

Dynamic Scheduling

- Pipelining: Issue instructions in every cycle (CPI \rightarrow 1)
- Problem: hazards delay instruction completion & increase the CPI
- Compiler scheduling (static scheduling) reduces impact of hazards
 - Increased compiler complexity, especially when attempting global scheduling (across BB's)
 - Limited information at compile time
 - Dynamically-linked libraries and OS functionality
 - Microarchitectural uncertainty: branch outcomes, memory addresses, cache misses
 - Not portable to different pipeline implementations
- Hardware scheduling so far: in-order instruction execution
 - Instructions after a stalled instruction must wait even if independent

Dynamic Scheduling: Main Idea

- Example:

```
DIV.D F0, F2, F4 ; F0=F2/F4
ADD.D F10, F0, F8 ; F10=F0+F8
SUB.D F12, F8, F14 ; F12=F8-F14
```

 - DIV.D is a long latency operation
 - ADD.D depends on DIV.D but SUB.D does not
- Solution: **out-of-order execution**
 - Detect dependence of ADD.D and block it
 - Detect that SUB.D is not dependent and execute it
 - Now SUB.D executes before ADD.D even though it comes after it in program order
 - Hardware must be able to look ahead of blocked instructions
 - Improves utilization of multiple functional units

Terminology

- **Instruction fetch**: fetch instruction from memory
- **Instruction issue**: decode instruction, check for structural hazards, and send to execution units
- **Instruction execution**: execute instruction after registers are read once dependences are cleared
- **Instruction completion** (or retire or commit): finish instruction and update processor state
- Some combinations are possible:
 - In-order issue, execution and completion
 - In-order issue and out-of-order execution and in-order completion ✓
 - Out-of-order issue, execution and completion **Impractical**

Data dependences

- Read after Write - RAW (Flow, True)
 - MUL R3, R1, R2
 - DADD R5, R3, R4

- Write after Read – WAR (Anti, Name)
 - MUL R3, R1, R2
 - DADD R1, R5, R6

- Write after Write – WAW (Output, Name)
 - MUL R3, R1, R2
 - DADD R3, R4, R5

Dynamic Scheduling 1: Scoreboarding

- Handles all RAW, WAR, and WAW with proper stalls, but allows independent instructions to proceed
- Step 1: **Issue** (part of original ID stage)
 - Issue instruction to functional unit iff functional unit is free and no earlier instruction writes to the same destination register (WAW)
- Step 2: **Read operands** (part of original ID stage)
 - Wait until source registers become available from earlier instructions through register file (RAW)
- Step 3: **Execute** (original EXE stage)
 - Execute instruction and notify scoreboard when done
- Step 4: **Write result** (original WB stage)
 - Wait until earlier instructions read operands before writing to register file (WAR)

Scoreboard Organization

- Instruction status: either one of the four steps of the instruction operation (Issue, Read Op, Execute, Write)
- Functional unit status:
 - **Busy** – functional unit is being used
 - **Op** – type of operation to be performed (e.g., add, sub, etc.)
 - **F_i** – destination register
 - **F_j, F_k** – source registers
 - **Q_j, Q_k** – functional units producing F_j and F_k
 - **R_j, R_k** – ready flag, indicates if F_j and F_k are ready but not yet read. Set to “no” after operands are read.
- Register result status: indicates which functional unit will write the register next (one per register), and also reserves register thereby detecting WAW hazards.

Scoreboarding Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	$Busy(FU) \leftarrow \text{yes}; Op(FU) \leftarrow op;$ $Fi(FU) \leftarrow 'D'; Fj(FU) \leftarrow 'S1';$ $Fk(FU) \leftarrow 'S2'; Qj \leftarrow \text{Result}('S1');$ $Qk \leftarrow \text{Result}('S2'); Rj \leftarrow \text{not } Qj;$ $Rk \leftarrow \text{not } Qk; \text{Result}('D') \leftarrow FU;$
Read operands	Rj and Rk	$Rj \leftarrow \text{No}; Rk \leftarrow \text{No}$
Execution complete	Functional unit done	
Write result	$\forall f((Fj(f) \neq Fi(FU) \text{ or } Rj(f) = \text{No}) \&$ $(Fk(f) \neq Fi(FU) \text{ or } Rk(f) = \text{No}))$	$\forall f(\text{if } Qj(f) = FU \text{ then } Rj(f) \leftarrow \text{Yes});$ $\forall f(\text{if } Qk(f) = FU \text{ then } Rj(f) \leftarrow \text{Yes});$ $\text{Result}(Fi(FU)) \leftarrow 0; Busy(FU) \leftarrow \text{No}$

WAW

WAR

Scoreboard Example

- Instruction sequence:

```
L.D    F6, 34(R2)
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
```

- Latencies:

- Integer → 1 cycle
- FP add → 2 cycles
- FP multiply → 10 cycles
- FP divide → 40 cycles

- Functional units: 1 integer (also for ld/st), 1 FP adder, 2 FP multipliers, 1 FP divider

Scoreboard example – cycle 1

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read operands</i>	<i>Execute</i>	<i>Write Result</i>
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Issue
LD #1

Functional unit status

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
1									
	<i>FU</i> Integer								

Scoreboard example – cycle 2

<u>Instruction status</u>				Read	Execution	Write	
Instruction	<i>j</i>	<i>k</i>		Issue	operat	complete	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				

LD #2 can't issue since integer unit is busy.
MULT can't issue because we require in-order issue.

