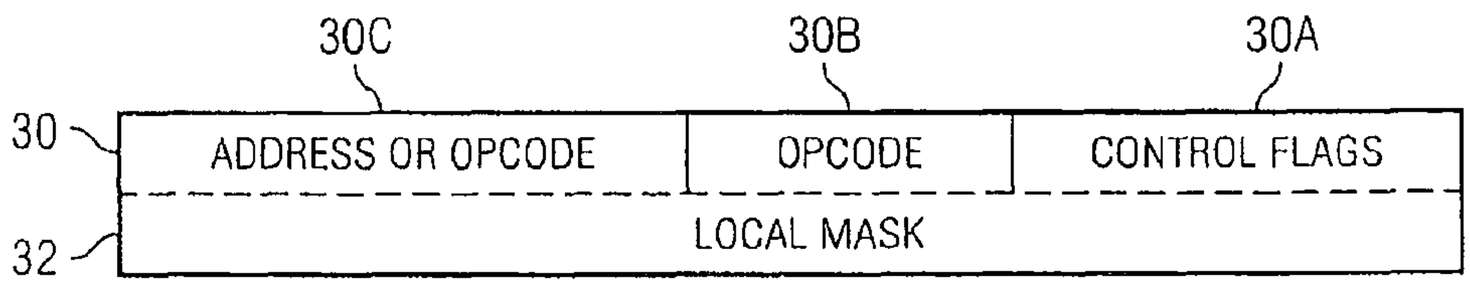
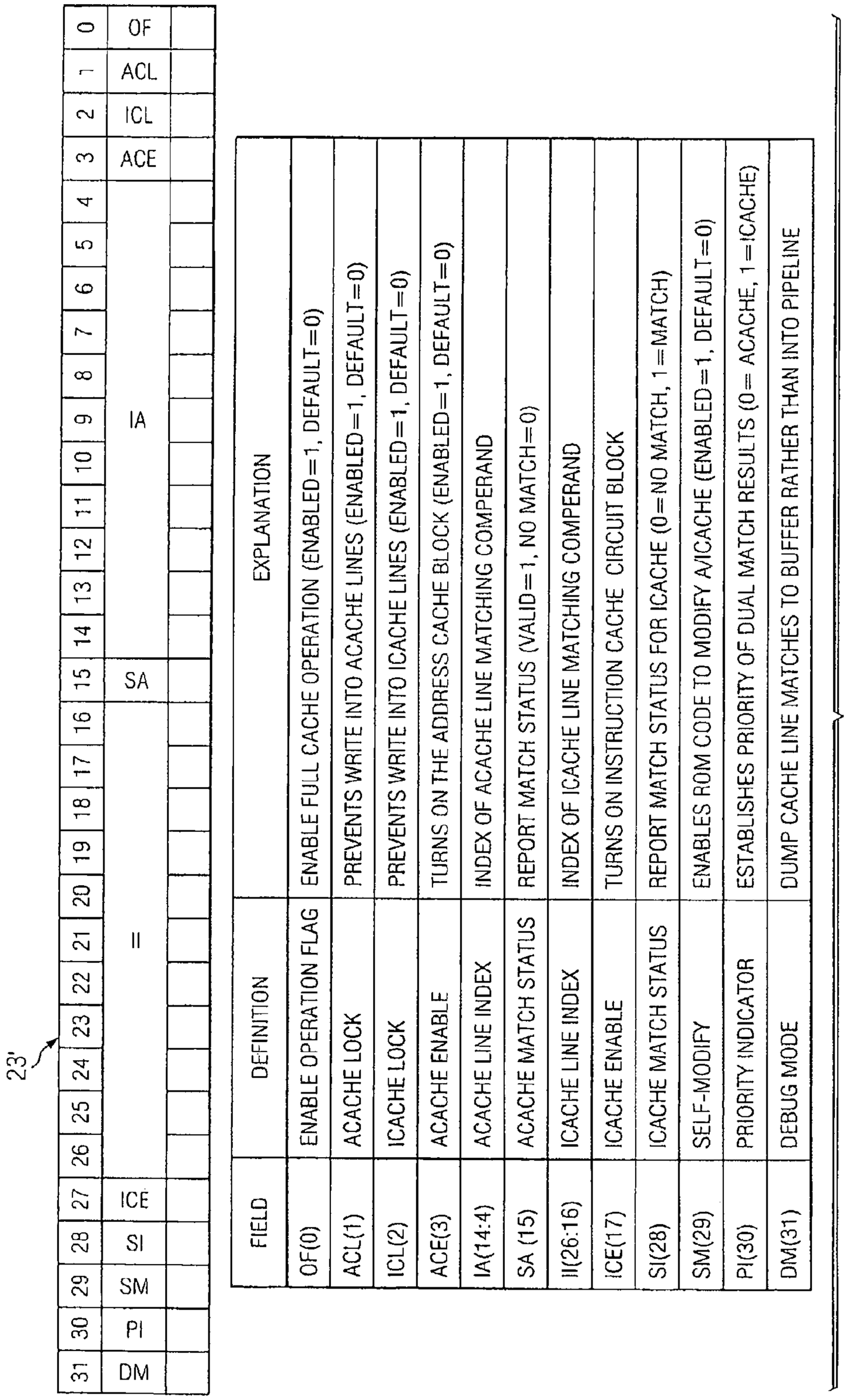


