EITF20: Computer Architecture

Part3.1.1: Pipeline - 2

Liang Liu
liang.liu@eit.lth.se

Outline

- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

Previous lecture

Introduction to pipeline basics

- General principles of pipelining
- Techniques to avoid pipeline stalls due to hazards
- What makes pipelining hard to implement?
- **Support of multi-cycle instructions in a pipeline**

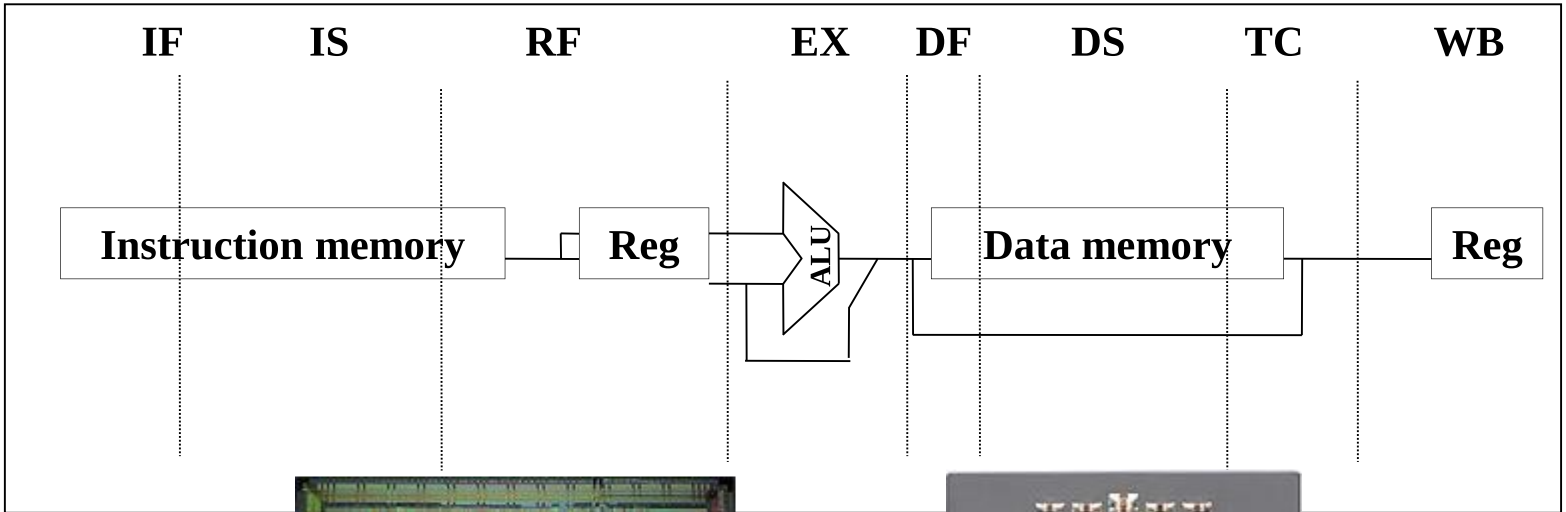
Outline

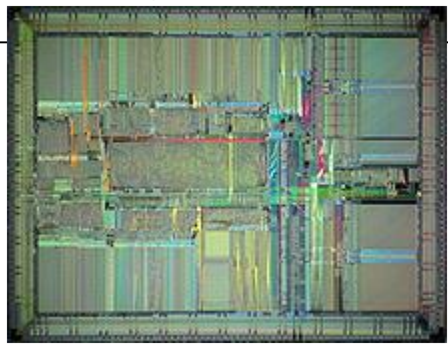
- Reiteration
- **Case Study: MIPS R4000**
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

The MIPS R4000 (deeper pipeline)

R4000 - MIPS64:

- First (1992) true 64-bit architecture (addresses and data)
- Clock frequency (1997): 100 MHz-250 MHz
- Medium deep 8 stage pipeline (super-pipelined)



