PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/38 | A2

(11) International Publication Number: WO 97/30389

(43) International Publication Date: 21 August 1997 (21.08.97)

(21) International Application Number: PCT/US97/02184

(22) International Filing Date: 10 February 1997 (10.02.97)

(30) Priority Data:
08/601,744 15 February 1996 (15.02.96) US

(71) Applicant: HAL COMPUTER SYSTEMS, INC. [US/US]; 1315 Dell Avenue, Campbell, CA 95008 (US).

(72) Inventors: KULKARNI, Paritosh, M.; 2275 South Bascom Avenue #1504, Campbell, CA 95008 (US). REEVE, Richard; Apartment 19, 200 Towne Terrace, Los Gatos, CA 95030 (US). SAXENA, Nirmal, R.; 24390 Summerhill Avenue, Los Altos Hills, CA 94024 (US).

(74) Agents: PATEL, Rajiv, P. et al.; Fenwick & West L.L.P., Suite 700, Two Palo Alto Square, Palo Alto, CA 94306 (US).

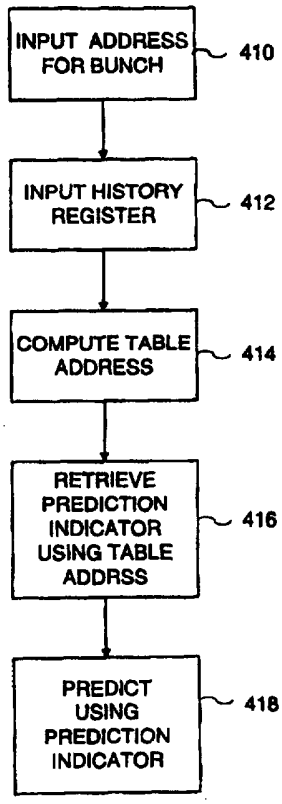
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR IMPROVED BRANCH PREDICTION ACCURACY IN A SUPERSCALER MICRO-PROCESSOR

(57) Abstract

Methods and apparatuses predict whether conditional branch computer instructions should be taken or not taken. A history register is maintained to record the history of groups of instructions, updated only once for each group. The history register and an address of one of the bytes of one of the instructions in each group are appended or otherwise combined to create an address to a table of two-bit saturating counters. The value of one of the bits of the counter at the address created is used for predicting all the conditional branch instructions for each branch in the group.



FOR THE PURPOSES OF INFORMATION ONLY

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

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

METHOD AND APPARATUS FOR IMPROVED BRANCH PREDICTION ACCURACY IN A SUPERSCALER MICROPROCESSOR

Field of Invention

The present invention relates to microprocessors and more specifically to the prediction of branch instructions in superscalar microprocessors.

Background of Invention

Conventional microprocessors which do not use a superscalar or multipipelined architecture accept instructions from a serial instruction stream, and process those instructions sequentially, in a logical order allowing jumps and branches. When a conditional branch instruction is encountered, the microprocessor tests certain flags which have been set by instructions previously executed by the microprocessor, and either resumes executing at the instruction which followed the conditional branch instruction in the serial instruction stream, or resumes execution at an instruction stored at a location described by the conditional branch instruction.

Superscalar microprocessors can accept a serial instruction stream, and produce the same results as a non superscalar microprocessor. However, superscalar microprocessors may internally process multiple instructions simultaneously, which may cause instructions to be executed out of their logical order, the order intended by the original creator of the instructions.

Referring now to Figure **1**, a conventional superscalar microprocessor **102** and memory **104** is shown. Fetch circuitry **106** directs memory **104** to transfer blocks of instructions **110, 112** starting with the memory address contained in the fetch program counter **108** a block at a time into memory area **114**, for simultaneous processing by execution units **116, 118, 120**. Although the size of the blocks **110, 112** and memory area **114** shown in Figure **1** are four words, and the number of execution units **116, 118, 120** shown in Figure **1** is three, conventional

superscalar microprocessors may have blocks **110, 112** and storage areas **114** of any size, and any number of execution units **116, 118, 120**.

Producing results in a superscalar microprocessor which are identical to the results which would be produced by a conventional non-superscalar microprocessor poses certain problems for a superscalar microprocessor. One problem posed by a superscalar microprocessor design arises in the processing of a conditional branch instruction. Because the instructions which set the flags in a superscalar microprocessor may not have been processed at the time the branch instruction is ready for execution by the superscalar microprocessor, it is impossible to determine with certainty which instruction the non-superscalar microprocessor would have executed after the execution of the conditional branch instruction without waiting for all instructions which logically precede the conditional branch instruction to execute. Waiting for all such preceding instructions to execute would introduce undesirable delays.

One approach to avoid these delays has been to attempt to predict the result of the conditional branch instruction without waiting for the logically preceding instructions to execute, and continue processing instructions as if the prediction was accurate. When the instructions which logically precede the conditional branch have all completed execution, the prediction may be tested for accuracy. If the result of the branch prediction is indeed accurate, processing continues and the undesirable delays are avoided. If the result of the branch prediction is inaccurate, processing stops, and resumes at the instruction which should have been executed after the conditional branch, with the delay no greater than if processing had suspended waiting for execution of the instructions which logically preceded the conditional branch instruction.

Various conventional ideas exist for predicting which branch direction to take. One approach is to always predict the branch described in the branch instruction will be taken. Such a prediction can often be correct more than fifty percent of the time, as many programs contain loop instructions that result in

the branch described in the branch instruction being taken more often than not. For example, the PASCAL instructions:

```
For i:=1 to 100 do begin
...
end;
```

cause the branch described in the branch instruction to be taken 99 percent of the time. Of course, other instructions, such as if..then, while..do, and repeat..until may not yield the same prediction accuracy, but the scheme is relatively simple to implement, saving valuable area in a superscalar microprocessor **102**.

