Advanced Computer Architecture

Dynamic Instruction Level
Parallelism

Going Faster – Dynamic Methods

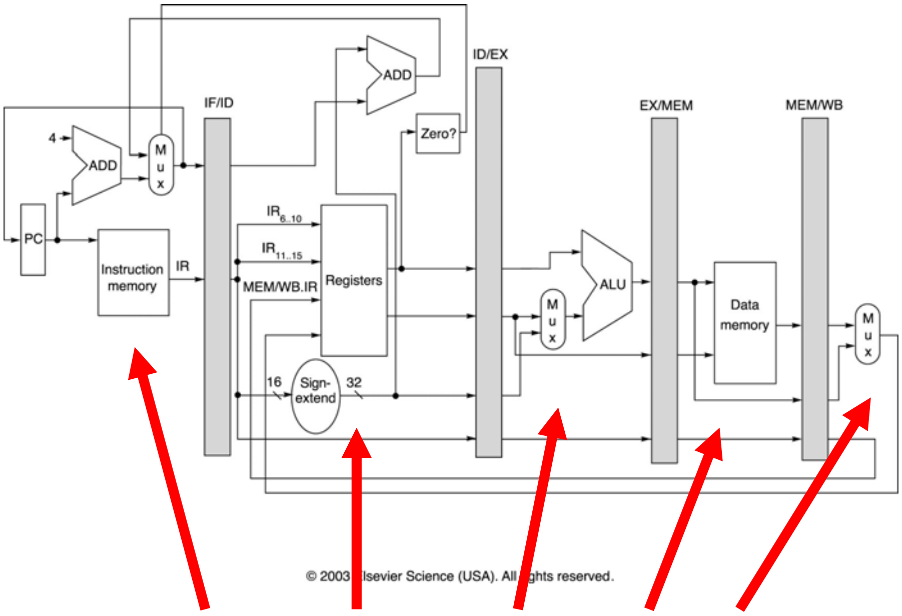
- Dynamic Scheduling
- Branch Prediction

First Class

- Multiple Issue
- Speculation

Second Class

Go Faster –
Execute Multiple Instructions



© 2003 Elsevier Science (USA). All rights reserved.

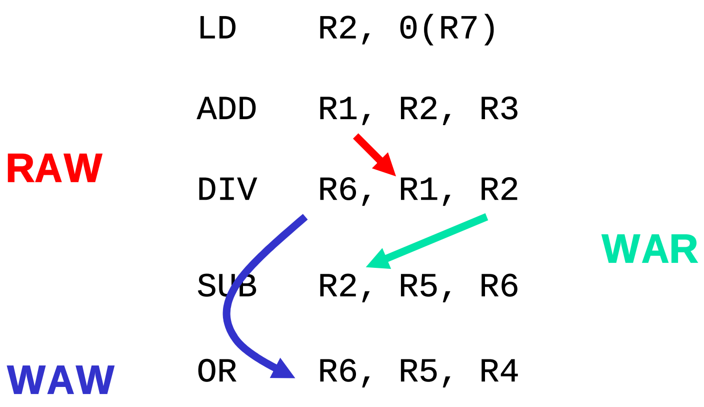
Five instructions execute at once

Limits to Performance

why instructions aren't parallel

- Structural Hazards
  - Insufficient resources
- Data Hazards
  - Data values
  - Name
- Control Hazards

Data Hazards



```
LD      R2, 0(R7)
ADD     R1, R2, R3
DIV     R6, R1, R2
SUB     R2, R5, R6
OR      R6, R5, R4
```

RAW

WAR

WAW

Pipeline blocking

```
LD      R2, 0(R7)
ADD     R1, R2, R3
DIV     R6, R1, R2
SUB     R8, R5, R4
SW      R8, 4(R7)
```

Instead of stalling, we could let SUB and SW execute

Issues with simple pipeline

- In-order issue, execute, completion
- Problem for machines with
  - Long memory latency
  - Slow instructions
  - Limited number of registers
- Can we get around these issues?