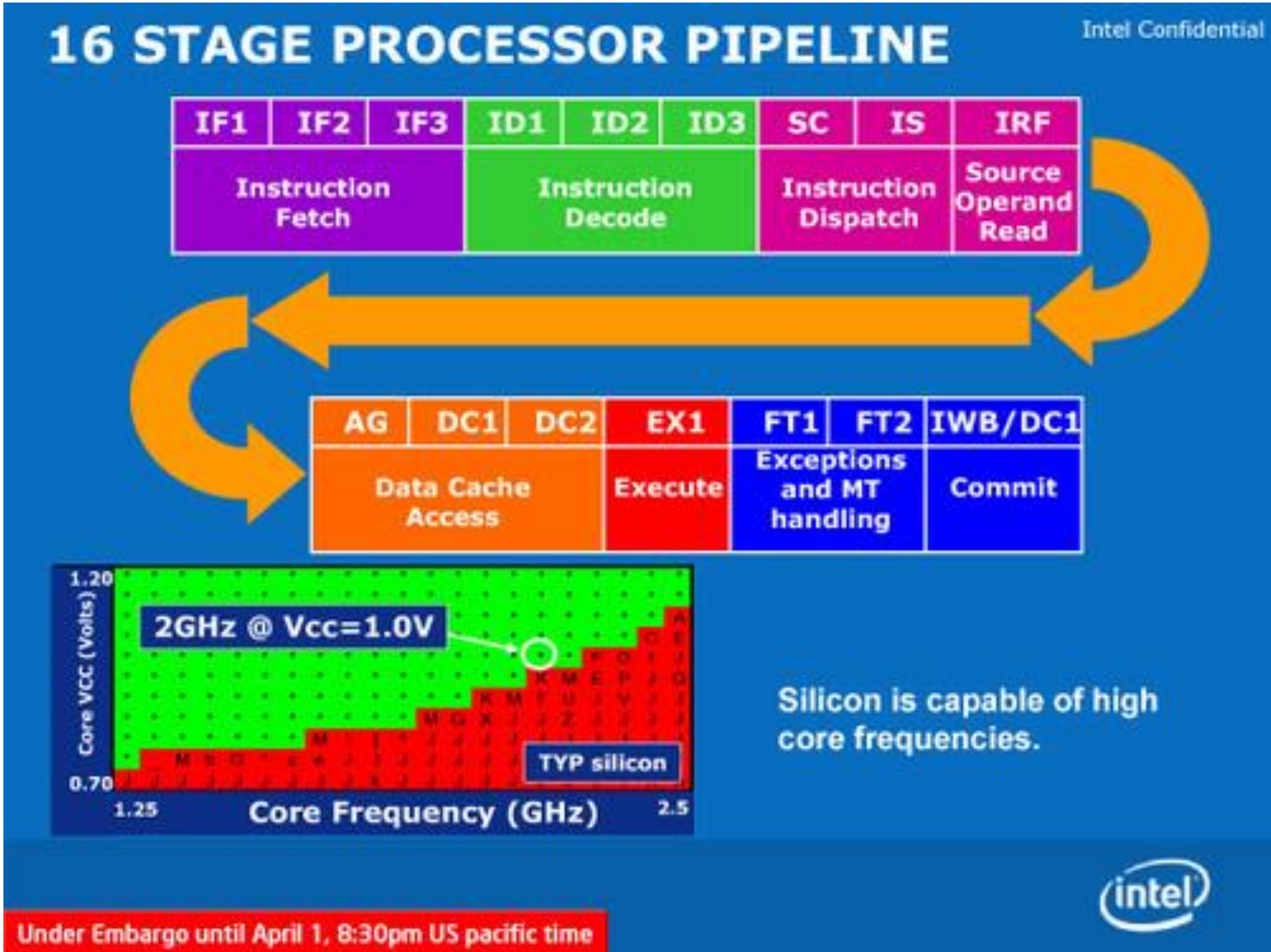


The MIPS R4000

8 Stage Pipeline:

- IF – first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access
- IS – second half of access to instruction cache
- RF – instruction decode and register fetch, hazard checking and also instruction cache hit detection
- EX – execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation
- DF – data fetch, first half of access to data cache
- DS – second half of access to data cache
- TC – tag check, determine whether the data cache access hit
- WB – write back for loads and register-register operations

Modern CPU architecture – Intel Atom



Deeper pipeline

Implications of deeper pipeline

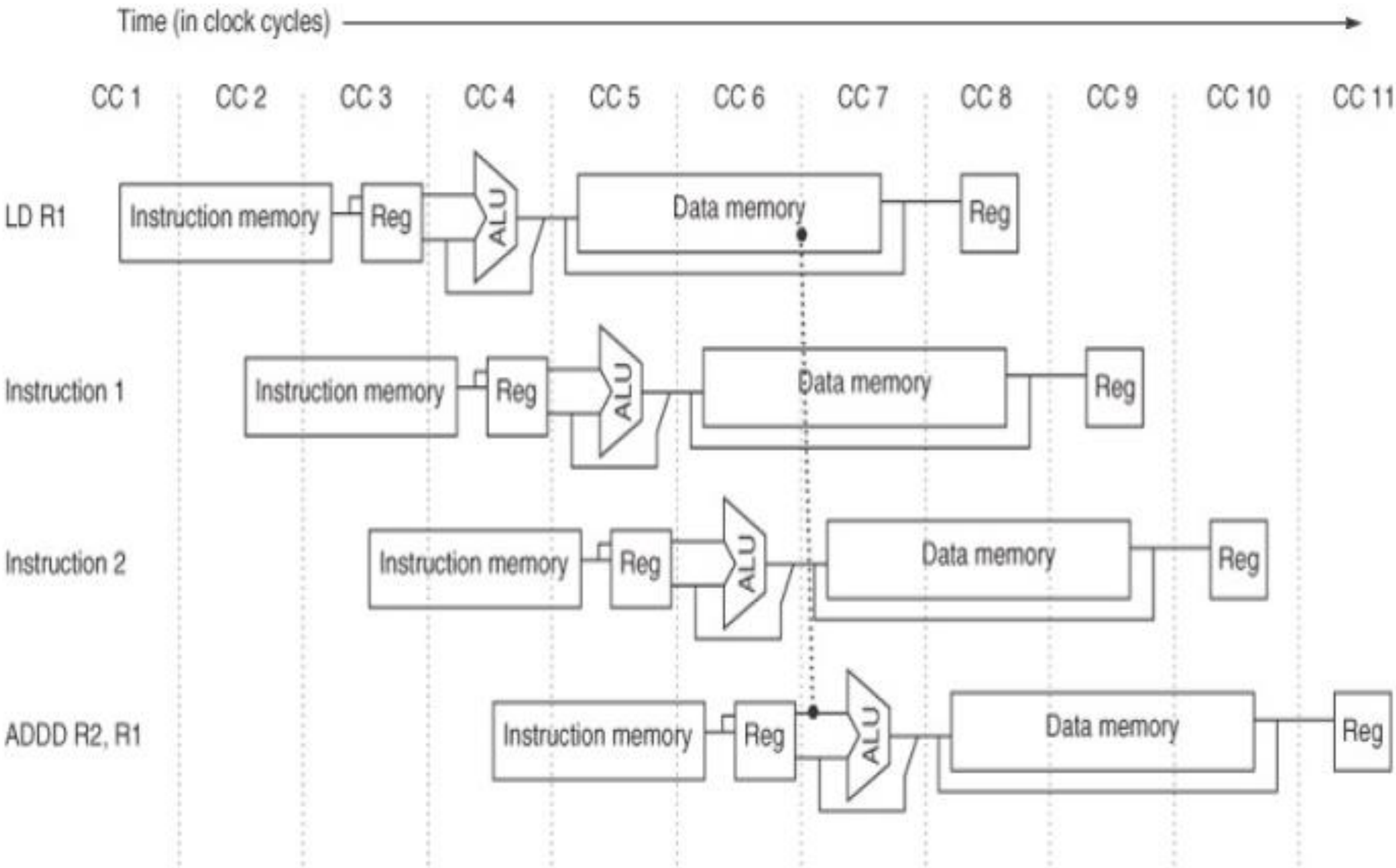
- Load latency: 2 cycles
- Branch latency: 3 cycles (incl. one delay slot) ⇒ High demands on the compiler
- Bypassing (forwarding) from more stages
- More instructions “in flight” in pipeline
- Faster clock, larger latencies, more stalls

Win or loose?