When an instruction described in the conditional branch instruction is executed following the conditional branch instruction, the action is described as "taking the branch" and thus, the branch or "direction" of the branch is "taken". When the instruction which physically follows the branch instruction is executed because the conditions of the conditional branch instruction were not true, the action is described as "not taking the branch" and the branch or "direction" of the branch is described as "not taken."

One idea which can improve the accuracy of branch prediction is known as "bimodal" branch prediction and involves the use of a two-bit saturating counter as a prediction indicator to indicate whether a branch should be taken. A two bit saturating counter makes use of the assumption that branches should be taken in groups, and so the whether a branch or group of branches should be taken may be predicted by reference to whether the last branch or branches were taken. Referring now to Figures **2A** and **2B**, an illustration of a state table of a two-bit saturating counter is shown. State **210** represents a strong indication that the branch should not be taken. State **212** represents a weak indication that the branch should not be taken. State **214** represents a weak indication that the branch should be taken. State **216** represents a strong indication that the branch should be taken. The state of the prediction may be initialized to any state **210, 212, 214, 216**. The branch is

predicted taken if the most significant bit of the current prediction state has a value of "1", such as states **214, 216**, and the branch is not taken if the most significant bit in the current prediction state has a value of "0", such as states **210, 212**. When the prediction is tested after the instructions logically preceding the branch have been executed, the state of the prediction is changed according to table **218**. Column **220** represents the current state, column **222** represents the new state, and column **224** represents the actual branch action: taken, meaning the branch was actually taken, or not taken, meaning the branch was not actually taken. From a strong indication, two actual branches opposite the indication are required before a change is made to the branch prediction. Other arrangements of counters, including those with more than two bits, may be utilized to vary the number of actual branches opposite the strong indication required to change the prediction.

The states of Figure **2A** may also have the values opposite those shown: strong taken, weak taken, weak not taken and strong not taken for states **210, 212, 214, 216**, respectively. In this case, the most significant bit having a value of "1" indicates the branch should be predicted not taken, "0" indicates the branch should be predicted taken. Table **218** of Figure **2B** is used as described above, with the opposite actual actions in column **224**.

The accuracy of bimodal branch prediction may be enhanced through the use of a history register, which records the history of the actual branch action taken. The use of a history register assumes that conditional branches are taken according to repeating patterns. For example, in the following PASCAL program:

```
For i:= 1 to 100 do
  For j:= 1 to 3 do begin
...
end;
```

the inner branch will be taken two times, but not the third, followed by the outer branch taking its branch, behavior which will be repeated ninety-eight times due to the outer branch. Knowledge of the behavior of the last four branches of both the inner and outer branch can predict the behavior of the next branch with higher accuracy than bimodal branch prediction. A shift register may be used as a history register to keep track of the behavior of the branches by shifting bits one position in a single direction (right or left) for each branch encountered, shifting in a "1" for each branch that is actually taken, and shifting in a "0" for each branch that is not actually taken. For example, a left shift register would read 1101 after the outer branch was taken, with the zero in the second least significant position showing that the end of the inner loop had been reached. The next branch should be predicted taken, as it will be the first branch in the next iteration of the inner loop.

The history register is used with a history table and the two-bit saturating counters of bimodal branch prediction to complete the prediction. Referring now to Figure **3**, the contents of a history register **308** as described above are used as an index to a history table **312**. The pointer **316** having the same index **314** as that of history register **308** points to a two-bit saturating counter **318, 320, 322, 324, 326** having a state table as described above with reference to Figure **2A** which is used to determine the branch prediction as described above. The entire history register **308** may be used as an index to the table **312**, or a certain number of bits including and adjacent to the bit most recently shifted in to the history register **308** may be used as an index to the table **312**.

Another method is similar to the history table method described above, except that the address of all, or a certain number of the least significant bits, of the address of the conditional branch instruction are used in place of the history register **308** as the index to the table **312**.

Still other methods combine the low order bits of the address of the conditional branch instruction and some or all of

the branch history, for example by concatenation or exclusive-OR-ing, to create an index to the table **312**, in place of the history register **308** alone.

Referring again to Figure **1**, if the address of the conditional branch instruction is used to create the index, the address must be computed from the fetch program counter register **108** and the position of the conditional branch instruction in the memory **114**, causing added complexity of the microprocessor **102** and computational delay. If the history is used to create the index, it must be updated for each conditional branch instruction executed, resulting in additional complexity in the design of the microprocessor **102**.

Summary of Invention

A method and apparatus predicts whether each conditional branch instruction in a bunch of instructions retrieved from a block of memory should be taken using a table of pointers to an array of two-bit saturating counters. For each conditional branch instruction in the bunch, the index to the table is derived from the least significant bits of an address of the same one of the bytes in the block appended to, or otherwise combined with a history register, which is updated only once for the bunch by shifting in a "1" if any of the branches in the bunch were actually taken, "0" otherwise. Because the index does not require the computation of the exact memory location of each conditional branch instruction, the time and complexity required to determine the index is reduced. Because the history table is updated only once for the bunch, instead of once for each conditional branch instruction in the bunch, complexity is further reduced.

Brief Description of the Drawings

Figure **1** is a block schematic diagram of a conventional superscalar microprocessor and a conventional memory.

Figure **2A** is a state diagram of a conventional two-bit saturating counter.

Figure **2B** is a state table illustrating the operation of a conventional two-bit saturating counter described by Figure **2A**.

Figure **3** is a block schematic diagram of a conventional branch predictor utilizing a table.

Figure **4A** is a flowchart illustrating a method of predicting a conditional branch according to one embodiment of the present invention.