Dynamic Scheduling

- Tomasulo's algorithm
  - IBM 360/91 had limits
    - No cache
    - Slow FP operations
    - Only 4 double FP registers
- Change assumptions
  - Out of order completion
  - Reservation stations

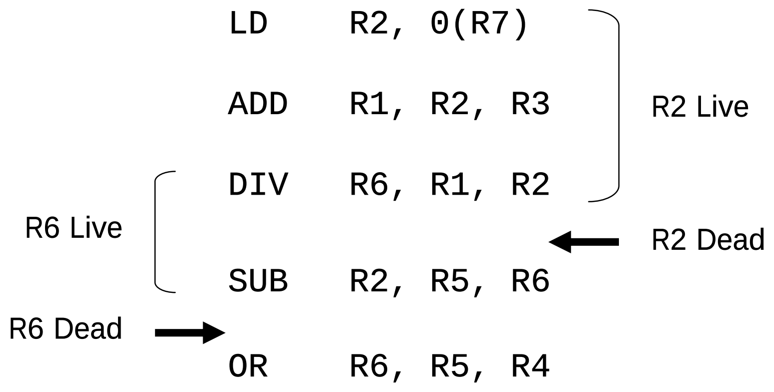
Processor characteristics

- Multiple function units
  - Multiply/divide won’t block add/sub
- Long pipelines
  - Hide latency of long operations
- Reservation stations
  - Issue only when data is ready
  - Effectively adds extra registers

Observations

- Some code sequences are independent
  - Order of execution may not effect result
- Registers have two uses
  - Fast access to program variables
  - Temporaries
- Registers not needed after they are dead

Live vs. Dead



```
LD      R2, 0(R7)
ADD     R1, R2, R3
DIV     R6, R1, R2
SUB     R2, R5, R6
OR      R6, R5, R4
```

R2 Live

R2 Dead

R6 Live

R6 Dead

Use temp registers

```
LD       S, 0(R7)
ADD     R1, S, R3
DIV     T, R1, S
SUB     R2, R5, T
OR      R6, R5, R4
```

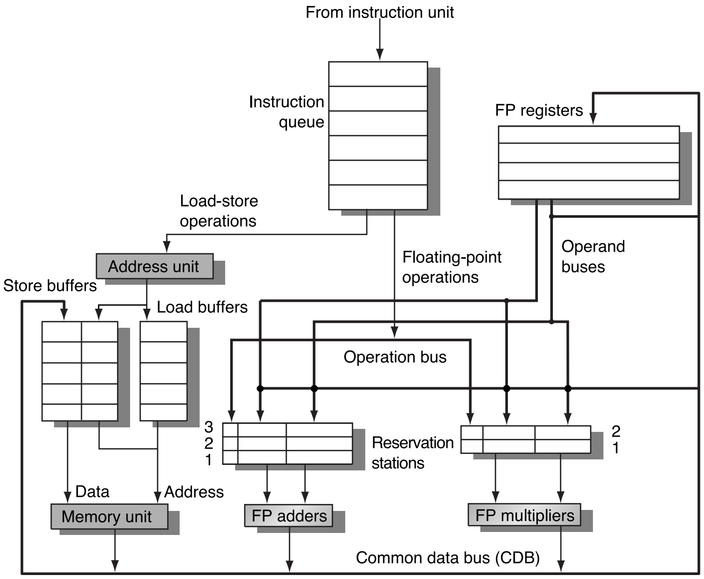
Implementation

- Separate Pipeline into phases
  - Instruction Issue
  - Execute
  - Write result
- Queue instructions at Execution Units until ready to run, with temp registers
  - If data not available, hold pointer to data
  - Issue in program order *within* function units
- Single Writeback bus (CDB)
  - Analogous to bypass paths
  - Local detect and update

Reservation Stations

- Each reservation station has:
  - OP – operation
  - Qj, Qk – address of source operands j and k
  - Vj, Vk – value of source operands j and k
  - A – memory address for loads/stores
  - Busy – indicates station is occupied
- Each architectural register has
  - Qi – address of reservation station that will write this register

Tomasulo FP Unit



© 2003 Elsevier Science (USA). All rights reserved.

