Dynamic Branch Prediction (1): Branch Prediction (History) Buffer

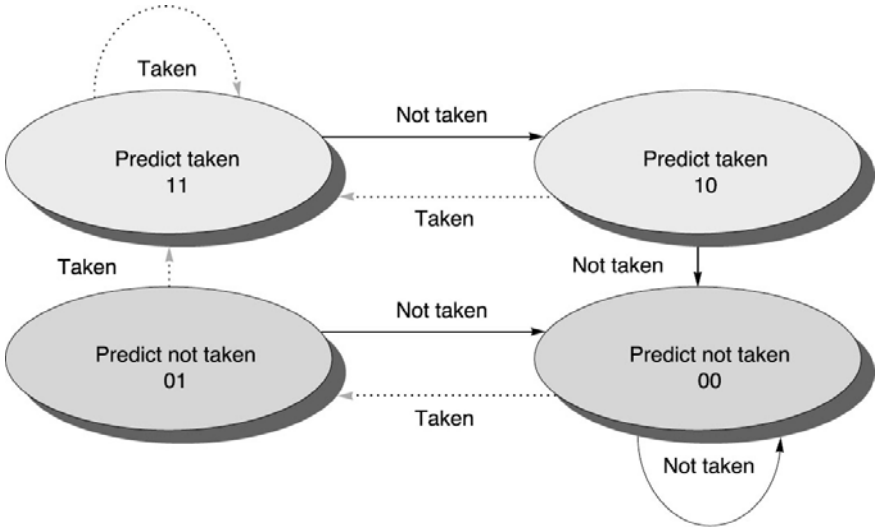
- Small memory indexed by the low-order bits of the branch instruction
  - Stores a single bit of information: T or NT
    - Starts off as T, flips whenever a branch behaves opposite to prediction
  - Benefits for larger pipelines, more complex branches

- Problems with this simple scheme
  - Prediction value may not correspond to branch being considered
    - Cannot avoid this: Branch Prediction Buffer serves as a cache without tags
  - Does not do a good job of predicting "mostly-taken branches"
    - E.g, a loop: `for (i=0; i<10; i++) { … }`
    - Repeated executions of the loop will result in 2 incorrect predictions
      - Last iteration flips T to NT
      - First iteration flips NT to T
    - So, prediction accuracy of 80%

- Can we do better?