Figure **4B** is a flowchart illustrating a method of updating a prediction indicator according to one embodiment of the present invention.

Figure **4C** is a flowchart illustrating a method of updating a history register according to one embodiment of the present invention.

Figure **5** is a block schematic diagram of conditional branch prediction circuitry in a superscalar microprocessor according to one embodiment of the present invention.

Detailed Description of a Preferred Embodiment

Referring now to Figure **4A**, a flowchart illustrating a method of predicting a conditional branch instruction in a bunch of several instructions according to the present invention is shown. A portion or all of a storage address for one of the instructions in the bunch is identified **410**. In one embodiment, this address is the memory address of the first byte of the first instruction of the bunch of instructions. Other embodiments use the address of other bytes of the first or other instructions in the bunch, or another identifier unique to the bunch.

Some or all of a history register is retrieved **412** for use as described below. The history register may be retrieved **412** after, prior to, or at substantially the same time as the storage address is retrieved **410**. The history register may be initialized to any value, such as all zeros, and updated as described in Figure **4B** below.

A prediction table address is computed using the history register and the storage address of one of the instructions in the bunch **414**. In one embodiment, the table address is a concatenation formed by placing the entire history register in the most or least significant table address bit positions and a certain number of least significant bits of the storage address of one of the instructions in the bunch in the remaining table address bit positions. In another embodiment, the table address is a concatenation of a certain number of the history register bits in the most or least significant table address bit positions and a certain number of least significant bits of the storage address of one of the instructions in the bunch in the remaining table address bit positions. In one embodiment, the history register is updated by shifting a bit into the least significant bit position of the history register, the four least significant bits of the history register are placed into the four most significant table address bit positions, and the eight least significant bits of the address of the first instruction in the bunch of instructions are placed into the remaining eight least significant table address bits to form a twelve bit table address.

In one of the embodiments described above, the bits of the address of one of the bytes of the instructions in the bunch to be concatenated are the least significant bits of the address and the bits of the history register to be concatenated are the history register bits including and adjacent to the bit most recently shifted in.

In another embodiment, the history register is used without the address of one of the bytes of the instructions in the bunch to compute the table address. In another embodiment, the address of one of the bytes of the instructions in the bunch is used without the history register to compute the table address.

The table address can also be computed using the history register and the address of one of the bytes of an instruction in the bunch using methods other than concatenation. In another embodiment, some or all of the bits of the history register and some or all of the bits of the address of one of the bytes of

the instructions in the bunch are exclusive-OR-ed to create a history table. In one embodiment, the bits of the address of one of the bytes of the instructions in the bunch to be exclusive-OR-ed are the least significant bits of the address, and the bits of the history register to be exclusive-OR-ed are the history register bits including and adjacent to the bit most recently shifted in.

The table address may then be used to retrieve a portion or all of a prediction indicator **416**. In one embodiment, the prediction indicator is located at the table address in a table of prediction indicators. In another embodiment, the prediction indicator is located via a pointer at the table address. The prediction is made according to the prediction indicator retrieved. In one embodiment, each prediction indicator acts as a two-bit counter, such as the two-bit saturating counter described above and having the state table illustrated in Figure **2A**, with the conditional branch predicted taken if the most significant bit of the two-bit saturating counter corresponding to the table address has one value such as a "1" **214, 216** and predicted not taken if the most significant bit of the two-bit saturating counter corresponding to the table address has the opposite value, such as a "0" **210, 212**. A single prediction derived using this method may be performed once and used for every conditional branch in the bunch.

Optionally, the prediction indicator may be updated based on whether any branch in the bunch was actually taken, using a method such as the method described above using Figure **2B**. Referring now to Figures **4B** and **2B,** if any branch in the bunch is actually taken, the state of the prediction indicator is updated using the lower portion **228** of table **213**. If no branches in the bunch are actually taken, the prediction indicator is updated using the upper portion **226** of table **218**.

In one embodiment, the history register is updated once for each bunch of instructions, after all predictions for branch instructions in the bunch have been made. In one embodiment, the history register is updated as shown in Figure **4C**. If any branch in the bunch was actually taken **430**, a value such as a

"1" is shifted into the history register **432**, otherwise, if no branches in the bunch were actually taken, the opposite value, such as a "0" is shifted into the history register **434**. Bits may be shifted into the history register from either direction as long as the direction of shifts is consistent among a large number of bunches.

In one embodiment, the history register is only updated if there is a conditional branch instruction in the bunch **436**. This embodiment allows a history register of a certain size to track a longer history than a history register which is updated even for bunches which do not contain conditional branch instructions as described above.

Referring now to Figure **5**, one embodiment of an apparatus used to predict conditional branch instructions according to the present invention is shown. Fetch program counter **508** holds the storage address of the first instruction of a bunch of instructions stored in a storage device such as a memory, not shown on Figure **5**, but similar to memory **104** of Figure **1**. Retriever **504** addresses such storage device via address bus **506** to retrieve a certain number of instruction bytes from such storage device via data bus **509** into memory **510**. Execution unit loader **502** loads instructions stored in memory **510** into execution units **512**. Next instruction decode **516** appends a tag into tag storage **518** if the instruction stored in memory **510** following the instruction loaded into execution unit **512** is a conditional branch instruction. The tag stored in tag storage **518** is made up of a number of bits, each bit corresponding to the condition or conditions which must be met for the conditional branch instruction stored in memory **510** following the instruction in the respective execution unit **512** to take the branch. One or more execution units **512** each contain a flag register **514** which sets condition flag bits in the flag register **514** based upon the results produced by the execution unit **512**. Each bit in flag register **514** corresponds to conditions such as "result = 0" "result > 0" or "result < 0". Result compare **520** compares the flag bits in flag register **514** with the tag bits stored in tag storage **518**. If all of the tag bits in tag storage **518** match the corresponding set flag bits in flag

register **514**, result compare **520** outputs true to history latch **521**, indicating that the conditions for one of the conditional branches in the bunch to take the branch has actually occurred. History latch **521** is shifted into history register **522**, after all of the instructions preceding conditional branch instructions in the bunch have been executed by execution units **512**.