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

Performance equation: CPI ⁎ Tc must be lower for the longer pipeline to make it worthwhile

Load penalties

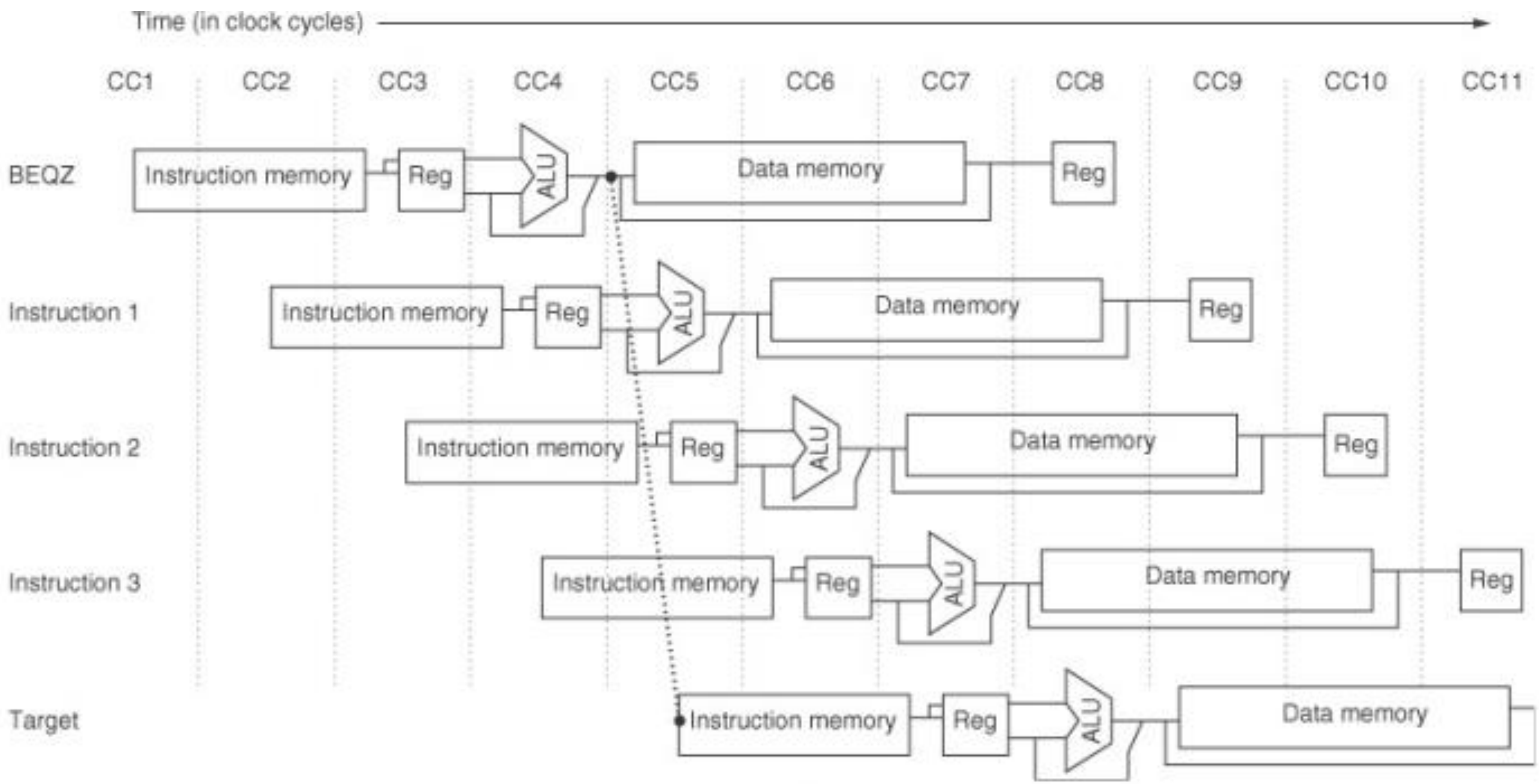


2 stalls even with forwarding

Branch penalties

Handle branch hazard

- 3 branch delay slot (comparing to simple MIPS)
- Predict-Not taken scheme squashes the next two sequentially fetched instructions if the branch is taken (given on delay slot)



R4000 performance

	CPI penalties			
Average CPI	Load	Branch	FP data hazard	FP struct hazard
2.00	0.10	0.36	0.46	0.09

(SPEC92 CPI measurements)

R4000 performance

- The penalty for control hazards is very high for integer programs
- The penalty for FP data hazards is also high
- The higher clock frequency compensates for the higher CPI

Outline

- Reiteration
- Case Study: MIPS R4000
- **Instruction Level Parallelism**
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

Instruction level parallelism (ILP)

- **ILP: Overlap execution of unrelated instructions: Pipelining**
- **Pipeline CPI = Ideal CPI + Structural stalls + Data hazard stalls + Control stalls**
- **MIPS program has 15-25% average branch frequency:**
  - 5 sequential instructions / branch
  - These 5 instructions may depend on each other
  - Must look beyond a single basic block to get more ILP
- **Loop level parallelism**