Dynamic Branch Prediction (2): 2-bit Prediction Schemes

- Store 2 bits of information in branch history buffer

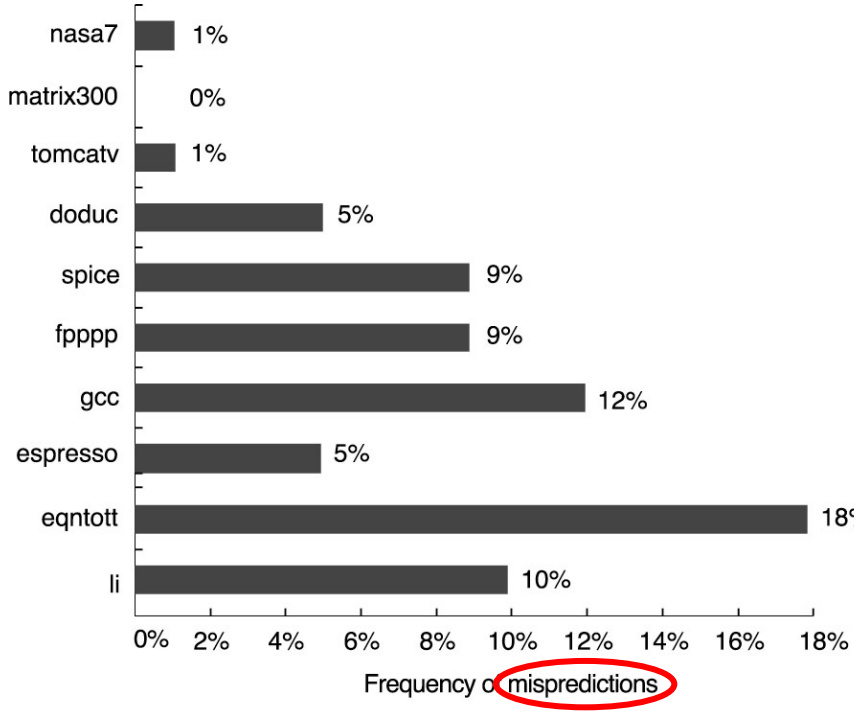


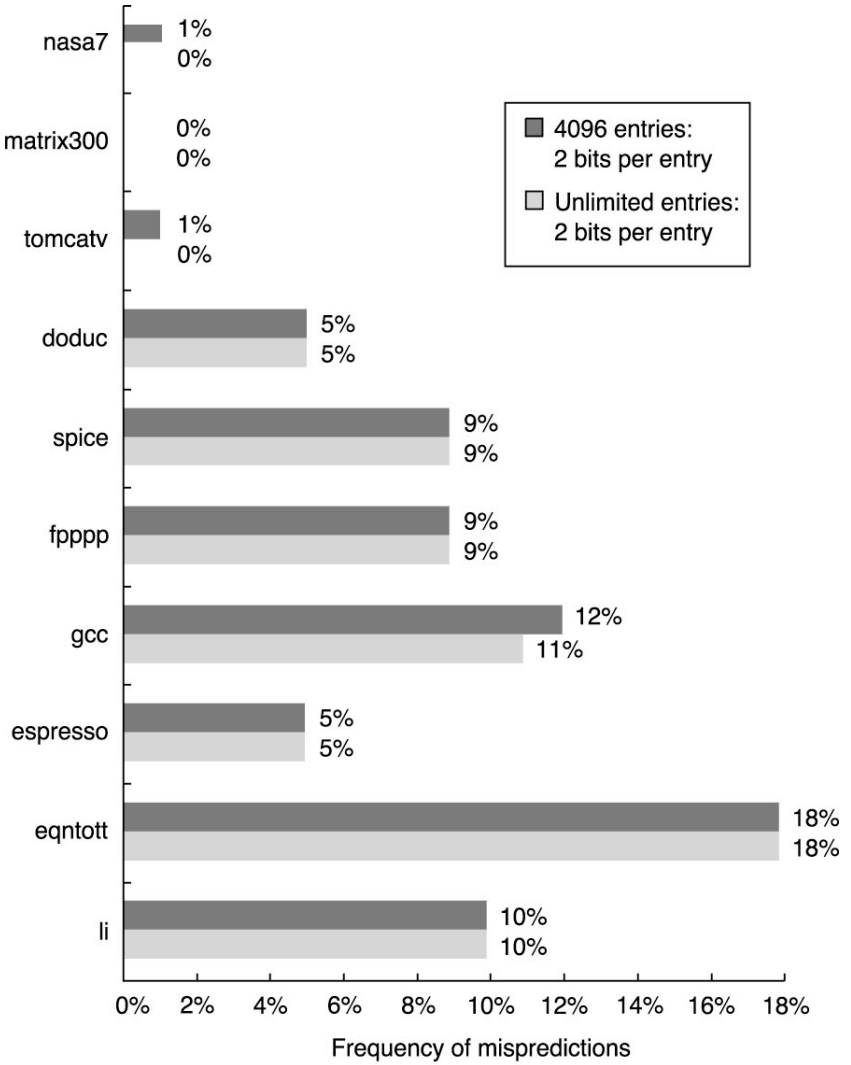
- How does this do on our loop example?
  - 1 misprediction per iteration if we start off in the (11) state
  - 1 misprediction per iteration (plus 2 mispredictions the first time) if we start in (00) state

- Generalization: n-bit saturating counters
  - Increment if taken, decrement if not
  - Predict T if value more than half, else NT
  - Not too much of a win over 2-bit counters

Prediction Accuracy of 2-bit Prediction Schemes

- SPEC89 benchmarks using a 4096-entry 2-bit prediction buffer (Pan, So, and Rameh [1992])





- Is hit-rate in the cache an issue?

Dynamic Branch Prediction (3): Correlating Branch Predictors

- 2-bit predictor uses only the recent behavior of a single branch to predict its future behavior
- Is branch direction affected by more "global" properties?

```
if (aa == 2)
   aa = 0;
if (bb == 2)
   bb = 0;
if (aa != bb)
   { … }
```

```
assuming aa and bb are in R1 and R2
      DSUBUI  R3, R1, #2
      BNEZ    R3, L1        ; branch b1
      DADD    R1, R0, R0
L1:   DSUBUI  R3, R2, #2
      BNEZ    R3, L2        ; branch b2
      DADD    R2, R0, R0
L2:   DSUBU   R3, R1, R2
      BEQZ    R3, L3        ; branch b3
```

- Behavior of b3 is correlated with that of b1 and b2
  - if both b1 and b2 are NT, b3 will be T
- Can (how do) we predict such branches?

A (1,1) Correlating Predictor

```
if (d == 0)
    d = 1;
if (d == 1)
    { … }
```

```
        BNEZ    R1, L1      ; branch b1
        DADDUI  R1, R0, #1
L1:     DADDUI  R3, R1, #-1
        BNEZ    R3, L2      ; branch b2
...
L2:
```

d_{ini}	=0?	b1	=1?	b2
0	yes	NT	yes	NT
1	no	T	yes	NT
2	no	T	no	T

Behavior of a 1-bit predictor for repeated executions of above with values of d=2, 0, 2, 0,…

d_{ini}	$b1_p$	$b1_{act}$	$b1_{new}$	$b2_p$	$b2_{act}$	$b2_{new}$
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