FPC latch **511** is coupled to the fetch program counter **508** to preserve the value of some or all of the bits contained in the fetch program counter **508**. History register **522** is concatenated with FPC latch **511** in the combination or fashion as described above via concatenator **524** to address table RAM **526** which contains a table of two-bit saturating counters as described above. In one embodiment concatenator **524** concatenates bits of the history register **522** and the bunch address contained in FPC latch **511**. In another embodiment, concatenator **524** exclusive-ORs the bits as described above. The most significant bit of the two-bit saturating counters of table RAM **526** is output on output line **528** to branch execution unit **532** to indicate to the branch execution unit **532** whether to take any branch in the bunch loaded into memory **510** as described above.

Update unit **530** is coupled to history latch **521** and table RAM output **529** to receive both bits stored in the two-bit saturating counter addressed by concatenator **524**. Update unit **530** updates the two-bit saturating counter in table RAM **526** using the values illustrated in Figure **2B**. Update unit **530** updates the table RAM after all non-branch instructions stored in memory **510** corresponding to the instructions loaded by the fetch program counter **508** corresponding to the bits stored in FPC latch **511**. This means the update unit **530** updates a single address of table RAM **526** corresponding to the table RAM address output by concatenator **524** once for each time instructions are loaded into memory **510**. Other arrangements for loading memory **510** are possible, such as a double buffer arrangement whereby pointers are used to correspond to memory **510**, one pointer points to the next location in memory **510** into which instructions are to be loaded and one pointer points to the next

location in memory **510** from which instructions are to be transferred to execution units **512**. In such an arrangement, update unit **530** updates the two-bit counter in table RAM **526** corresponding to the address at the output of concatenator **524** one time for each time all instructions which had been in memory **510** at one time have been executed.

What is claimed is:

1. A method of locating in an addressable table of a plurality of prediction indicators a prediction indicator indicating a direction of a conditional branch instruction in a bunch of instructions comprising a plurality of instructions, each instruction having at least one unique identifier, the method comprising:

selecting at least a portion of a unique identifier of one of the instructions in the bunch other than the conditional branch instruction;

building an address for the table of prediction indicators using the portion of the unique identifier selected; and

addressing the table of prediction indicators using the address built.

2. The method of claim 1 wherein at least one unique identifier of each instruction in the bunch comprises a storage address corresponding to said each instructions.

3. The method of claim 2 wherein each instruction comprises at least one byte and the storage address corresponding to each instruction is a storage address of a byte of said each instructions.

4. The method of claim 1 wherein the building step comprises appending the portion of the unique identifier selected with a number of bits of a history register comprising a shift register.

5. The method of claim 4 wherein:

the unique identifier comprises a number of bits comprising a least significant bit and having an order;

the portion of the unique identifier comprises a group of eight bits of the unique identifier adjacent to and including the least significant bit;

the history register bits have an order and comprise a bit most recently shifted in; and

the number of bits of the history register is four, said number of bits of the history register comprising the bit most recently shifted in and three bits adjacent to the bit most recently shifted in.

6. A method of updating a history register comprising a plurality of bits, each having a first value and a second value, to maintain the conditional branch history of one of a plurality of bunches of instructions comprising a plurality of instructions, at least one bunch of instructions including at least one conditional branch instruction, each conditional branch instruction having an actual taken state being selectable from a first state and a second state, the method comprising:

determining the presence of at least one of the conditional branch instructions in the bunch having an actual taken state of the first state;

responsive to the presence of at least one of the conditional branch instructions in the bunch having an actual taken state of the first state, shifting a first value into the history register; and

responsive to the absence of at least one conditional branch instructions in the bunch having an actual taken state of the first state, shifting a second value into the history register.

7. The method of claim 6 comprising the additional steps of, for at least one instruction in the bunch:

determining whether a conditional branch instruction was actually taken;

responsive to the conditional branch instruction having been actually taken, selecting the actual taken state of the first state; and

responsive to the conditional branch instruction having not actually been taken, selecting the actual taken state of said branch of the second state.

8. A method of updating a history register comprising a plurality of bits having a first value and a second value to maintain the conditional branch history of one of a plurality of bunches of instructions comprising a plurality of instructions, at least one bunch of instructions including a plurality of conditional branch instructions, each conditional branch instruction having an actual taken state being selectable from a first state and a second state, the method comprising:

determining the presence of a conditional branch instruction in the bunch; and

responsive to the presence of a conditional branch instruction in the bunch:

determining the presence of one of the conditional branch instructions in the bunch having an actual taken state of the first state;

responsive to the presence of at least one of the conditional branch instructions in the bunch having an actual taken state of the first state, shifting a first value into the history register; and

responsive to the absence of at least one conditional branch instructions in the bunch having an actual state of the first state, shifting a second value into the history register.

9. The method of claim 8 comprising the additional steps of, for at least one instruction in the bunch:

determining whether a conditional branch instruction was actually taken;

responsive to the conditional branch instruction having been actually taken, selecting the actual taken state of the first state; and

responsive to the conditional branch instruction having not actually been taken, selecting the actual taken state of said branch of the second state.