Loop-level parallelism

```
for (i=1; i≤1000; i=i+1)
        x[i] = x[i] + 10;
```

- There is very little available parallelism within an iteration
- However, there is parallelism between loop iterations; each iteration is independent of the other

MIPS code for the loop

```
loop:  LD       F0, 0(R1)       ; F0 = array element
       ADDD     F4, F0, F2      ; Add scalar constant
       SD       F4, 0(R1)       ; Save result
       DADDUI   R1, R1, #-8     ; decrement array ptr.
       BNE      R1,R2, loop     ; reiterate if R1 != R2
```

Instruction producing result	Instruction using result	Latency
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Integer op	Integer op	0
Integer op	Cond. branch	1

Only consider data hazard

Loop showing stalls

1	loop:	LD	**F0**, 0(R1)	; F0 = array element
2		**stall**		
3		ADDD	<u>F4</u>, **F0**, F2	; Add scalar constant
4		<u>stall</u>		
5		<u>stall</u>		
6		SD	<u>F4</u>, 0(R1)	; Save result
7		DADDUI	**R1**, R1, #-8	; decrement array ptr.
8		**stall**		
9		BNE	**R1**,R2 loop	; reiterate if R1 != R2

How can we get rid of these stalls?

Reconstructed loop

```
Loop:  L.D      F0,0(R1)
       DADDUI   R1,R1,#-8
       ADD.D    F4,F0,F2
       stall
       stall
       S.D      F4,8(R1)
       BNE      R1,R2,Loop
```

- Swap DADDUI and SD by changing offset in SD
- 7 clock cycles per iteration
- Sophisticated compiler analysis required

Can we do better?

Unroll loop four times

```
loop:   LD      F0, 0(R1)
        ADDD    F4, F0, F2
        SD      F4, 0(R1)        ; drop DADDUI & BNE
        LD      F6, -8(R1)
        ADDD    F8, F6, F2
        SD      F8, -8(R1)       ; drop DADDUI & BNE
        LD      F10, -16(R1)
        ADDD    F12, F10, F2
        SD      F12, -16(R1)     ; drop DADDUI & BNE
        LD      F14, -24(R1)
        ADDD    F16, F14, F2
        SD      F16, -24(R1)
        DADDUI  R1, R1, #-32     ; alter to 4*8
        BNE     R1, R2, loop
```

- **14 + 4*(1+2) + 1 = 27 clock cycles, or 6.75 per iteration**

Scheduled unrolled loop

```
Loop:   L.D      F0,0(R1)
        L.D      F6,-8(R1)
        L.D      F10,-16(R1)
        L.D      F14,-24(R1)
        ADD.D    F4,F0,F2
        ADD.D    F8,F6,F2
        ADD.D    F12,F10,F2
        ADD.D    F16,F14,F2
        S.D      F4,0(R1)
        S.D      F8,-8(R1)
        DADDUI   R1,R1,#-32
        S.D      F12,16(R1)
        S.D      F16,8(R1)
        BNE      R1,R2,Loop
```

- **14 clock cycles, or 3.5 per iteration**

Can we unroll more?

- **Code size**
- **Avaliable registers**

```
loop:   LD       F0, 0(R1)
        ADDD     F4, F0, F2
        SD       F4, 0(R1)
        DADDUI   R1, R1, #-8
        BNE      R1,R2, loop
```

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- **Branch Prediction**
- Dependencies
- Instruction Scheduling
- Scoreboard