1-bit predictor that uses 1 bit of correlation

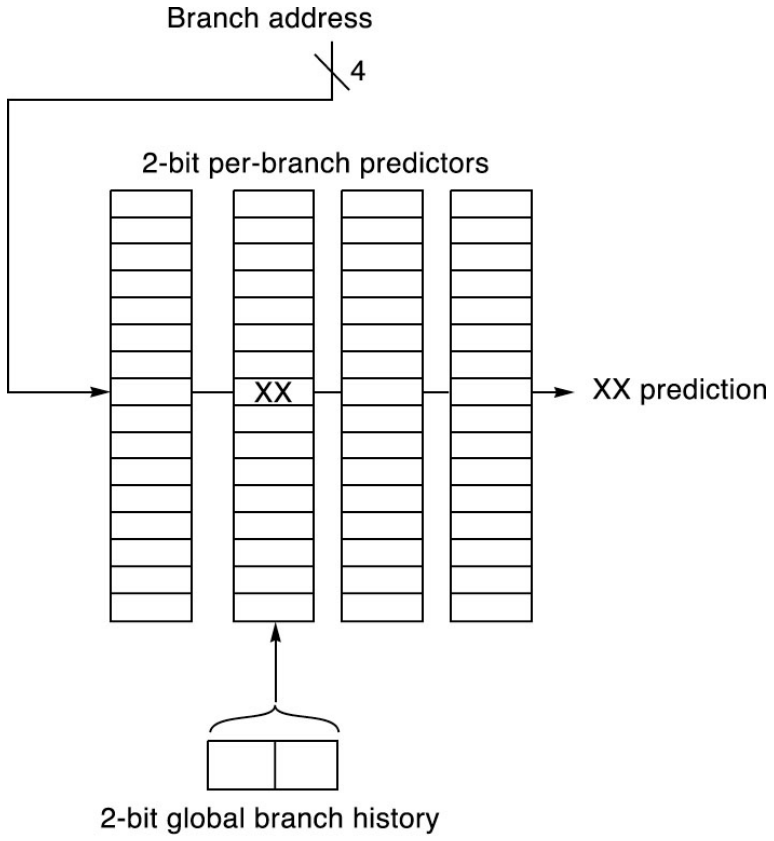
- *X/Y*: X if last branch was NT,
  Y if last branch was T

	$b1_p$	b1	$b1_n$	$b2_p$	b2	$b2_n$
2	**NT**/NT	T	T/NT	NT/**NT**	T	NT/T
0	T/**NT**	NT	T/NT	**NT**/T	NT	NT/T
2	**T**/NT	T	T/NT	NT/**T**	T	NT/T
0	T/**NT**	NT	T/NT	**NT**/T	NT	NT/T