10. An apparatus for predicting the direction of at least one conditional branch instruction in a bunch of a plurality of conditional branch instructions having at least one branch criteria and non-conditional branch instructions in a plurality of bunches of instructions, each bunch having a unique identifier, the apparatus having an output for indicating the direction predicted, the apparatus comprising:

a fetch program counter latch having an output, the fetch program counter latch for storing at least a portion of a unique identifier for the bunch of instructions;

a concatenator having a first input coupled to the fetch program counter latch and an output; and

an addressable storage device having a data input, an address input coupled to the concatenator output and a data output coupled to the apparatus output, the addressable storage device containing at least one prediction indicator in at least one addressable storage location.

11. The apparatus of claim 10 additionally comprising:

a history register comprising a shift register and having an output comprising a most-recently-shifted-in bit of the shift register and at least one additional bit of the shift register; and

wherein, the concatenator additionally comprises a second input coupled to the history register output.

12. The apparatus of claim 11 wherein the concatenator output comprises a number of fetch program counter latch bits, the most recently-shifted-in history register output bit and at least one of the additional bits of the history register output.

13. The apparatus of claim 12 wherein the number of fetch program counter latch bits is eight, and the concatenator output comprises three additional bits of the history register output.

14. The apparatus of claim 11 wherein the concatenator output comprises the first concatenator input exclusive-OR-ed with the second concatenator input.

15. The apparatus of claim 10 wherein each instruction in the bunch has a storage address and the unique identifier for each bunch of instructions comprises a storage address of at least one of the instructions in the bunch.

16. The apparatus of claim 15 wherein the instruction corresponding to the storage identifier for the bunch is a non-conditional branch instruction.

17. The apparatus of claim 10 additionally comprising:

at least one register unit having an output and an input coupled to receive at least one of the instructions and comprising an execution unit and at least one result compare having an output coupled to the register unit output and being state selectable from a first state responsive to said instructions preceding a conditional branch instruction and having at least one branch criteria of said conditional branch instruction and a second state responsive to said instructions not preceding a conditional branch instruction or not having at least one branch criteria of a conditional branch instruction following said instruction;

a result compare latch having at least one input coupled to at least one register unit output, and an output being state selectable from a first state responsive to at least one of the result compare latch inputs being in the first state and a second state responsive to none of the result compare latch inputs being in the first state; and

an update unit having a first input coupled to the result compare latch output, a bunch of second inputs coupled to at least one of the addressable storage device data outputs and an

output coupled to the addressable storage device data input and having a plurality of selectable states responsive to the first input and the second set of second inputs.

18. The apparatus of claim 17 wherein:

the update unit output is state selectable from a first state, a second state, a third state and a fourth state; and

the addressable storage device outputs coupled to the update unit set of second inputs have a first state, a second state, a third state and a fourth state.

19. The apparatus of claim 18 wherein:

responsive to the result compare latch in the first state, the update unit selects the update unit output to the:

first state responsive to the update unit in the first state or the second state;

second state responsive to the update unit in the third state; and

third state responsive to the update unit in the fourth state; and

responsive to the result compare latch in the second state, the update unit selects the update unit output to the:

second state responsive to the update unit in the first state;

third state responsive to the update unit in the second state; and

fourth state responsive to the update unit in the third or fourth state.

20. The apparatus of claim 18 wherein:

responsive to the result compare latch in the second state, the update unit selects the update unit output to the:

first state responsive to the update unit in the first state or the second state;

second state responsive to the update unit in the third state; and

third state responsive to the update unit in the fourth state; and

responsive to the result compare latch in the first state, the update unit selects the update unit output to the:

second state responsive to the update unit in the first state;

third state responsive to the update unit in the second state; and

fourth state responsive to the update unit in the third or fourth state.