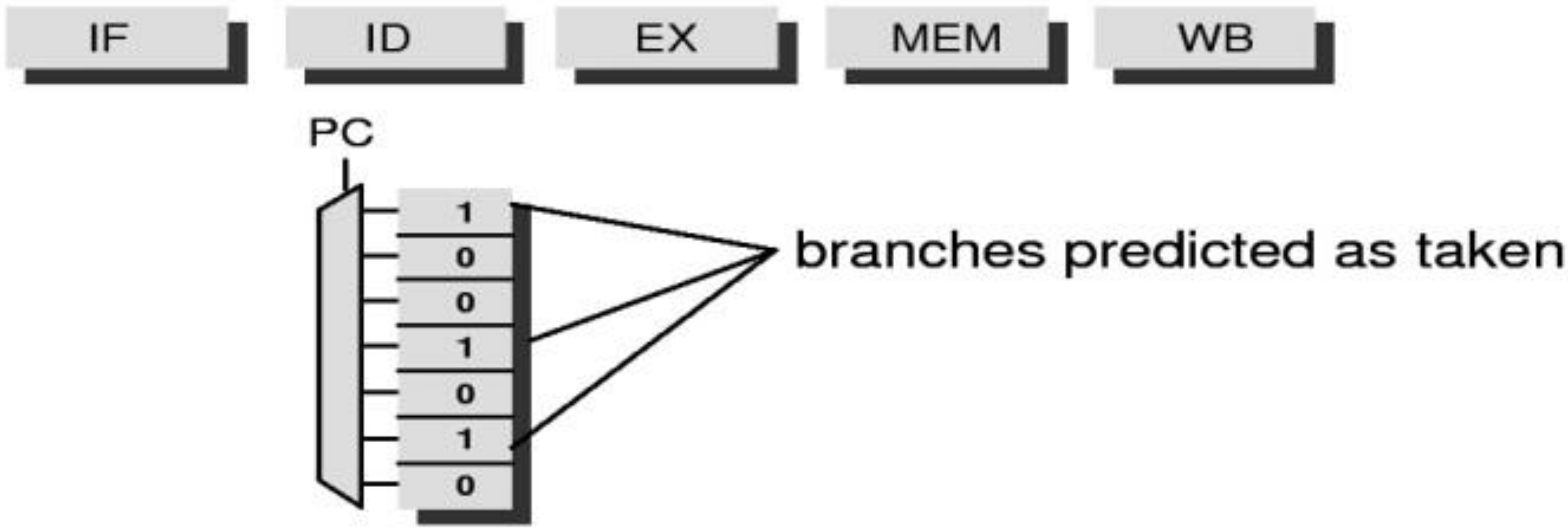
Dynamic branch prediction

- **Branches limit performance because:**
  - Branch penalties
  - Limit to available Instruction Level Parallelism
- **Delayed braches becomes insufficient for deeper pipeline**
- **Dynamic branch prediction to predict the outcome of conditional branches**
- **Principles:**
  - Reduce the time to **when the branch condition is known** (if we can predict correctly)
  - Reduce the time to **calculate the branch target address** (if we can buffer the target address)

Branch history table

Simple branch prediction

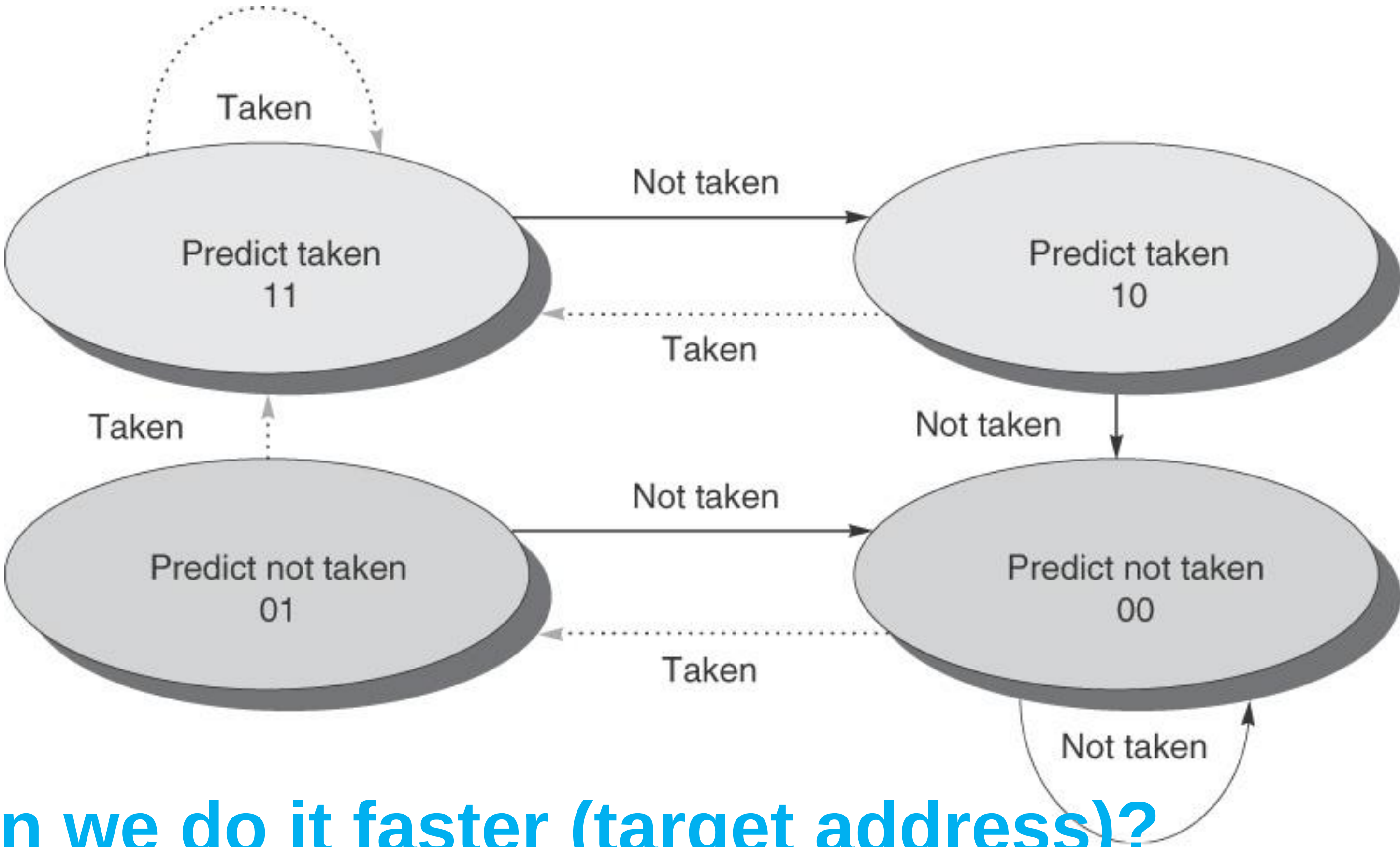
- The branch-prediction buffer is indexed by low order part of branch-instruction address (at ID stage)
- The bit corresponding to a branch indicates whether the branch is predicted as taken or not
- When prediction is wrong: invert bit



Is one bit good enough?