Example

```
L.D         F6, 34(R2)
L.D         F2, 45(R3)
MUL.D       F0,F2,F4
SUB.D       F8,F2,F6
DIV.D       F10,F0,F6
ADD.D       F6,F8,F2
```

Execution

Instruction		I	E	W
L.D	F6,34(R2)			
L.D	F2,45(R3)			
MUL.D	F0,F2,F4			
SUB.D	F8,F2,F6			
DIV.D	F10,F0,F6			
ADD.D	F6,F8,F2			

Register	Qi
F0	
F2	
F4	
F6	
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	no						
Mult2	no						

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓		
L.D	F2,45(R3)			
MUL.D	F0,F2,F4			
SUB.D	F8,F2,F6			
DIV.D	F10,F0,F6			
ADD.D	F6,F8,F2			

Register	Qi
F0	
F2	
F4	
F6	Load1
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					34(R2)
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	no						
Mult2	no						

Execution

Instruction	I	E	W
L.D F6,34(R2)	✓	✓	
L.D F2,45(R3)	✓		
MUL.D F0,F2,F4			
SUB.D F8,F2,F6			
DIV.D F10,F0,F6			
ADD.D F6,F8,F2			

Register	Qi
F0	
F2	Load2
F4	
F6	Load1
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					34(R2)
Load2	yes	Load					45(R3)
Add1	no						
Add2	no						
Add3	no						
Mult1	no						
Mult2	no						

Execution

Instruction	I	E	W
L.D F6,34(R2)	✓	✓	
L.D F2,45(R3)	✓	✓	
MUL.D F0,F2,F4	✓		
SUB.D F8,F2,F6			
DIV.D F10,F0,F6			
ADD.D F6,F8,F2			

Register	Qi
F0	Mult1
F2	Load2
F4	
F6	Load1
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					34(R2)
Load2	yes	Load					45(R3)
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL		Regs[F4]	Load2		
Mult2	no						

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	
L.D	F2,45(R3)	✓	✓	
MUL.D	F0,F2,F4	✓		
SUB.D	F8,F2,F6	✓		
DIV.D	F10,F0,F6			
ADD.D	F6,F8,F2			

Register	Qi
F0	Mult1
F2	Load2
F4	
F6	Load1
F8	Add1
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					34(R2)
Load2	yes	Load					45(R3)
Add1	Yes	SUB			Load2	Load1	
Add2	no						
Add3	no						
Mult1	yes	MUL		Regs[F4]	Load2		
Mult2	no						

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	
L.D	F2,45(R3)	✓	✓	
MUL.D	F0,F2,F4	✓		
SUB.D	F8,F2,F6	✓		
DIV.D	F10,F0,F6	✓		
ADD.D	F6,F8,F2			

Register	Qi
F0	Mult1
F2	Load2
F4	
F6	Load1
F8	Add1
F10	Mult2

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					34(R2)
Load2	yes	Load					45(R3)
Add1	yes	SUB			Load2	Load1	
Add2	no						
Add3	no						
Mult1	yes	MUL		Regs[F4]	Load2		
Mult2	no	DIV			Mult1	Load1	

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	✓
L.D	F2,45(R3)	✓	✓	
MUL.D	F0,F2,F4	✓		
SUB.D	F8,F2,F6	✓		
DIV.D	F10,F0,F6	✓		
ADD.D	F6,F8,F2	✓		

Register	Qi
F0	Mult1
F2	Load2
F4	
F6	Add2
F8	Add1
F10	Mult2

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	Load					45(R3)
Add1	yes	SUB		Mem[1]	Load2		
Add2	yes	ADD			Add1	Load2	
Add3	no						
Mult1	yes	MUL		Regs[F4]	Load2		
Mult2	yes	DIV		Mem[1]	Mult1		

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	✓
L.D	F2,45(R3)	✓	✓	✓
MUL.D	F0,F2,F4	✓		
SUB.D	F8,F2,F6	✓		
DIV.D	F10,F0,F6	✓		
ADD.D	F6,F8,F2	✓		

Register	Qi
F0	Mult1
F2	
F4	
F6	Add2
F8	Add1
F10	Mult2

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	yes	SUB	Mem[2]	Mem[1]			
Add2	yes	ADD		Mem[2]	Add1		
Add3	no						
Mult1	yes	MUL	Mem[2]	Regs[F4]			
Mult2	yes	DIV		Mem[1]	Mult1		