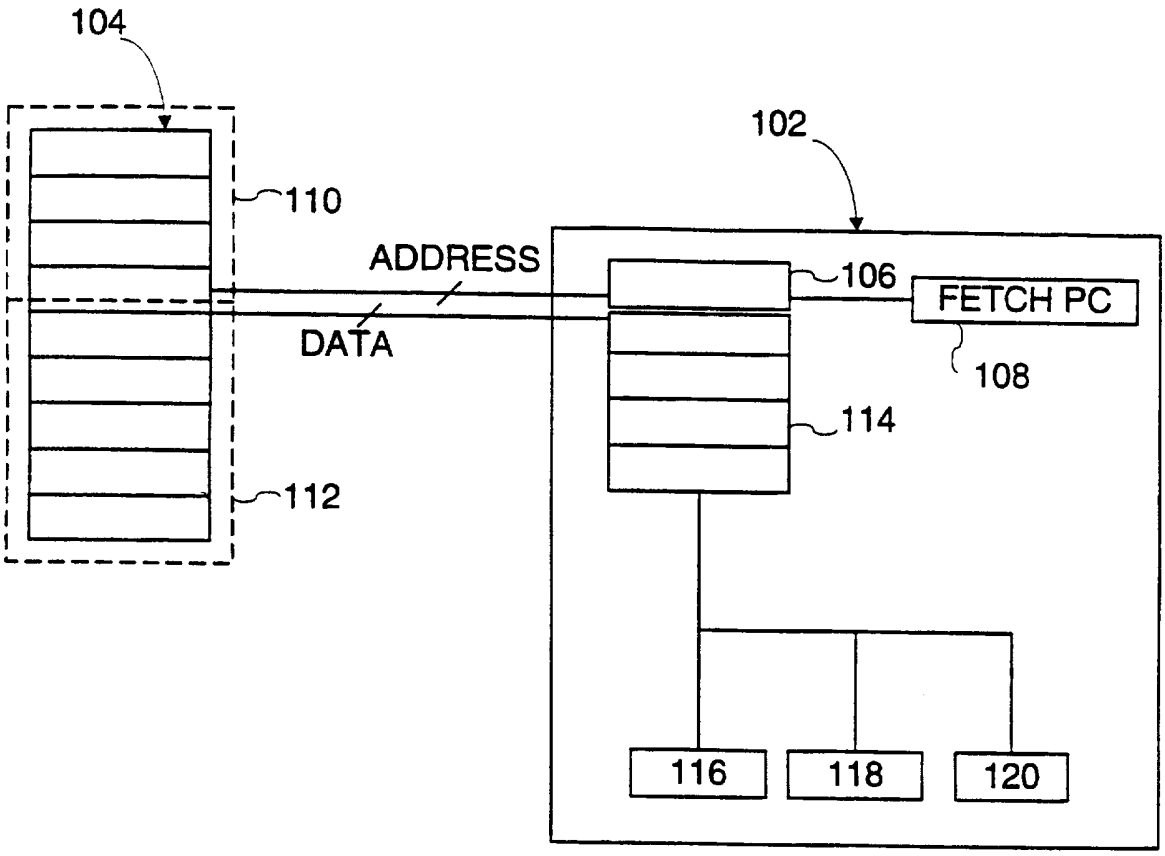


FIG. 1
(PRIOR ART)

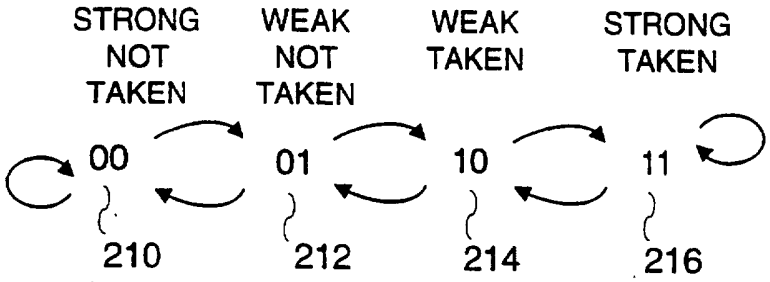


FIG. 2A
(PRIOR ART)

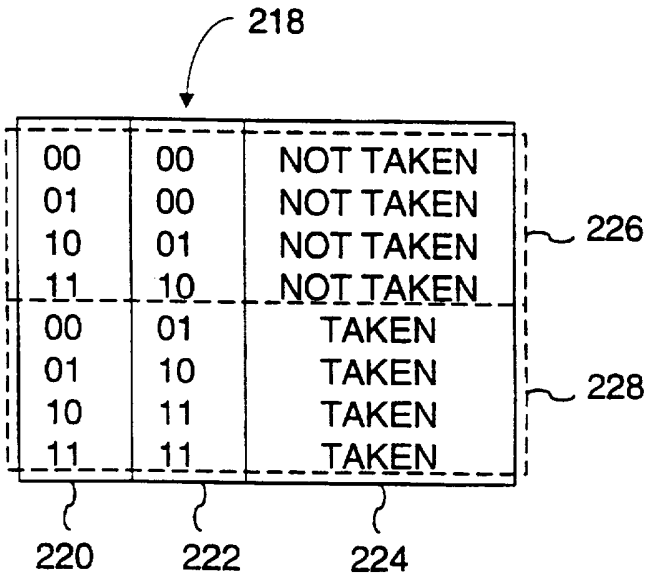


FIG. 2B
(PRIOR ART)

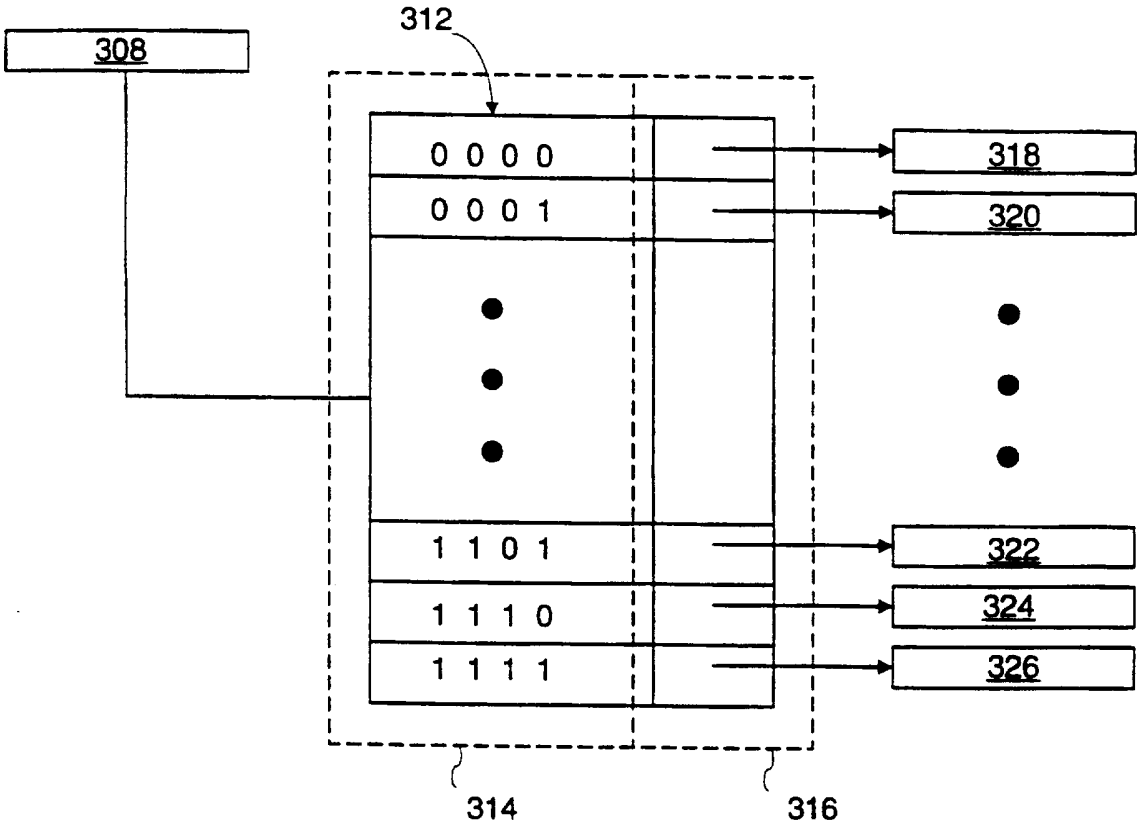


FIG. 3
(PRIOR ART)