FIG. 4



FIELD	DEFINITION	EXPLANATION
OF(0)	ENABLE OPERATION FLAG	ENABLE FULL CACHE OPERATION (ENABLED=1, DEFAULT=0)
ACL(1)	ACACHE LOCK	PREVENTS WRITE INTO ACACHE LINES (ENABLED=1, DEFAULT=0)
ICL(2)	ICACHE LOCK	PREVENTS WRITE INTO ICACHE LINES (ENABLED=1, DEFAULT=0)
ACE(3)	ACACHE ENABLE	TURNS ON THE ADDRESS CACHE BLOCK (ENABLED=1, DEFAULT=0)
IA(14:4)	ACACHE LINE INDEX	INDEX OF ACACHE LINE MATCHING COMPERAND
SA (15)	ACACHE MATCH STATUS	REPORT MATCH STATUS (VALID=1, NO MATCH=0)
II(26:16)	ICACHE LINE INDEX	INDEX OF ICACHE LINE MATCHING COMPERAND
ICE(17)	ICACHE ENABLE	TURNS ON INSTRUCTION CACHE CIRCUIT BLOCK
SI(28)	ICACHE MATCH STATUS	REPORT MATCH STATUS FOR ICACHE (0=NO MATCH, 1=MATCH)
SM(29)	SELF-MODIFY	ENABLES ROM CODE TO MODIFY A/ICACHE (ENABLED=1, DEFAULT=0)
PI(30)	PRIORITY INDICATOR	ESTABLISHES PRIORITY OF DUAL MATCH RESULTS (0= ACACHE, 1=ICACHE)
DM(31)	DEBUG MODE	DUMP CACHE LINE MATCHES TO BUFFER RATHER THAN INTO PIPELINE

FIG. 5

METHOD AND APPARATUS FOR CHANGING MICROCODE TO BE EXECUTED IN A PROCESSOR

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue.

This application is a continuation of prior application Ser. No. 09/475,927 filed on Dec. 30, 1999 now U.S. Pat. No. 6,691,308.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates generally to a processors and, more specifically, to a method and apparatus which provides run-time correction for microcode, code enhancement, and/or interrupt vector reassignment.

2. Description of the Prior Art

For integrated circuits which are driven by microcode which is embedded in internal memory, many times it is necessary to either have the instruction content of the embedded memory device or the behavior of the Central Processing Unit (CPU) pipeline itself corrected or debugged in the field. This may require on-the-fly modifications driven by customer request or updates due to evolution of industry protocol standards. However, this creates problems since it is difficult to correct and/or debug these types of circuits. Debugging and/or changing the embedded microcode is a time consuming effort which generally requires messy CRC changes or related checksum modifications.

Therefore, a need existed to provide a circuit by which either the instruction content of the internal memory and/or the behavior of the CPU pipeline itself could be corrected and/or debugged in the field. The debug circuit should consume only a small amount of silicon real estate, be inexpensive to implement and allow changes at a faster rate then current techniques. The debug circuit must also provide a means by which the debug circuit could download data to debug the device. Data could be downloaded by the host system or managed via a simplistic communication scheme as described in the ST52T3 data book written by STMicroelectronics, Inc.