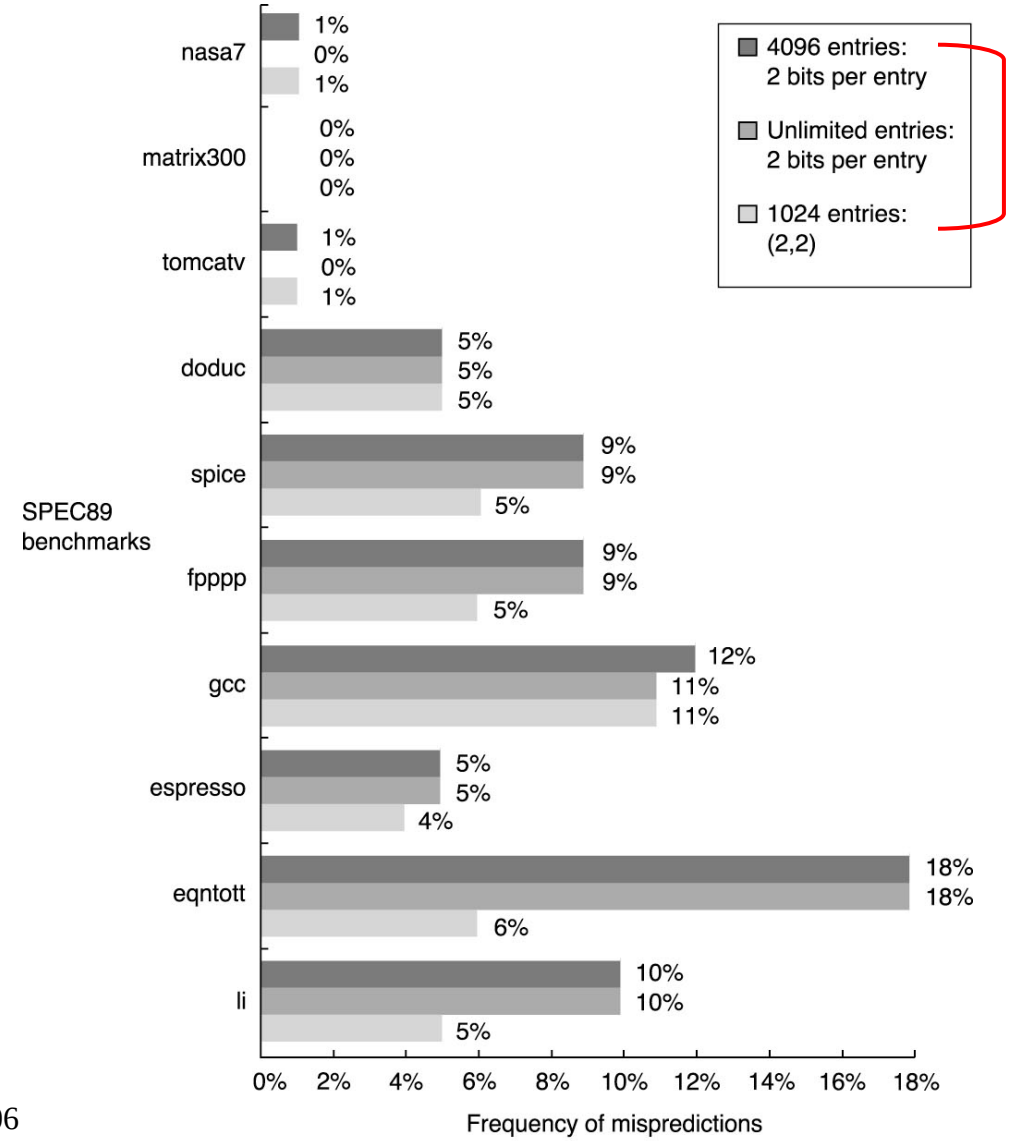
(m,n) Correlating Predictors

- Use behavior of the last m branches to choose from among 2^m n-bit predictors (for a single branch)
  - Makes the predictor context-sensitive
- Yields improved prediction accuracy for small hardware cost
  - History of last m branches can be kept as a shift register
    - Each bit records whether corresponding branch was T/NT
  - Branch prediction buffer can then be indexed by concatenating the lower-order bits of address with the m-bit history
- Variant: gshare
  - History and branch address bits are xor-ed

A (2,2) Correlating Predictor



Prediction Accuracy of Correlating Predictors



Is this a fair comparison?

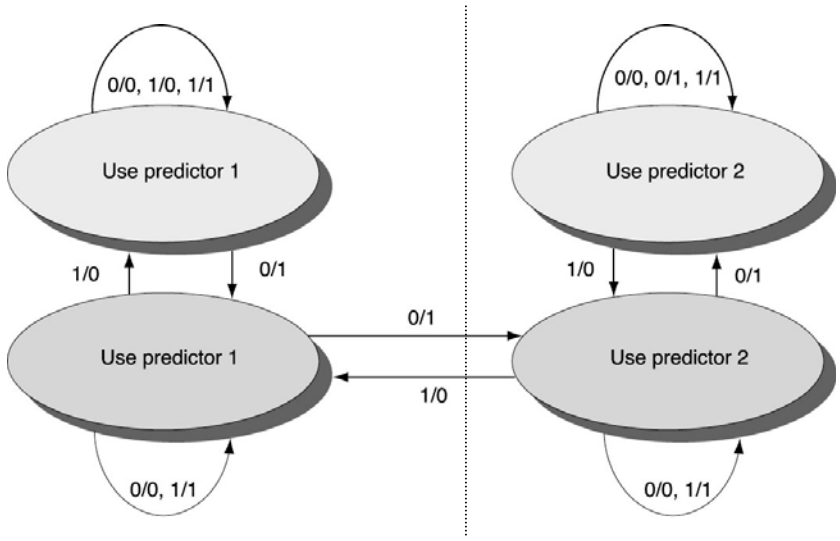
Metric: Number of bits

4096 entries (2-bit): 8K

1024 entries (2,2): $2^2 \times 2 \times 1024$ = 8K

Tournament Predictors

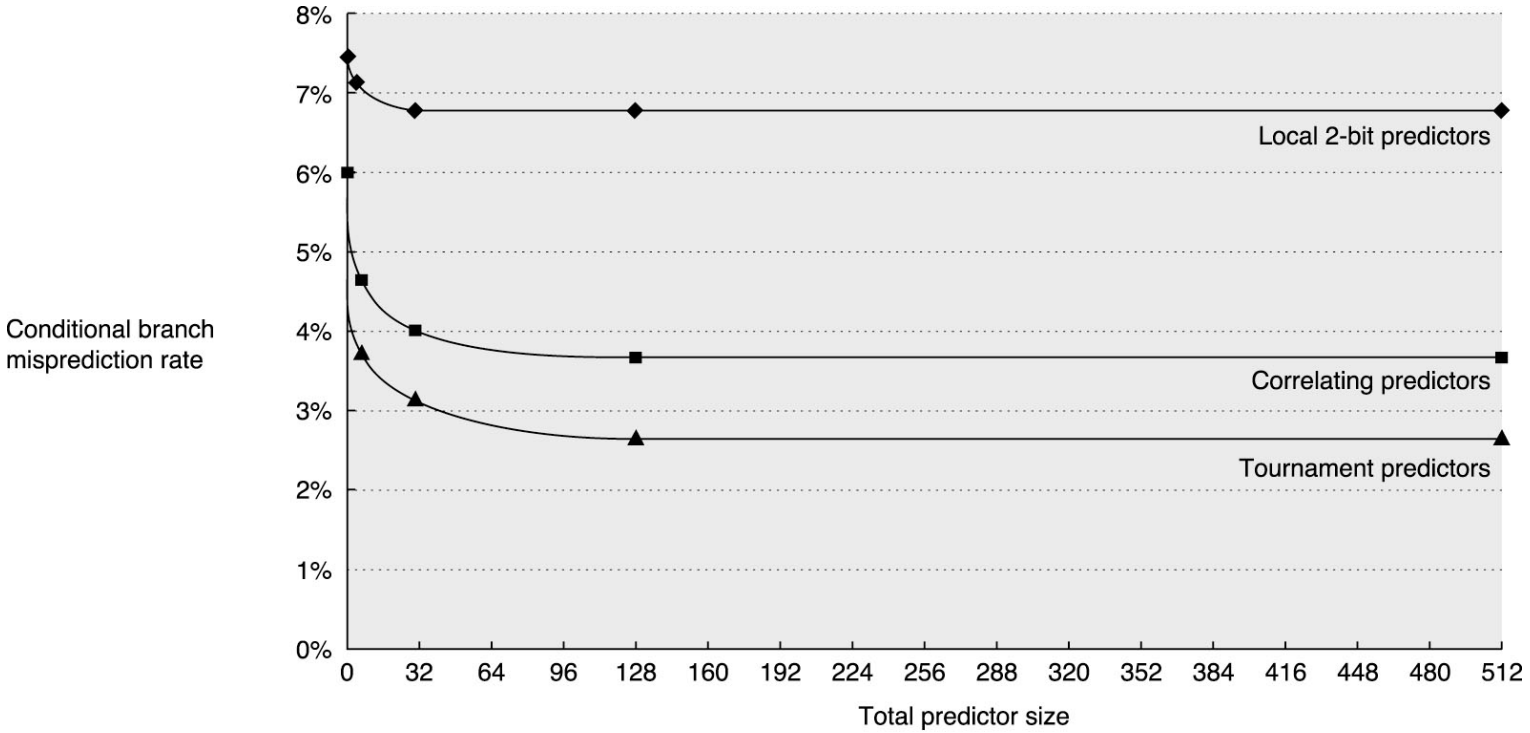
- Adaptively combine local and global predictors
  - Make more effective use of large numbers of prediction bits
- Generalization: Multilevel branch predictors
  - Several levels of branch prediction tables
    - Combination of local (with different bits/branch) and global (different m,n)
  - Selector algorithm that chooses which predictor to apply
- A selector should be adaptive ("learn" from its mistakes)
- Saturating counters as in local predictors
  - Increment/decrement based on which predictor is correct (when the other is incorrect)