Execution

Instruction		I	E	W
`L.D`	`F6,34(R2)`	✓	✓	✓
`L.D`	`F2,45(R3)`	✓	✓	✓
`MUL.D`	`F0,F2,F4`	✓	✓	
`SUB.D`	`F8,F2,F6`	✓	✓	
`DIV.D`	`F10,F0,F6`	✓		
`ADD.D`	`F6,F8,F2`	✓		

Register	Qi
`F0`	`Mult1`
`F2`	
`F4`	
`F6`	`Add2`
`F8`	`Add1`
`F10`	`Mult2`

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	no						
Add2	yes	ADD		Mem[2]	Add1		
Add3	no						
Mult1	no						
Mult2	yes	DIV		Mem[1]	Mult1		

Execution

Instruction		I	E	W
`L.D`	`F6,34(R2)`	✓	✓	✓
`L.D`	`F2,45(R3)`	✓	✓	✓
`MUL.D`	`F0,F2,F4`	✓	✓	✓
`SUB.D`	`F8,F2,F6`	✓	✓	
`DIV.D`	`F10,F0,F6`	✓		
`ADD.D`	`F6,F8,F2`	✓		

Register	Qi
`F0`	
`F2`	
`F4`	
`F6`	`Add2`
`F8`	`Add1`
`F10`	`Mult2`

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	yes	ADD		Mem[2]	Add1		
Add3	no						
Mult1	no						
Mult2	yes	DIV	F0	Mem[1]			

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	✓
L.D	F2,45(R3)	✓	✓	✓
MUL.D	F0,F2,F4	✓	✓	✓
SUB.D	F8,F2,F6	✓	✓	✓
DIV.D	F10,F0,F6	✓	✓	
ADD.D	F6,F8,F2	✓		

Register	Qi
F0	
F2	
F4	
F6	Add2
F8	
F10	Mult2

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	yes	ADD	F8	Mem[2]			
Add3	no						
Mult1	no						
Mult2	no						

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	✓
L.D	F2,45(R3)	✓	✓	✓
MUL.D	F0,F2,F4	✓	✓	✓
SUB.D	F8,F2,F6	✓	✓	✓
DIV.D	F10,F0,F6	✓	✓	✓
ADD.D	F6,F8,F2	✓	✓	

Register	Qi
F0	
F2	
F4	
F6	Add2
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	no						
Mult2	no						

Execution

Instruction		I	E	W
L.D	F6,34(R2)	✓	✓	✓
L.D	F2,45(R3)	✓	✓	✓
MUL.D	F0,F2,F4	✓	✓	✓
SUB.D	F8,F2,F6	✓	✓	✓
DIV.D	F10,F0,F6	✓	✓	✓
ADD.D	F6,F8,F2	✓	✓	✓

Register	Qi
F0	
F2	
F4	
F6	
F8	
F10	

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	no						
Mult2	no						

Issues for Tomasulo

- Hardware complexity
  - Complex control logic
  - Associative lookup at each reservation station
- Only 1 Common Data Bus
- Imprecise interrupts
  - Instructions issue in order, but do not necessarily complete in order
- Control flow
  - Branches cause stall

Cost of branches

- Pipeline flushes
  - Instructions on wrong path
- Redirected instruction fetch
  - Target address not prefetched
- Frequency
  - Branches are 20%-25% of instructions executed
- Can we reduce branch penalties?

Loops

```
for(i=0;i<100;i++) {

     …

}
```

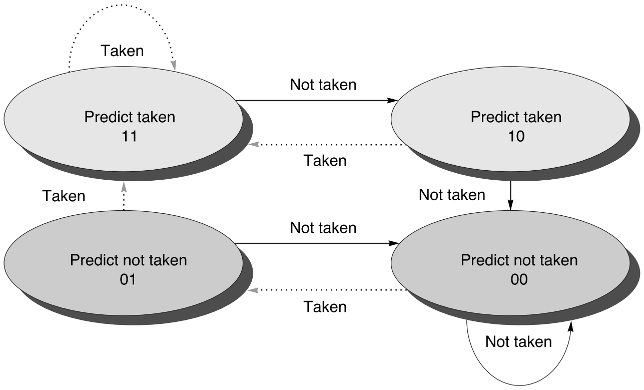
This branch is usually taken

Branch prediction

- Static prediction
  - E.g., predict not taken
- Software hint
- Add state
  - 1 bit that which direction branch went last time it was executed
  - Note: wrong twice per loop!