SUMMARY OF THE INVENTION

It is object of the present invention to provide a circuit by which either the instruction content of an internal memory and/or the behavior of the CPU pipeline itself could be corrected and/or debugged in the field.

It is another object of the present invention to provide a debug circuit which consumes only a small amount of silicon real estate, is inexpensive to implement and allow changes at a faster rate then current techniques.

It is still a further object of the present invention to provide a debug circuit which provides a means by which the debug circuit could download data to debug the device.

BRIEF DESCRIPTION OF THE PREFERRED EMBODIMENTS

In accordance with one embodiment of the present invention, a hot patch system for changing of code in a processor is disclosed. The hot patch system has a memory, such as a Read Only Memory (ROM), for storing a plurality of instructions. A program counter is coupled to the memory for indexing of the memory to access an instruction. A cache system is coupled to the memory and to the program counter. The cache system is used for comparing information associated with the instruction from memory with information stored in the cache system. If there is a comparison match, the cache system alters the instruction stream as designated by information stored in the cache system. If no match occurs, the cache system sends the instruction from memory into the instruction stream.

In accordance with another embodiment of the present invention, a method of altering the code of a pipeline processor is disclosed. The method requires that a plurality of instructions be stored in memory. A cache is provided and information is stored in the cache. The memory is indexed to access one of the instructions stored in memory. Information associated with the instruction from memory is compared with information stored in the cache. If a comparison match is made, the instruction stream is altered as designated by the information stored in the cache. If no comparison match is made, the instruction from memory is inserted into the instruction stream.

The foregoing and other objects, features, and advantages of the invention will be apparent from the following, more particular, description of the preferred embodiments of the invention, as illustrated in the accompanying drawing.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a simplified block diagram of one embodiment of the hot patch circuit of the present invention.

FIG. **2** is a simplified block diagram of a second embodiment of the hot patch circuit of the present invention.

FIG. **3** shows one example of the different fields associated with the cache used in the present invention.

FIG. **4** shows one example of the control codes used in the control flag field of FIG. **3**.

FIG. **5** shows one example of the bit configuration of the cache control register used in the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. **1**, one embodiment of a hot patch circuit **10** (hereinafter circuit **10**) is shown. The circuit **10** provides a means whereby the instruction content of an embedded memory device or the behavior of the Central Processing Unit (CPU) may be corrected, modified and/or debugged. The circuit **10** is preferably used in a pipeline CPU.

The circuit **10** has a program counter **12**. The program counter **12** is coupled to a memory device **14** and to a register **18**. The program counter **12** is used to generate an address of an instruction to be executed. When the address is generated, the program counter **12** will index the memory unit **14**. The memory unit **14** stores instructions which are to be executed by the CPU. The memory unit **14** is a nonvolatile memory device like a Read Only Memory (ROM) device. Once the program counter **12** access the instruction which is stored in the memory unit **14**, the instruction is sent to a multiplexer **16**.

The register **18** is coupled to the memory unit **14**, the program counter **12**, and to a cache unit **20**. The register **18** is used to store data which will be compared to data which is stored in the cache unit **20**. The register **18** may either store

the address sent from the program counter **12** or the instruction which the program counter **12** access from the memory unit **14**.

As may be seen in FIG. **4**, the cache unit **20** is comprised of a plurality of cache lines **30**. Each cache line **30** is comprised of at least three different fields: a control flag field **30**A, an op-code field **30**B which stores the new instruction to be inserted into the instruction stream, and an address/op-code field **30**C which stores the data which is to be compared to the data stored in the register **18**. The size of the fields will vary based on the implementation of the circuit **10**. In accordance with one embodiment of the present invention, the width of the cache line **30** would be able to accommodate at least a 32-bit op-code (field **30**B) along with a 10 to 32-bit address/op-code (field **30**C) and a 2 to 8-bit control flag field (field **30**A). This would yield a cache line width between 44 to 72-bits. However, it should be noted that these field lengths are only given as one example and should not be seen as limiting the scope of the present invention. As stated above, bit field dimensions will vary depending on the size of the memory unit **14**.