0|1: Predictor 1 was wrong, Predictor 2 was correct

Performance of Tournament Predictors

- Misprediction rate on SPEC89 as total number of bits is increased
  - "optimal" correlating predictor chosen at each point



Examples of Branch Predictors in Use

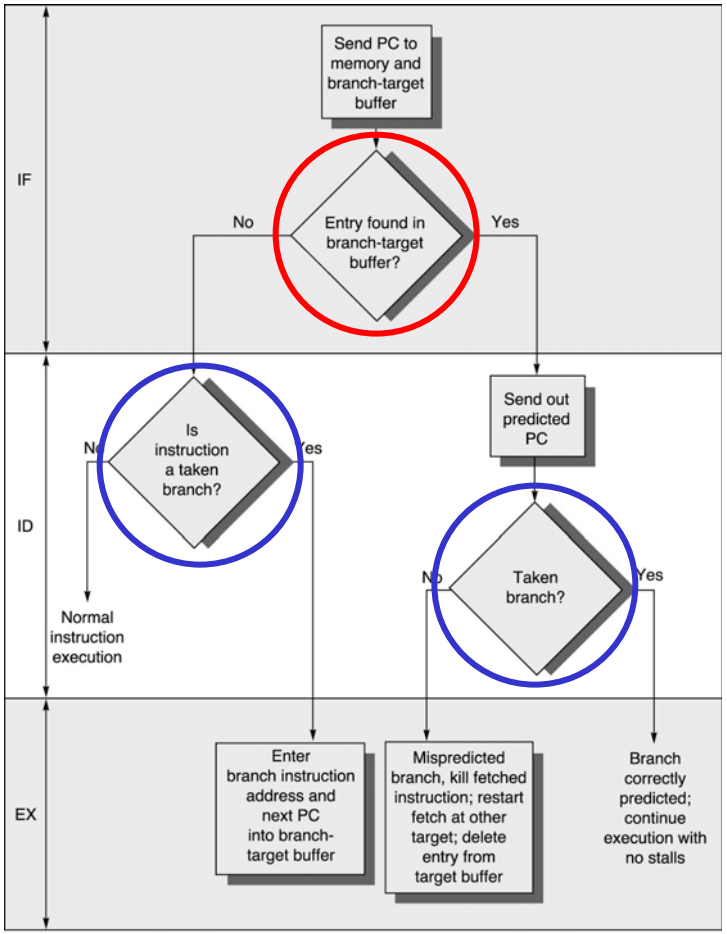
- Pentium
  - 2-bit local predictor
    - direct jump from (00) to (11) state
- Pentium MMX, Pentium Pro, Celeron, Pentium II
  - (4,2) correlating predictor
- Pentium III
  - 2-level adaptive tournament predictor
    - details are sketchy
  - 512-entry Branch Target Buffer (BTB) (next)
- Pentium IV
  - 4096-entry BTB
  - Execution trace cache (later)
- Alpha 21264 used a tournament predictor
- Selector
  - Uses 4K 2-bit counters indexed by the local branch address to select between local and global predictor
- Global predictor
  - 4K entries, indexed by history of last 12 branches
  - Each entry is a 2-bit predictor
  - (12,2) correlating predictor
- Local predictor is itself 2-level
  - Top-level: 1K 10-bit history of (local) branch outcomes
    - Detects patterns
  - History used to index a 1K 3-bit saturating counter table