Improving branch prediction

- 2-bit saturating counter

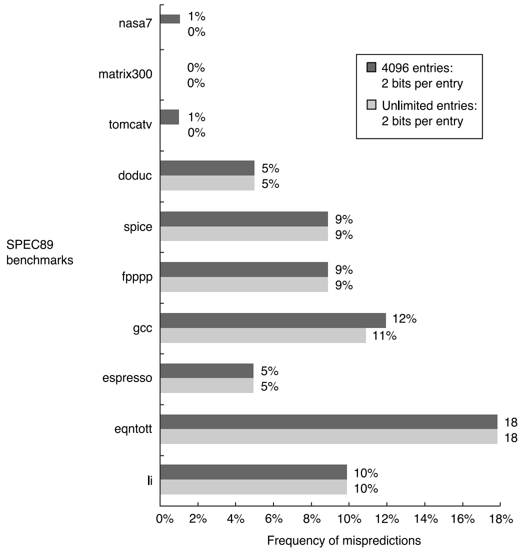


© 2003 Elsevier Science (USA). All rights reserved.

Implementing branch prediction

- Use a “cache” of counters indexed by PC
  - Doesn’t matter that the counter at address might be for a different branch
    - (it’s only an optimization)
- Increase size of counters
  - After 2 bits, it doesn’t much matter
- Make the number of entries larger
  - After a while (4K, 8K?) it doesn’t much matter
- Note: Performance is a function of prediction accuracy ***and*** branch frequency

Prediction accuracy

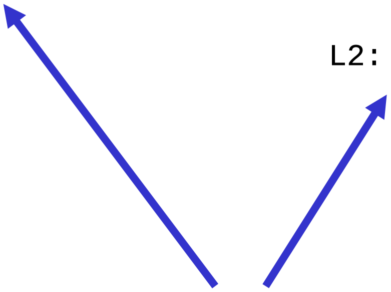


© 2003 Elsevier Science (USA). All rights reserved.

Correlating branch predictors

```
if(aa==2)
      aa=0;
if(bb=2)
      bb=0;
if(aa==bb) {
```

```
    DSUBUI  R3,R1,#2
    BNEZ    R3,L1
    DADD    R1,R0,R0
L1: DSUBUI  R3,R2,#2
    BNEZ    R3,L2
    DADD    R2,R0,R0
L2: DSUBU   R3,R1,R1
    BEQZ    R3,L3
```



This branch is correlated
with the previous branches

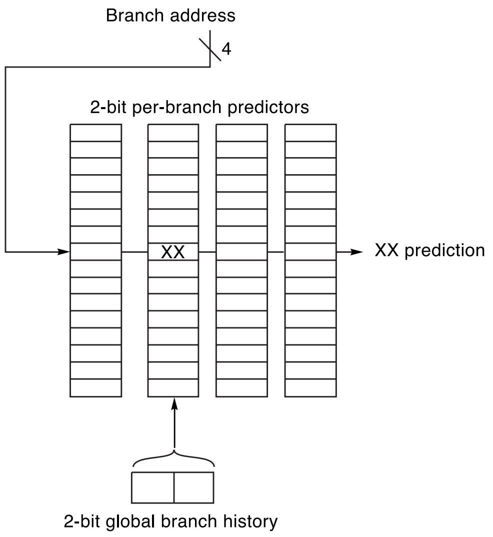
Implementation

- Record the direction of the last *m* branches
  - Generates an *m*-bit bit vector
- Use the *m*-bit vector to index into a table of 2^m *k*-bit predictors
- Called a global predictor
  - Previous example is called a local predictor