The control flag field **30**A is used to dictate both the semantic content and the execution behavior of individual or multiple cache lines **30**. The number of control flags is dependent upon the allocated field size. In some cases, combination of control flags may be useful. Control flags may be used to either delete or enable cache unit entries, or provide alternate semantic information regarding the register content. Referring to FIG. **3**, some examples of the control flag code is shown. The valid flag "V" indicates whether the entry in the cache unit **20** is valid. The "A" and "O" flags indicate whether the information to be compared is an address or an op-code. The global flag "G" allows for greater than a 1:1 mapping. For example, if the address flag "A" is set, one would only be comparing the one particular address in the memory unit **14**. Thus, there is only a 1:1 mapping. However, if the op-code "O" and global "G" flags are set, one would be able to replace every occurrence of a particular instruction that is accessed from the memory unit **14**. Thus, the global flag "G" allows one to make better use of the space in the cache unit **20**. The insert "I", match "M", block assignment "B", and delete "X" flags are used by the cache control logic **22** to control access to the instruction stream. The "I" flag implies that the associated op-code in the cache is to be inserted into the instruction stream. The "M" flag indicates that when the contents of the register **18** matches that in the cache unit **20**, the cache unit instruction is to replace the instruction from the memory unit **14** in the instruction stream. The "B" flag allows for more than one instruction (i.e., a block of instructions) that is stored in the cache unit **20** is to be clocked into the instruction stream. The "X" indicates that the relevant instruction is to be ignored or deleted (i.e., no operation (NOP)). The "E", "H", "L", and "Q" flags are pipeline control flags. The "E" flags indicates that if there is a match to jump to external memory using the address in the "opcode field" and to execute the instructions in external memory starting at that location. The "H" flag allows one to stop the clock for purposes of debugging the pipeline. The "L" flag allows one to lock the cache unit **20** and the "Q" flag is a generate trap flag. The control codes shown in FIG. **3** are just examples and should not be seen to limit the scope of the present invention. Different sets or embodiments of flags could be used depending on the particular implementation.

In the embodiment depicted in FIG. **1**, the cache unit is a fully associative or direct-mapped cache which would contain memory unit addresses with associated control flags, executable instructions, and tag information. The cache unit **20** may be a content addressable memory whereby the data in the register **18** is compared to all the contents in the cache unit **20**.

The cache **20** is also coupled to a bus **21**. The bus **21** could be coupled to a host bus or to external memory. The bus **21** allows data to be downloaded into the cache **20** or for allowing instructions to be executed from the external memory. Contents of the cache **20** could be downloaded by the host system or managed via a simple communication scheme as described in the ST52T3 data book written by STMicroelectronics, Inc.

Cache control logic **22** is coupled to the cache unit **20** and to te multiplexer **16**. The cache control logic **22** controls the operation of the cache unit **20** and when a particular instruction will be inserted into the instruction stream of the pipeline **24**. If there is no comparison match, the cache control logic **22** will let the instruction from the memory unit **14** flow through the multiplexer **16** to the pipeline **24**. When there is a comparison match, the instruction from the memory unit **14** is replaced by a new instruction from the cache unit **20** in the pipeline **24**. The cache control logic **22** will have a cache control register **23**. The cache control register **23** allows one to control the cache unit **20** and to control insertion of an instruction into the pipeline **24**. By setting various bits in the cache control register **23**, one would be able to enable/disable the cache unit **20**, modify the contents of the cache unit **20** and control the general operation of the cache unit **20**. The cache control register **23** will be described in further detail in relation to the dual cache system of FIG. **2**.

A mask register **26** may be coupled to the cache unit **20**. The mask register **26** may be a global mask register which would affect the entire cache unit **20** or a local mask register **32** (FIG. **3**) whereby a single cache line **30** would have an associated local mask register **32**. The mask register **26** provides flexibility to the circuit **10**. The mask register **26** allows flexibility by allowing one to control how the data from the memory unit **14** is matched with data in the cache unit **20**. For example, if all of the bits in the global mask register **26** were set to 1, then what ever data came through the register **18** would be matched one to one against that of the cache unit **20**. One could also set the global mask register **26** to invalidating the cache unit **20** and let the memory unit instructions be executed as accessed by the program control **12**. The mask register **26** may also be used to modify the contents of the cache unit **20** by using simple write instructions.