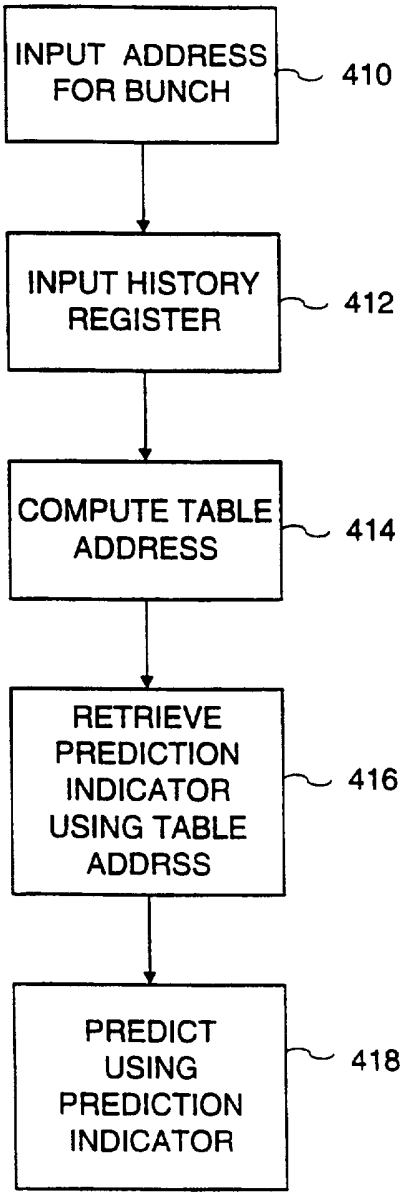

INPUT ADDRESS FOR BUNCH
410
INPUT HISTORY REGISTER
412
COMPUTE TABLE ADDRESS
414
RETRIEVE PREDICTION INDICATOR USING TABLE ADDRSS
416
PREDICT USING PREDICTION INDICATOR
418

FIG. 4A

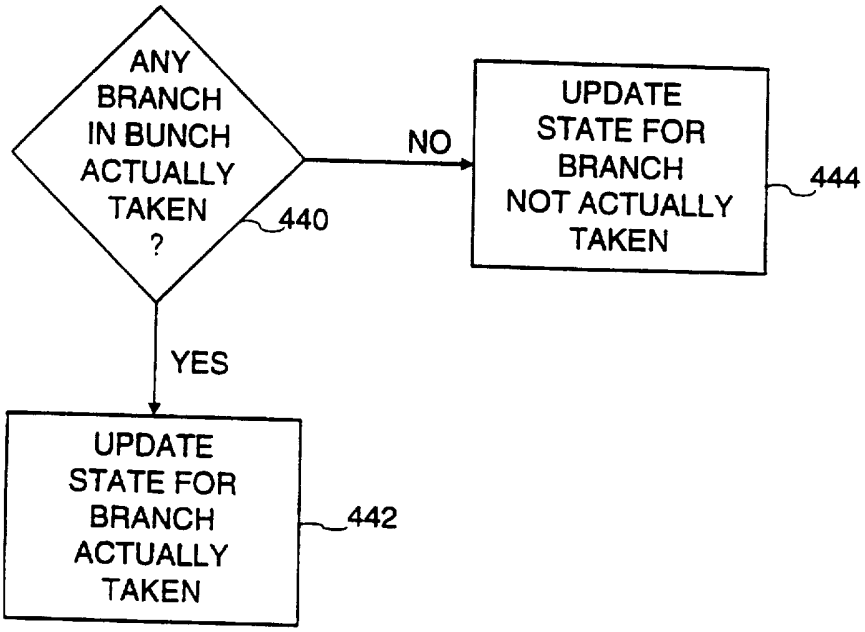

ANY BRANCH IN BUNCH ACTUALLY TAKEN ?
440
NO
UPDATE STATE FOR BRANCH NOT ACTUALLY TAKEN
444
YES
UPDATE STATE FOR BRANCH ACTUALLY TAKEN
442