A 2-bit branch predictor

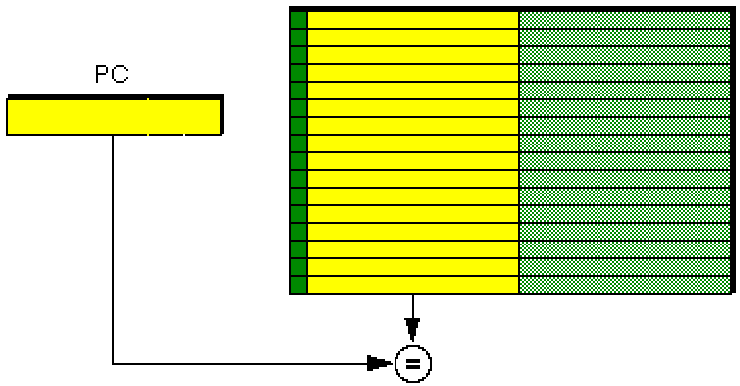
- **2-bit branch prediction with saturating counter**
  - Requires prediction to miss twice in order to change prediction ⇒ better performance
  - 1%-18% miss prediction frequency for SPEC89



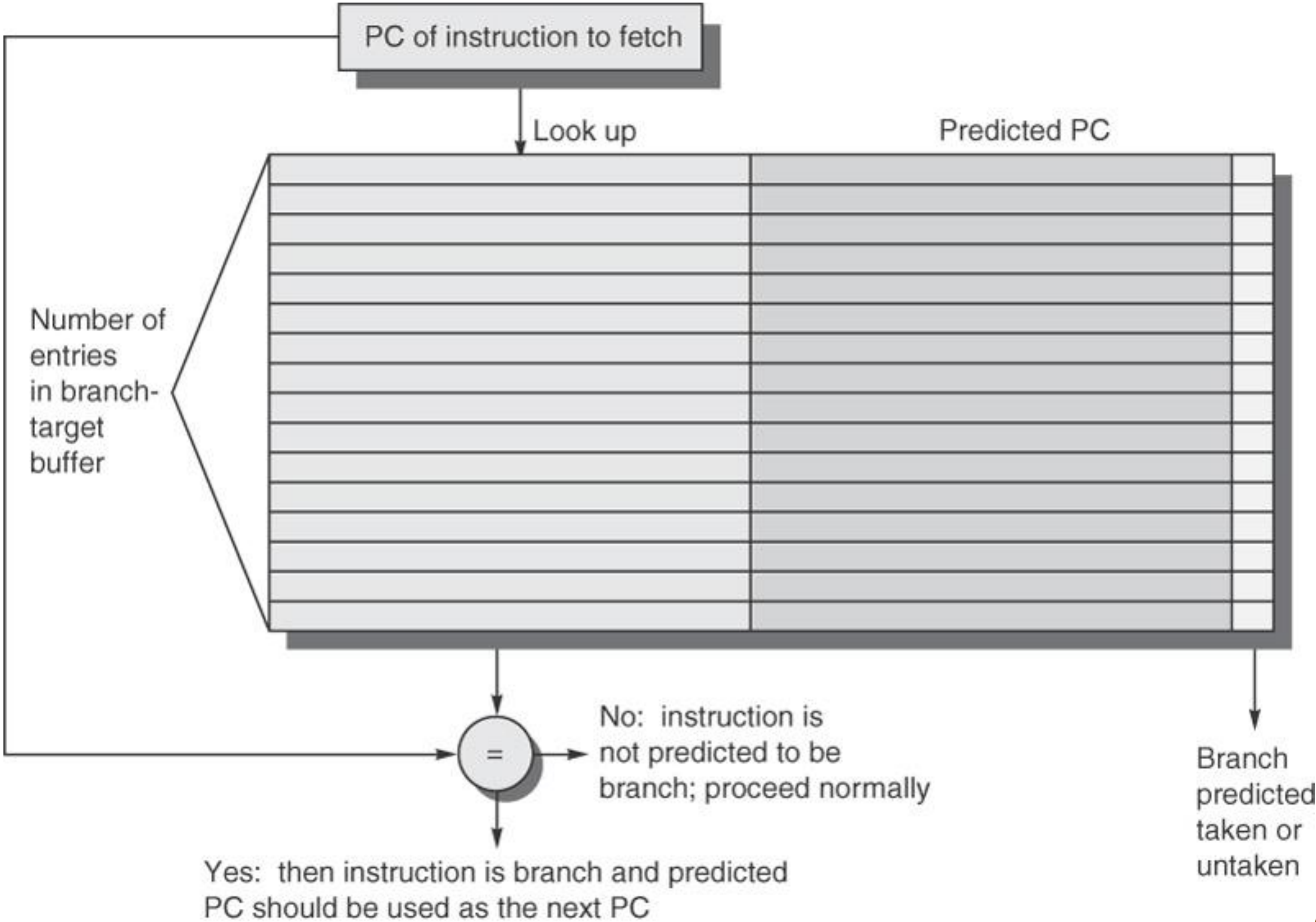
Can we do it faster (target address)?

Branch target buffer

- **Observation: Target address remains the same for a conditional direct branch across dynamic instances**
  - Idea: Store the target address from previous instance and access it with the PC (at **IF** stage, instead of ID)
  - Called Branch Target Buffer (BTB) or Branch Target Address Cache
- **Three steps to be predicted at fetch stage:**
  - Whether the fetched instruction is a branch (somewhat ID)
  - (Conditional) branch direction (only store the predicted-taken braches)
  - Branch target address (if taken)



Branch target buffer



Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- **Dependencies**
- Instruction Scheduling
- Scoreboard

Dependencies

- **Two instructions must be independent in order to execute in parallel**
- **There are three general types of dependencies that limit parallelism:**
  - Data dependencies (RAW)
  - Name dependencies (WAR,WAW)
  - Control dependencies
- **Dependencies are properties of the program**
- **Whether a dependency leads to a hazard or not is a property of the pipeline implementation**

Data dependencies

- **An instruction *j* is data dependent on instruction *i* if:**
  - Instruction *i* produces a result used by instr. *j*, or
  - Instruction *j* is data dependent on instruction k and instr. k is data dependent on instr. *i*

	LD	**F0,0(R1)**
Example:	**ADDD**	**F4,F0,F2**
	SD	**0(R1),F4**

  - Easy to detect for registers

Name dependencies

- **Two instructions use same name (register or memory address) but do not exchange data**
  - Anti-dependence (WAR if hazard in HW)

```
ADDD    F2,F0,F2    ; Must execute before LD
LD      F0,0(R1)
```