Referring to FIG. **2**, a second embodiment of the present invention is shown wherein like numerals represent like elements with the exception of a "" to indicate another embodiment. The circuit **10'** looks and operates in a similar fashion as circuit **10** depicted in FIG. **1**. One difference in circuit **10'** is that the cache **20'** is divided into two separate caches: an address cache **20**A' and an instruction cache **20**B'. Thus, for the address cache **20**A', the third field of the cache line will contain the memory unit address location to be matched, and for the instruction cache **20**A', the third field of the cache line will contain the memory unit instruction to be matched.

The cache control logic **22'** operates in a similar fashion as disclosed above. For the dual cache system, one implementation of the cache control register **23'** is shown in FIG. **5**. As can be seen in FIG. **5**, by setting different bits in the cache control register **23'**, one is able to control the operation of the cache unit **20'**. The catch control register **23'** depicted in FIG. **5** would be used in the dual cache system of FIG. **2**. In this particular embodiment, the cache control register **23'** has locking, enabling, indexing, and match status bits for both the address cache **20**A' and the index cache **20**B'. Bits like the enable operation bit and the debug mode bit could be used in

either the single cache system of FIG. **1** or the dual cache system of FIG. **2**. The cache control register bit definition as shown in FIG. **5** is just one example and should not be seen to limit the scope of the present invention. Different configuration of bits could be used depending on the particular implementation.

The dual cache system also uses two multiplexers **16**A' and **16**B'. The first multiplexer **16**A' has a first input coupled to the output of the address cache **20**A', a second input coupled to the output of the instruction cache **20**B', a third input coupled to the cache control logic **22'**, and an output coupled to the second multiplexer **16**B'. The second multiplexer **16**B' has a first input coupled to the output of the first multiplexer **16**A', a second input coupled to the output of the memory device **14'**, a third input coupled to the cache control logic **22'**, and outputs coupled to the pipeline **24'** and the status buffer **34'**. In operation, the cache control logic **23'** will control which cache **20**A' or **20**B' is enabled and if there is a dual match if both caches **20**A' and **20**B' are enabled, which cache has priority. If there is a comparison match, the cache control logic **22'** will cause the multiplexer **16**A' to send an output from the cache unit **20'** to the second multiplexer **16**B'. The cache control logic **22'** will then cause the multiplexer **16**B' to insert the output from the cache unit **20'** into the instruction stream to be executed. If there is no comparison match, the cache control logic **22'** will cause the multiplexer **16**B' to insert the instruction from the memory unit **14'** into the pipeline **24'**.

In the embodiment depicted in FIG. **2**, the circuit **10'** has a status buffer **34'**. The status buffer **34'** has an input coupled to the cache control logic **22'**, an input coupled to the second multiplexer **16**B', and an input coupled to the bus **36'**. The status buffer is used to store information related to the operation of the circuit **10'**. For example, the status buffer could be used to gather debug information such as what line of code was matched. Although not shown in FIG. **1**, it should be noted that the status buffer **34'** could also be used in the embodiment depicted in FIG. **1**.

OPERATION

Referring now to Table 1 below, the operation of circuit **10** will be described. It should be noted that the operation of circuit **10'** is similar to **10** and will not be described in detail.

TABLE 1

	Cache				
	Flags	Address	Op-code	Program Counter	Code Stream
1	MA	0111111	CP32 A,C	0111111	CP32 A,C
2	IR	1000000	MOV A,B	1000000	100000
3	RA	1000010	SAV B	1000001	MOV A,B
4	RA	1000011	ADD B,C	1000010	100001
5	XA	1000101		1000011	SAV B
				1000101	ADD B,C
				1000110	NOP
				1000111	1000110
					1000111

When the program counter **12** generates the address 0111111, the program counter **12** will index the memory unit **14**. The instruction associated with address 0111111 from the memory unit **14** will be stored in the multiplexer **16**. The address from the program counter **12** is also sent to the register **18** where it is compared to the data stored in the cache unit **20**. As can be seen above, for address 0111111 there is a comparison match with cache line 1. Since the "M" flag is set for cache line 1, the op-code in cache line 1 will replace the instruction from memory. Thus the cache control logic **23** will send the CP32 A,C instruction associated with cache line 1 through the multiplexer **16** into the pipeline **24** to be execute.