Branch Target Buffer (BTB)

- In the classic 5-stage pipeline, an instruction is identified as a branch only in the ID stage
  - Branch prediction buffer can help decide whether to fetch from target address (fast pipeline), or fall-through
  - However, IF still ends up fetching a (possibly) useless instruction
    - So, even with perfect branch prediction, cannot achieve 0-cycle branch latency
- Solution: Branch Target Buffer (BTB)
  - Accessed during the IF stage
  - A cache that stores predicted address for the next instruction after a branch
- Variants
  - Store only predicted taken branches
    - Works for 1-bit local predictors: store entry when changing prediction to T
  - Use separate target and prediction buffers

Combining Target and Prediction Buffers

- Assuming IF only has BTB, and ID is responsible for prediction



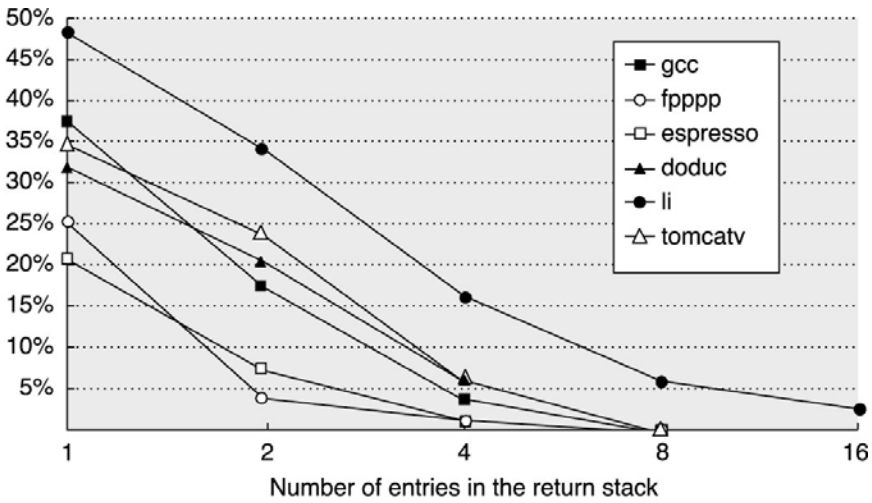
- Branch penalties
  - Assuming new target is written into PC only at the end of EX

BTB hit	T/NT?	Actual	Penalty
Yes	T	T	0
Yes	Mispredict or predict NT		2
No		T	2
No		NT	0

- Current-day processors combine target and prediction logic into a separate Instruction Fetch Unit
  - operates independently of the pipeline
  - Variant: Store (decoded) instructions in BTB (as opposed to just the target)

Return Address Predictors

- Techniques discussed so far work only with direct branches
  - Target is PC-relative (part of the instruction)
- An important category of indirect jumps: returns
  - 85% of indirect jumps in SPEC89 (more in languages like C++, Java)
  - Can we predict the return address?
- *Option* 1: Use BTBs
  - Accuracy tends to be low
  - return address depends on call site
- *Option* 2: Use a Return Address Stack (RAS)
  - Push an entry to the RAS at a call
  - Pop the entry upon the return
  - Perfect prediction if call depth does not exceed RAS buffer size
- Misprediction rates



Role of Branch Prediction

- Branch prediction techniques achieve 80 – 95% accuracy
  - Exact benefit varies based on program type, size of buffer
  - Crucial for current-day microprocessors
    - Need to supply multiple instructions per cycle to subsequent stages
- Can also reduce branch penalties by reducing misprediction penalties
  - Fetch from both predicted/unpredicted paths
  - Store/buffer instructions from both paths in the BTB
    - Extension of this idea: Execution Trace Cache (later)
  - Possible to do the above because of increasing transistor budgets