  - Output dependence (WAW if hazard in HW)

```
ADDD    F0,F2,F2    ; Must execute before LD
LD      F0,0(R1)
```

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- **Instruction Scheduling**
- Scoreboard

Instruction scheduling

Instruction scheduling

- Scheduling is the process that determines when to **start** an instruction, when to **read** its operands, and when to **write** its result,
- Target of scheduling: rearrange instructions to **reduce stalls** when data or control dependencies are present

Static (compiler at compile-time) scheduling:

- Pro: May look into future; no HW support needed
- Con: Cannot detect all dependencies, e.g.; hardware dependent

Dynamic (hardware at run-time) scheduling:

- Pro: works when cannot know dependence at compile time; makes the compiler simpler;
- Con: Cannot look into the future (practically); HW support needed which complicates the pipeline hardware

CISC vs RISC

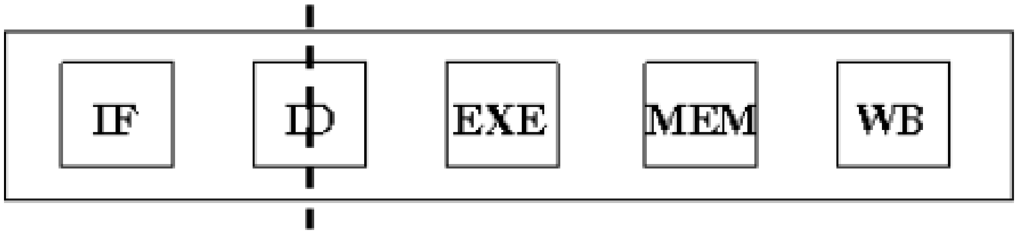
- Simple instruction sets is easier to be scheduled (more flexibility)…

	Load R1,A
	Load R2,B
Add A, B	Add R3, R1, R2
Add C, D	Load R4, C
	Load R5, D
	Add R6, R4, R5

Dynamic instruction scheduling

- Key idea: allow instructions behind stall to proceed

```
DIVD     F0,F2,F4      ; takes long time
ADDD     F10,F0,F8     ; stalls waiting for F0
```



MULTD   F12,F8,F14   ; Let this instr. bypass the ADDD

- MULTD is not data dependent on anything in the pipeline
- Enables out-of-order execution ⇒ out-of-order completion
- ID stage checks for structural and data dependencies
  - Scoreboard (CDC 6600 in 1963)
  - Tomasulo (IBM 360/91 in 1967)

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- **Scoreboard**

Scoreboard pipeline

- **Goal of score-boarding** is to maintain an execution rate of CPI=1 by executing an instruction as early as possible
- **Instructions execute out-of-order** when there are sufficient resources and no data dependencies
- **A scoreboard is a hardware unit that keeps track of**
  - the **instructions** that are in the process of being executed
  - the **functional units** that are doing the executing
  - and the **registers** that will hold the results of those units
- **A scoreboard centrally performs** all hazard detection and instruction control

CDC 6000, *Seymour Cray, 1963*

Main features

- Ten functional units
- Scoreboard for dynamic scheduling of instructions
- Very fast clock, 10 MHz (FP add in 4 clocks)
- >400,000 transistors, 750 sq. ft., 5 tons, 150 kW,
- 3MIPS, **fastest machine in world for 5 years** (until CDC7600)
- over 100 sold ($6-10M each)

CDC 6000 *Seymour Cray, 1963*

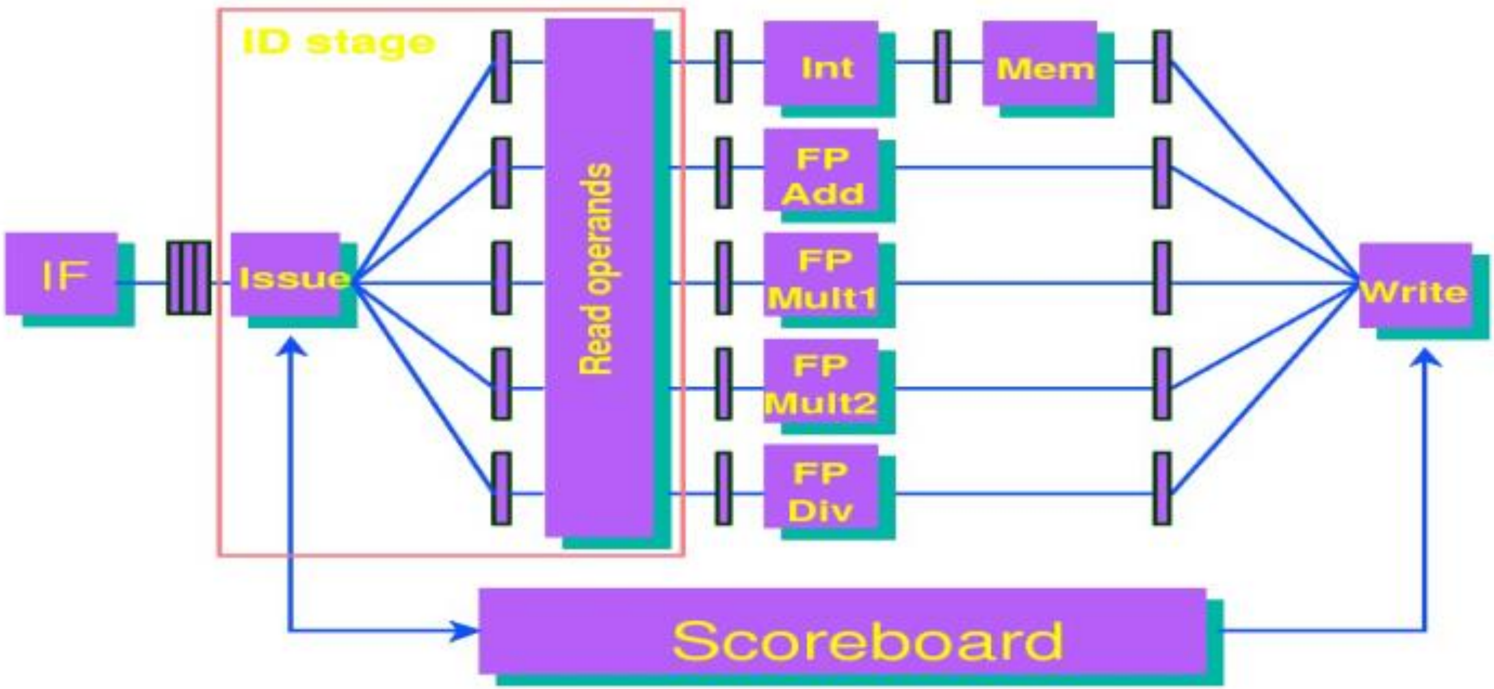
Thomas Watson Jr., IBM CEO, August 1963:

> ***"Last week, Control Data ... announced the 6600 system. I understand that in the laboratory developing the system there are only 34 people including the janitor. Of these, 14 are engineers and 4 are programmers... Contrasting this modest effort with our vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer."***

To which Cray replied: ***"It seems like Mr. Watson has answered his own question."***

Scoreboard pipeline

- **Issue:** decode and check for structural & WAW hazards
- **Read operands:** wait until no data hazards, then read operands
- **All data hazards are handled by the scoreboard**



Scoreboard complications

Out-of-order execution

⇒ WAR & WAW hazards

- **Solutions for WAR:**
  - Stall instruction in the **Write** stage until all previously issued instructions (with a WAR hazard) have read their operands
- **Solution for WAW:**
  - Stall in **Issue** until other instruction completes
- **RAW hazards handled in Read Operands**
- **Scoreboard keeps track of dependencies and state of operations**

Scoreboard functionality

- **Issue:** An instruction is issued if (**in order**):
  - The needed functional unit is free (there is no **structural hazard**)
  - No functional unit has a destination operand equal to the destination of the instruction (resolves **WAW hazards**)
- **Read:** Wait until no data hazards, then read operands
  - Performed in parallel for all functional units
  - Resolves **RAW hazards** dynamically
- **EX:** Normal execution (**out of order**)
  - Notify the scoreboard when ready
- **Write:** The instruction can update destination if:
  - All earlier instructions have read their operands (resolves **WAR hazards**)

Scoreboard components

- **Instruction status:** keeps track of which of the 4 steps the instruction is in
- **Functional unit status:** Indicates the state of the functional unit (FU). 9 fields for each FU:
  - ***Busy***: Indicates whether the unit is busy or not
  - ***Op***: Operation to perform in the unit (e.g. add or sub)
  - ***Fi***: Destination register name
  - ***Fj, Fk***: Source register names
  - ***Qj, Qk***: Name of functional unit producing regs Fj, Fk
  - ***Rj, Rk***: Flags indicating when Fj and Fk are ready
- **Register result status:** Indicates which functional unit will write each register, if any

Scoreboard example

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU								

Clock: 0

Scoreboard example, CP1

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	...	*F30*
FU				Integer				

Clock: 1

Scoreboard example, CP2

Issue 2nd load?

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Integer				

Clock: 2

Scoreboard example, CP3

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Issue MULT?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Integer				

Clock: 3

Scoreboard example, CP4

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				-				

Clock: 4

Scoreboard example, CP5

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU		Integer						

Clock: 5

Scoreboard example, CP6

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Integer						

Clock: 6

Scoreboard example, CP7

Read ops for Mult?

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Integer			Add			

Clock: 7

Scoreboard example, CP8

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? 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 result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	-			Add	Divide		

Clock: 8

Scoreboard example, CP9

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Read operands for MULT & SUB

Issue ADDD?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				Add	Divide		

Clock: 9

Scoreboard example, CP11

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

SUBD completes execution

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				Add	Divide		

Clock: 11

Scoreboard example, CP12

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Read operands for DIVD?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No							-	-
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				-	Divide		

Clock: 12

Scoreboard example, CP13

Issue ADDD

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add		Divide		

Clock: 13

Scoreboard example, CP16

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Can ADDD write result?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
3	*Mult1*	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	*Add*	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*...*	*F30*
FU	Mult1			Add		Divide		

Clock: 16

Scoreboard example, CP17

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

ADDD stalls, waiting for DIVD to read F6

Resolves a WAR hazard!

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	*Mult1*	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*...*	*F30*
FU	Mult1			Add		Divide		

Clock: 17

Scoreboard example, CP20

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	-		Yes	Yes

Register result status

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	...	*F30*
FU	-			Add		Divide		

Clock: 20

Scoreboard example, CP21

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Add		Divide		

Clock: 21

Scoreboard example, CP22

Instruction status				Issue	Read ops	Exec. compl.	Write result
Instruction		j	k				
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Now ADDD can safely write its result in F6

Functional unit status				dest	src 1	src 2	FUsrc1	FUsrc2	Fj?	Fk?
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			No	No

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				-		Divide		

Clock: 22

Scoreboard example, CP62

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU						-		

Clock: 62

Factors that limits performance

The scoreboard technique is limited by:

- **The amount of parallelism available in code**
- **The number of scoreboard entries (window size)**
  - A large window can look ahead across more instructions
- **The number and types of functional units**
  - Contention for functional units leads to structural hazards
- **The presence of anti-dependencies and output dependencies**
  - These lead to WAR and WAW hazards that are handled by **stalling** the instruction in the Scoreboard
- **Number of data-paths to registers**