The next address generated by the program counter **12** is 1000000. The memory unit instruction associated with address 1000000 is sent from the memory unit **14** and stored in the multiplexer **16**. The address generated by the program counter **12** is sent to the register **18** where it is compared to the data stored in the cache unit **20**. For the address 1000000 there is a comparison match with cache line 2. Since the "I" flag is set for cache line 2, the op-code in cache line 2 (i.e., MOV A,B) will be inserted into the instruction stream after the instruction associated with the memory unit address location 1000000.

The next address generated by the program counter **12** is 1000001. For this address there is no comparison match. Thus, the cache control logic **23** will send the instruction associated with memory unit address location 1000001 through the multiplexer **16** into the pipeline **24** to be execute.

For the next address, 1000010, there is a comparison match with cache line 3. Since the "R" flag is set in cache line 3, the op-code in cache line 3 (i.e., SAV B) replaces the memory unit instruction associated with the address 1000010 in the instruction stream.

The next address generated by the program counter is 1000011. For this address, there is a comparison match with cache line 4. Since the "R" flag is set in cache line 4, the op-code ADD B,C in cache line 4 replaces the memory unit instruction associated with the address 1000011 in the instruction stream.

The next address in the program counter is 1000101. Again there is a comparison match. This time the match is with cache line 5. Cache line 5 has the "X" flag set so the instruction is ignored or deleted (i.e., no operation (NOP)).

For the last two addresses in the program counter, 1000110 and 1000101, this is no comparison match. Thus, the cache control logic **23** will send the instruction associated with these memory unit address locations through the multiplexer **16** into the pipeline **24** to be execute.

While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that the foregoing and other exchanges in form and details may be made therein without departing from the spirit and scope of the invention.

What is claimed is:

1. A hot patch system comprising:

a read only memory storing a plurality of instructions;

a cache memory storing alternate instructions for at least one instruction stored within the read only memory, each cache line within the cache memory having associated therewith one or more selection flags;

a program counter coupled to the read only memory and to the cache memory, the program counter transmitting an address to both the read only memory and the cache memory; and

cache control logic coupled to the cache memory, the cache control logic comparing selected information associated with an instruction stored at the address in the read only memory with counterpart selected information associated with an instruction stored at the address in the cache memory, wherein the selected information and counterpart selected information are selected based upon the one or more selection flags associated with a cache line corresponding to the address.

2. The hot patch system according to claim **1**, wherein the one or more selection flags include a first flag indicating

whether the cache control logic is to compare addresses and a second flag indicating whether the cache control logic is to compare opcodes.

3. The hot patch system according to claim **2**, wherein the second flag, when set, indicates that the cache control logic is to compare an opcode stored at the address within the read only memory with an opcode stored at the address within the cache memory.

4. The hot patch system according to claim **2**, wherein the first and second flags may be individually set, so that both flags may be set, one flag may be set while the other flag is not set, or both flags may be not set.

5. The hot patch system according to claim **2**, wherein the one or more selection flags further include a third flag indicating whether an opcode comparison should be performed for every instance of an opcode within the read only memory.

6. The hot patch system according to claim **1**, wherein each cache line within the cache memory has associated therewith one or more instruction flow control flags, wherein the cache control logic causes a change in instruction flow based upon the one or more instruction flow control flags associated with the cache line corresponding to the address when there is a comparison match between the selected information and the counterpart selected information.

7. The hot patch system according to claim **6**, wherein the one or more instruction flow control flags include an insert flag indicating that an opcode at the address within the cache memory is to be inserted prior to execution of an opcode at the address within the read only memory.

8. The hot patch system according to claim **6**, wherein the one or more instruction flow control flags include a replace flag indicating that an opcode at the address within the cache memory is to be executed instead of an opcode at the address within the read only memory.

9. The hot patch system according to claim **6**, wherein the one or more instruction flow control flags include a block flag indicating that more than one opcode starting at the address within the cache memory are to be executed instead of or before an opcode at the address within the read only memory.

10. The hot patch system according to claim **6**, wherein the one or more instruction flow control flags include a noop flag indicating that an opcode at the address within the read only memory is to be skipped.