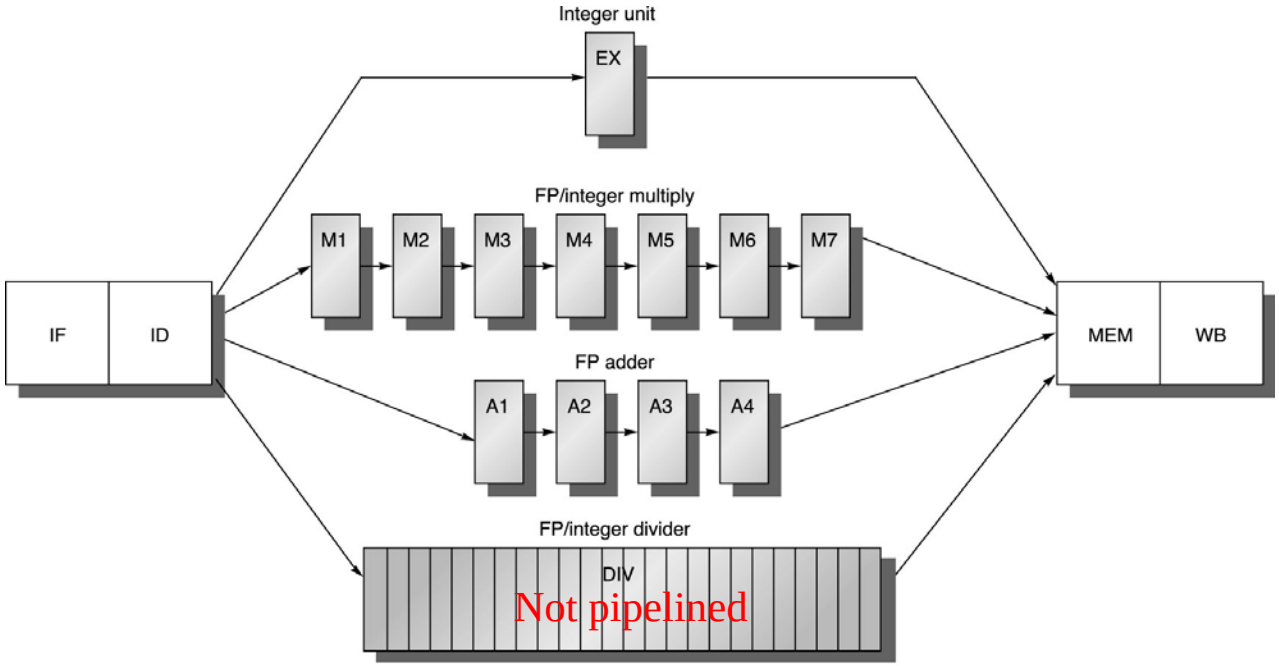
Advanced Methods for Dealing with Data Hazards

Dealing with Data Hazards

- Data hazards result in pipeline stalls
  - Because instructions need to wait for their results to become available
  - Also affects subsequent instructions that do not need these results
- So far:
  - Register and result forwarding
  - Relied on in-order execution of instructions
  - Use static (compiler) scheduling to reduce the number of stalls
- Now: Dynamic Scheduling
  Hardware rearranges the instruction execution to reduce stalls, while maintaining data flow and exception behavior
- To see the impact of these schemes let us look at a more advanced pipelined ISA with multi-cycle operations

Extending the RISC Pipeline to Handle Multicycle Operations

- Classic RISC pipeline assumes that all operations complete in 1 cycle
  - Impractical because of clock-period and logic considerations
- More practical is a model where the (original) EX stage is split across multiple functional units, each of which may or may not be pipelined



Instruction Flow through Longer Latency Pipelines

- Standard pipeline model extended with additional stages

MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	**MEM**	WB			

Red: Data needed **Blue: Results available**

- Increased stalls because of RAW hazards

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4, 0(R2)	IF	ID	EX	**MEM**	WB												
MUL.D F0, F4, F6		IF	ID		M1	M2	M3	M4	M5	M6	**M7**	MEM	WB				
ADD.D F2. F0, F8			IF		ID							A1	A2	A3	**A4**	MEM	
S.D F2, 0(R2)					IF							ID	EX				**MEM**

- Also increases stalls due to other hazards
  - Structural: Initiation intervals > 1, More than one register write per cycle
  - WAW hazards because different instructions take different number of cycles; WAR does not happen

Dynamic Scheduling

- Hardware rearranges the instruction execution to reduce stalls, while maintaining data flow and exception behavior

- Idea: Instruction can execute as soon as data dependencies are satisfied
  - Separation of instruction issue, execution, and commit stages
    - Classic pipeline: these correspond to IF and ID, EX and MEM, and WB
    - ID stage split into two
      - Issue: Decode instruction, check for structural hazards
      - Read Operands: Wait until no data hazards, then read operands
  - Variants
    - In-order issue, out-of-order execution, in-order commit
    - Out-of-order issue, out-of-order execution, in-order commit
    - Out-of-order issue, out-of-order execution, out-of-order commit

Major Dynamic Scheduling Implementations

Two implementations

- Scoreboarding (Appendix A)
- Tomasulo's algorithm (Chapter 3)
  - A variation of this is used in current-day microprocessors

Scoreboarding

- Named after the CDC 6600 scoreboard
  - Used to orchestrate parallel instruction execution among functional units
    - 4 floating-point, 5 memory-reference, 7 integer
- We will illustrate the technique using a simpler RISC machine
  - 2 FP multiply (10 EX cycles), 1 FP add (2 EX), 1 FP divide (40 EX)
  - 1 integer unit (1 EX)
- Scoreboard: Keeps track of instruction and machine status, permitting the instruction to execute as soon as its operands are available
  - Instructions can execute out-of-order
  - Have to deal with all three kinds of data hazards
    - RAW hazard: can happen as in the classic pipeline
    - WAW hazard: can happen as in pipelines with multi-cycle operations
    - **WAR hazard: can happen because of out-of-order execution**
  - Our scoreboard does not take advantage of forwarding, since it waits until results are written back to the register file; We emphasis on FP operations

Four Stages of Scoreboard Control

- Issue (ID1)
  - decode instructions
  - check for structural and WAW hazards; example:

```
DIV.D   F0,   F2,  F4
ADD.D   F10, F0, F8
SUB.D   F10, F8, F14
```

  - stall until structural and WAW hazards are resolved; no further issues
- Read operands (ID2)
  - wait until no RAW hazards (i.e., no earlier issued active instruction is going to write to source operands of this instruction)
  - then read operands
- Execution (EX)
  - operate on operands
  - may be multiple cycles - notify scoreboard when done
- Write result (WB)
  - finish execution
  - stall if WAR hazard; example:

```
DIV.D   F0,    F2, F4
ADD.D   F10, F0, F8
SUB.D   F8,   F8, F14
```

Three Parts of the Scoreboard

- Instruction status
  - Indicates which of 4 steps the instruction is in: ID1, ID2, EX, or WB.