FIG. 4B

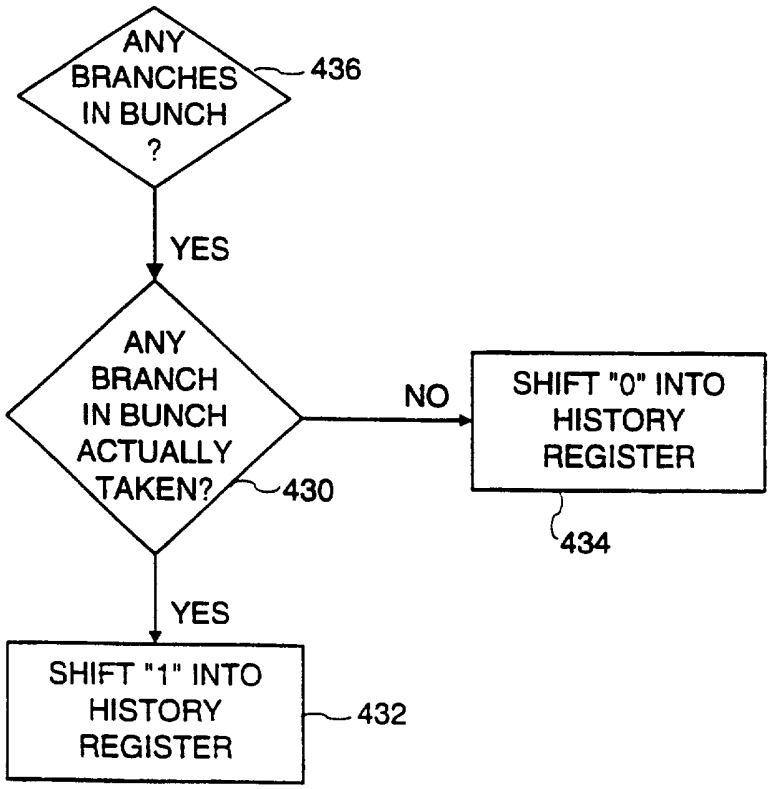

ANY BRANCHES IN BUNCH ?
436
YES
ANY BRANCH IN BUNCH ACTUALLY TAKEN?
430
NO
SHIFT "0" INTO HISTORY REGISTER
434
YES
SHIFT "1" INTO HISTORY REGISTER
432

FIG. 4C

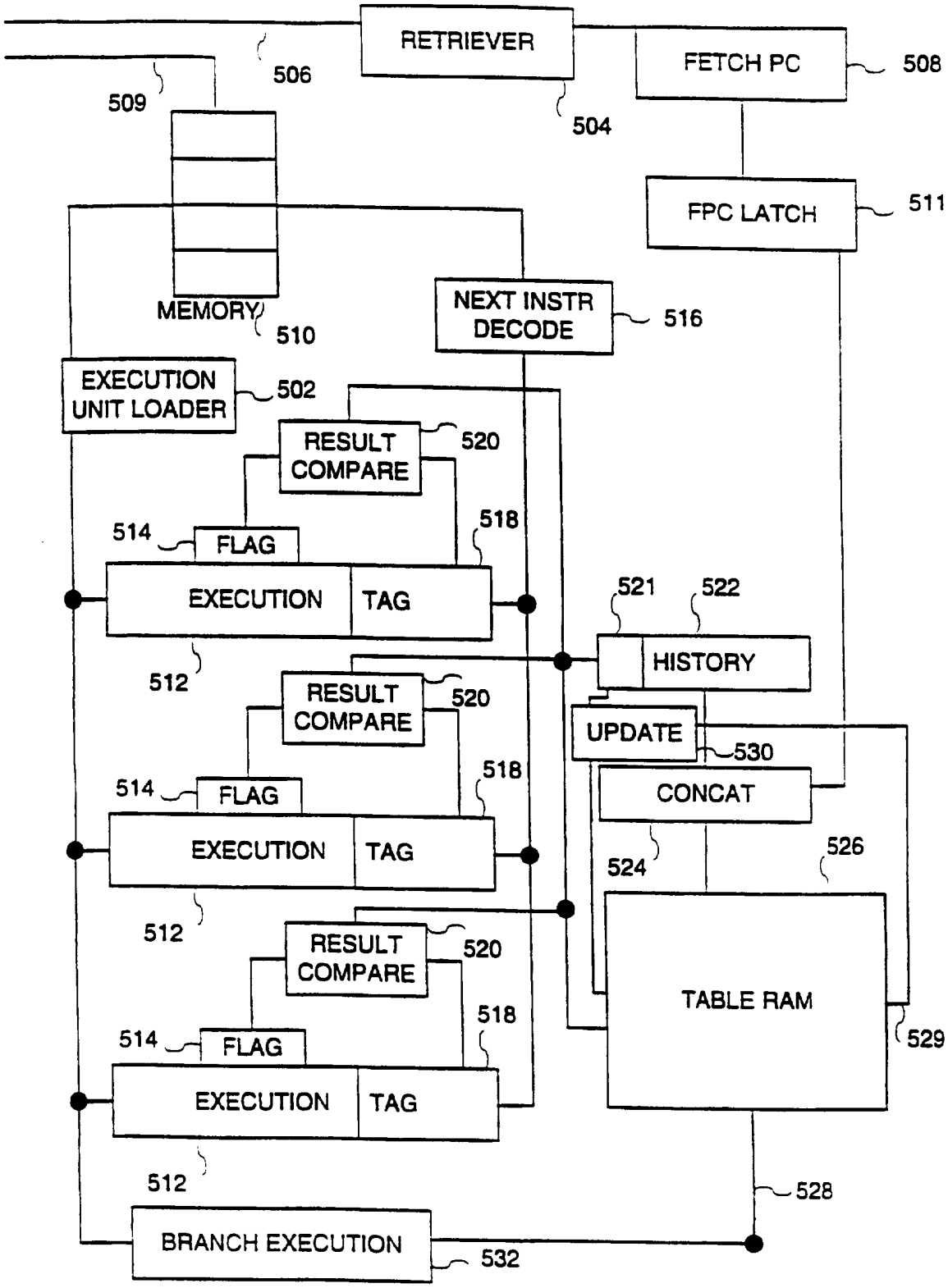

RETRIEVER
506
504
FETCH PC
508
509
FPC LATCH
511
MEMORY
510
NEXT INSTR DECODE
516
EXECUTION UNIT LOADER
502
RESULT COMPARE
520
514
FLAG
518
EXECUTION
TAG
521
522
HISTORY
512
UPDATE
530
CONCAT
524
526
TABLE RAM
529
528
BRANCH EXECUTION
532

FIG. 5