11. A *processor-implemented* hot patch method, comprising:

storing a plurality of instructions within a read only memory;

storing alternate instructions for at least one instruction in the read only memory within a cache memory, each cache line within the cache memory having associated therewith one or more selection flags;

transmitting an address to both the read only memory and the cache memory; and

comparing selected information associated with an instruction stored at the address in the read only memory with counterpart selected information associated with an instruction stored at the address in the cache memory, wherein the selected information and counterpart selected information are selected based upon the one or more selection flags associated with a cache line corresponding to the address.

12. The hot patch method according to claim **11**, wherein the one or more selection flags include a first flag indicating whether addresses are to he compared and a second flag indicating whether opcodes are to be compared.

13. The hot patch method according to claim **12**, wherein the second flag, when set, indicates that an opcode stored at the address within the read only memory is to be compared with an opcode stored at the address within the cache memory.

14. The hot patch method according to claim **12**, wherein the first and second flags may be individually set, so that either both flags may be set, one flag may be set while the other flag is not set, or both flags may be not set.

15. The hot patch method according to claim **12**, wherein the one or more selection flags further include a third flag indicating whether an opcode comparison should be performed for every instance of an opcode within the read only memory.

16. The hot patch method according to claim **11**, wherein each cache line within the cache memory has associated therewith one or more instruction flow control flags, wherein instruction flow is changed based upon the one or more instruction flow control flags associated with a cache line corresponding to the address when there is a comparison match between the selected information and the counterpart selected information.

17. The hot patch method according to claim **16**, wherein the one or more instruction flow control flags include an insert flag indicating that an opcode at the address within the cache memory is to be inserted prior to execution of an opcode at the address within the read only memory.

18. The hot patch method according to claim **16**, wherein the one or more instruction flow control flags include a replace flag indicating that an opcode at the address within the cache memory is to be executed instead of an opcode at the address within the read only memory.

19. The hot patch method according to claim **16**, wherein the one or more instruction flow control flags include a block flag indicating that more than one opcode starting at the address within the cache memory are to be executed instead of or before an opcode at the address within the read only memory.

20. The hot patch method according to claim **16**, wherein the one or more instruction flow control flags include a noop flag indicating that an opcode at the address within the read only memory is to be skipped.

21. A hot patch system, comprising:

a read only memory storing a plurality of instructions;

a cache memory storing alternate instructions for at least one instruction stored within the read only memory, each cache line within the cache memory having associated therewith one or more selection flags and one or more instruction flow control flags; and

cache control logic coupled to the cache memory, the cache control logic comparing selected information associated with an instruction stored at a specified address in the read only memory with counterpart selected information associated with an instruction stored at the specified address in the cache memory,

wherein the selected information and counterpart selected information are selected based upon the one or more selection flags associated with a cache line corresponding to the specified address, and

wherein the cache control logic causes a change in instruction flow based upon the one or more instruction flow control flags associated with the cache line corresponding to the address when there is a comparison match between the selected information and the counterpart selected information.

22. The hot patch system according to claim 21, wherein the one or more selection flags include:

a first flag indicating whether the cache control logic is to compare the specified address with an address of one or more instructions within the cache memory;

a second flag indicating whether the cache control logic is to compare an opcode stored at least at the specified address within the read only memory with an opcode stored at the specified address within the cache memory; and

a third flag indicating whether the cache control logic is to compare an opcode stored at any address within the read only memory with an opcode stored at the specified address within the cache memory.

23. The hot patch system according to claim 21, wherein the one or more instruction flow control flags include:

a replace flag indicating that an opcode at the specified address within the cache memory is to be executed instead of an opcode at the specified address within the read only memory;

a block flag indicating that more than one opcodes starting at the specified address within the cache memory are to be executed instead of or before an opcode at the specified address within the read only memory; and

a noop flag indicating that an opcode at the specified address within the read only memory is to be skipped.