Generalization

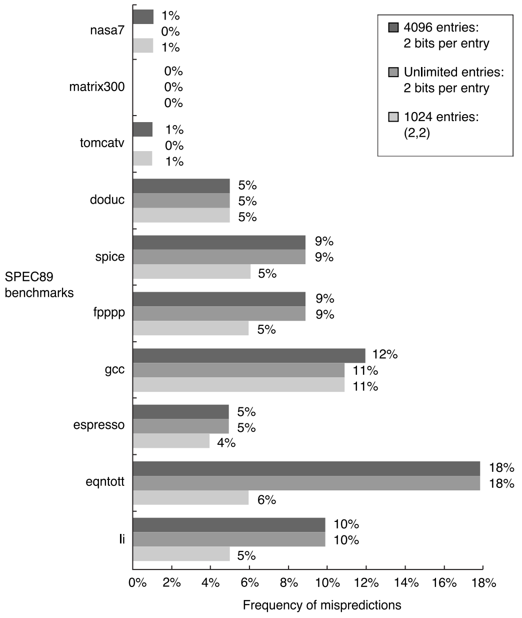
- (*m*,*n*) predictors
  - *m* bit global vector selects among 2^m local predictor tables
  - Each local predictor table entry is an *n* bit counter
- It still doesn’t matter whether the table entry really matches the current branch

2-level Branch Prediction Buffer



© 2003 Elsevier Science (USA). All rights reserved.

Impact of 2-level prediction

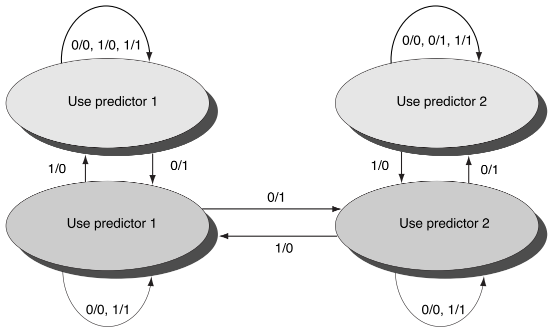


- 8K bits branch prediction table equivalent to infinite table
- 8K bits of branch prediction table organized (2,2) is up to twice as effective

© 2003 Elsevier Science (USA). All rights reserved.

Tournament Predictor

- Use state counter to select which predictor to use
  - E.g., 2-bit state counter to switch between a local and a global branch predictor

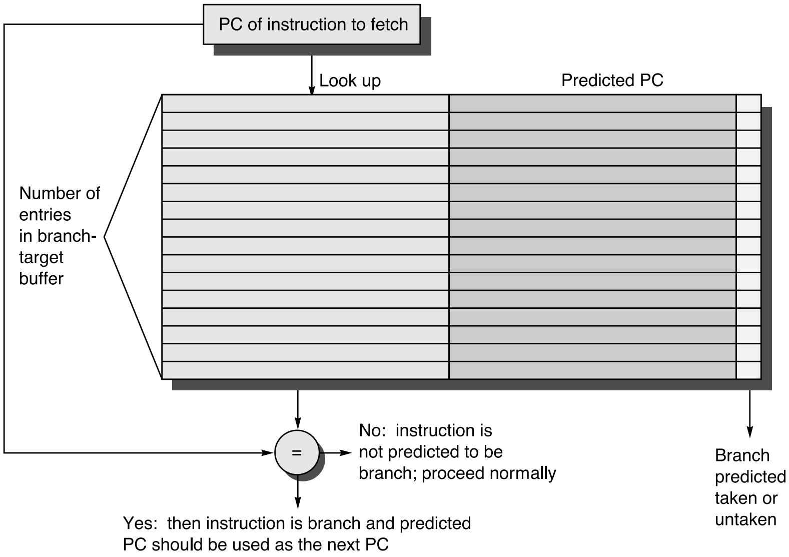


© 2003 Elsevier Science (USA). All rights reserved.

Speeding up instruction fetch after branches

- Branch Target Buffer
  - Cache that stores predicted address of the instruction that follows a branch
  - Functions just like a cache
    - i.e., unlike Branch Prediction Table, the target has to be for the correct branch
- Extension: store the *instruction* at the target address, not just its address

BTB Example


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be branch; proceed normally
Yes: then instruction is branch and predicted PC should be used as the next PC
Branch predicted taken or untaken

© 2003 Elsevier Science (USA). All rights reserved.