- Functional unit status: Indicates the state of each functional unit (FU)
  - Busy — Indicates whether the unit is busy or not
  - Op — Operation to perform in the unit (e.g., + or –)
  - F_i — Destination register
  - F_j, F_k — Source-register numbers
  - Q_j, Q_k — Functional units producing source registers Fj, Fk
  - R_j, R_k — Flags indicating when F_j, F_k are ready

- Register result status
  - Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

Scoreboard Example

Instruction

Instruction	Issue	Read Ops.	Execute	Write

No Forwarding

Loads (L.D) performed by “Integer”

ADDs and SUBs performed by “Add”

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16

Scoreboard Example: Cycle 1

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1			
L.D F2, 45(R3)				
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Integer				

Scoreboard Example: Cycle 2

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2		
L.D F2, 45(R3)	?			
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Integer				

Scoreboard Example: Cycle 3

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	
L.D F2, 45(R3)	?			
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Integer				

Scoreboard Example: Cycle 4

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	?			
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Integer				

Scoreboard Example: Cycle 5

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5			
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
	Integer						

Scoreboard Example: Cycle 6

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6		
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int.		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	Integer						

Scoreboard Example: Cycle 7

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	
MUL.D F0, F2, F4	6	?		
SUB.D F8, F6, F2	7			
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int.		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int.	Yes	No
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	Integer			Add			

Scoreboard Example: Cycle 8a

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2	7	?		
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

RAW Hazard

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int.		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int.	Yes	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	Integer			Add	Divide		

Scoreboard Example: Cycle 8b (end)

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2	7			
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

RAW Hazard

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1				Add	Divide		

Scoreboard Example: Cycle 9

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9		
DIV.D F10, F0, F6	8	**?**		
ADD.D F6, F8, F2	**?**			

RAW Hazard

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1				Add	Divide		

Scoreboard Example: Cycle 11

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

RAW Hazard

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1				Add	Divide		

Scoreboard Example: Cycle 12

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

RAW Hazard

Structural Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1				Add	Divide		

Scoreboard Example: Cycle 13

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13			

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1			Add		Divide		

Scoreboard Example: Cycle 14

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14		

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1			Add		Divide		

Scoreboard Example: Cycle 16

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

RAW Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1			Add		Divide		

Scoreboard Example: Cycle 17

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**		
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	**?**

RAW Hazard

WAR Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1			Add		Divide		

Scoreboard Example: Cycle 19

Instruction

Instruction	Issue	Read Ops.	Execute	Write	
L.D F6, 34(R2)	1	2	3	4	
L.D F2, 45(R3)	5	6	7	8	
MUL.D F0, F2, F4	6	**9**	19		
SUB.D F8, F6, F2	**7**	9	11	12	
DIV.D F10, F0, F6	8				RAW Hazard
ADD.D F6, F8, F2	13	14	16		WAR Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
Mult1			Add		Divide		

Scoreboard Example: Cycle 20

Instruction

Instruction	Issue	Read Ops.	Execute	Write	
L.D F6, 34(R2)	1	2	3	4	
L.D F2, 45(R3)	5	6	7	8	
MUL.D F0, F2, F4	6	**9**	19	20	
SUB.D F8, F6, F2	**7**	9	11	12	
DIV.D F10, F0, F6	8				RAW Hazard
ADD.D F6, F8, F2	13	14	16		WAR Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Add		Divide		

Scoreboard Example: Cycle 21

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**	19	20
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8	21		
ADD.D F6, F8, F2	13	14	16	

WAR Hazard

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
			Add		Divide		

Scoreboard Example: Cycle 22

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**	19	20
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8	21		
ADD.D F6, F8, F2	13	14	16	22

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
40	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
					Divide		

Scoreboard Example: Cycle 61

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**	19	20
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8	21	61	
ADD.D F6, F8, F2	13	14	16	22

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register

F0	F2	F4	F6	F8	F10	F12	F16
					Divide		

Scoreboard Example: Cycle 62

Instruction

Instruction	Issue	Read Ops.	Execute	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**	19	20
SUB.D F8, F6, F2	**7**	9	11	12
DIV.D F10, F0, F6	8	21	61	62
ADD.D F6, F8, F2	13	14	16	22

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register

F0	F2	F4	F6	F8	F10	F12	F16

Scoreboarding

- Speedup from scoreboard
  - 1.7 for FORTRAN programs
  - 2.5 for hand-coded assembly language programs
  - Effects of modern compilers?
- Hardware
  - Scoreboard hardware approximately same as one FPU
  - Main cost was buses
  - Could be more severe for modern processors
- Limitations
  - No forwarding logic
  - Limited to instructions in basic block
  - Stalls for WAW hazards
  - Wait for WAR hazards before WB
- Are there better solutions?