24. A processor-implemented method of executing instructions, comprising:

storing an instruction address compare value at a first instruction address in a cache memory store configured to hold alternate instructions for at least one instruction stored within a read only memory, each cache line within the cache memory store having associated therewith one or more selection flags;

receiving the first instruction address for an instruction within the read only memory from a program counter at the read only memory and the cache memory store;

based on one or more selection flags associated with a cache line at the first instruction address within the cache memory store, selecting and comparing the first instruction address with the instruction address compare value; and

selectively executing at least one instruction from a second instruction address, or executing a trap, if the first instruction address matches the instruction address compare value.

25. The method of claim 24, further comprising executing a trap if a predetermined sub-set of the bits comprising the program counter value match a corresponding sub-set of bits in the instruction address compare value.

26. A processor-implemented method of executing instructions, comprising:

storing an instruction opcode compare value at a first instruction address in a cache memory store configured to hold alternate instructions for at least one instruction stored within a read only program memory, each cache line within the cache memory store having associated therewith one or more selection flags;

receiving the first instruction address from a program counter;

retrieving a first instruction opcode from the read only program memory at the first instruction address;

based on one or more selection flags associated with a cache line at the first instruction address within the cache memory store, selecting and comparing the first instruction opcode with the instruction opcode compare value; and

executing at least one instruction from a second instruction address if the first instruction opcode matches the instruction opcode compare value.

27. A processor-implemented method of executing instructions, comprising:

storing an instruction opcode compare value at a first instruction address in a cache memory store configured to hold alternate instructions for at least one instruction stored within a read only program memory, each cache line within the cache memory store having associated therewith one or more selection flags;

receiving the first instruction address from a program counter;

retrieving an instruction opcode from the read only program memory at the first instruction address;

based on one or more selection flags associated with a cache line at the first instruction address within the cache memory store, selecting and comparing the first instruction opcode with the instruction opcode compare value; and

selectively executing at least one instruction from a second instruction address, or executing a trap, if the instruction opcode matches the instruction opcode compare value.

28. An apparatus, comprising:

a processor;

a program counter;

a cache memory address store configured to store an instruction address compare value and alternate instructions for at least one instruction stored within a read only memory;

a configurable selection bit associated with each cache line within the cache memory address store;

an address comparator configurable to select and compare an address from the program counter for an instruction stored at the address in the read only memory with an instruction address compare value contained at the address in the cache memory address store based on the configurable selection bit; and

control logic operable to cause execution of an instruction from an instruction address different than the instruction address compare value contained in the memory address store if the instruction address compare value contained in the cache memory address store matches the value of the program counter if the selection bit is of a first value and operable to cause execution of a trap if the selection bit is of a second value.

29. The apparatus of claim 28, further comprising a register to store a value of the program counter.

30. The apparatus of claim 28, further comprising logic to compare only a subset of bits comprising the program counter value and the instruction address compare value to cause a trap to occur upon the match of corresponding subsets of bits.

31. An apparatus, comprising:

a processor;

a program counter;

a cache memory opcode store operable to store an instruction opcode compare value and alternate instructions for at least one instruction stored within a read only memory;

a configurable selection bit associated with each line within the cache memory opcode store;

an opcode comparator configurable to select and compare an opcode retrieved from an address specified by the program counter for an instruction stored at the address in the read only memory with the opcode compare value contained in the memory opcode store based on the

configurable selection bit associated within an instruction opcode at the address; and

control logic operable to cause execution of an instruction from an instruction address different than the value contained in the program counter if the instruction opcode compare value contained in the address store matches the value of a retrieved opcode if the selection bit is of a first value and operable to cause execution of a trap if the selection bit is of a second value.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

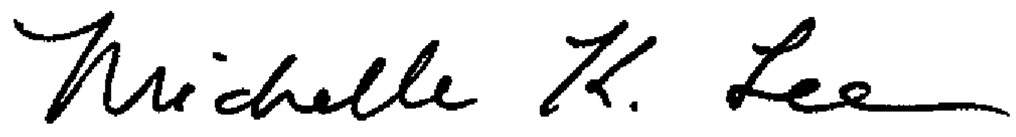
PATENT NO. : RE45,278 E
APPLICATION NO. : 12/914978
DATED : December 2, 2014
INVENTOR(S) : Kasper

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims

Column 7, Line 64, Claim 12	Delete: "he" Insert: --be--

Signed and Sealed this
Eighth Day of September, 2015



Michelle K. Lee
Director of the United States Patent and Trademark Office