<u>Functional unit status</u>			<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>	
2	<i>FU</i>	Integer									

Scoreboard example – cycle 4

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read</i>	<i>Execution</i>	<i>Write</i>	<i>Result</i>
				<i>operat</i>	<i>complete</i>		
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

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for Fj?</i> <i>Qj</i>	<i>FU for Fk?</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
4	<i>FU</i>								

Scoreboard example – cycle 5

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read operands</i>	<i>Execute</i>	<i>Write Result</i>
			1	2	3	4
LD	F6	34+	R2			
LD	F2	45+	R3	5		
MULT	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

Issue LD #2 since integer unit is now free.

Functional unit status

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest Fi</i>	<i>S1 Fj</i>	<i>S2 Fk</i>	<i>FU for j Qj</i>	<i>FU for k Qk</i>	<i>Fj? Rj</i>	<i>Fk? Rk</i>
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	<i>...</i>	<i>F30</i>
5		Integer							

Scoreboard example – cycle 6

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read	Execution	Write
				operat	complete	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				

Issue MULT.

Functional unit status

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU		Mult	Integer					

Scoreboard example – cycle 7

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read	Execution	Write
				operat	complete	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				

MULT can't read its operands (F2) because LD #2 hasn't finished.

Functional unit status

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
7		FU							
		Mult	Integer		Add				

Scoreboard example – cycle 8

DIVD issues.
MULT and SUBD
both waiting for F2.
LD #2 writes F2.

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read operand	Executi comple	Write Result	
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	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 <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
8	<i>FU</i>	Mult1			Add	Divide			

Scoreboard example – cycle 9

Now MULT and SUBD can both read F2.

<u>Instruction status</u>				Read	EX	Write
Instruction	<i>j</i>	<i>k</i>	<i>Issue Op</i>	<i>Op</i>	<i>compl</i>	<i>Result</i>
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				

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for j</i>	<i>FU for k</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
9	<i>FU</i>	Mult1			Add	Divide			

Scoreboard example – cycle 11

<u>Instruction status</u>				<i>Read</i>	<i>Execu</i>	<i>Write</i>
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	8
MUL	F0	F2 F4	6	9		
SUB	F8	F6 F2	7	9	11	
DIV	F10	F0 F6	8			
ADD	F6	F8 F2				

ADDD can't start because add unit is busy.

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for j</i>	<i>FU for k</i>	<i>Fj?</i>	<i>Fk?</i>	
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	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

Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
11	<i>FU</i>	Mult1				Add	Divide			

Scoreboard example – cycle 12

<u>Instruction status</u>				<i>Read</i>	<i>Execu</i>	<i>Write</i>
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>
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				

SUBD finishes.
DIVD waiting for F0.

<u>Functional unit status</u>		<i>Busy</i>	<i>Op</i>	<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for</i>	<i>FU for Fj?</i>	<i>Fk?</i>
<i>Tim</i>	<i>Name</i>	<i>Op</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
12	<i>FU</i>	Mult1				Divide			

Scoreboard example – cycle 13

<u>Instruction status</u>				<i>Read</i>	<i>Execu</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operai</i>	<i>compl</i>	<i>Result</i>	
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				

ADDD issues.

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for j</i>	<i>FU for k</i>	<i>Fj?</i>	<i>Fk?</i>
<i>Tim</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
13	<i>FU</i>	Mult1		Add		Divide			

Scoreboard example – cycle 14

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>
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		

Functional unit status

<i>Tim</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for Fj?</i>	<i>Fk?</i>
				<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>
	Integer	No						
5	Mult1	Yes	Mult	F0	F2	F4		No
	Mult2	No						No
2	Add	Yes	Add	F6	F8	F2		No
	Divide	Yes	Div	F10	F0	F6	Mult1	No

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
14	FU Mult1			Add		Divide			

Scoreboard example – cycle 16

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read</i>	<i>Execu</i>	<i>Write</i>	<i>Result</i>
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		

Functional unit status

<i>Tim</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for j</i>	<i>FU for k</i>	<i>Fj?</i>	<i>Fk?</i>
				<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
16	<i>FU</i>	Mult1		Add		Divide			

Scoreboard example – cycle 17

Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read Operat	Execu 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	

*This is fixable!
(anti-dependance)*

ADDD can't write because of DIVD WAR!

Functional unit status

Time	Name	Busy Op	dest <i>Fi</i>	<i>S1 Fj</i>	<i>S2 Fk</i>	<i>FU for Qj</i>	<i>FU for Qk</i>	<i>Fj? Rj</i>	<i>Fk? Rk</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
17	FU Mult1			Add		Divide			

Scoreboard example – cycle 19

Instruction status

Read Execu Write

Instruction *j k Issue operai compl Result*

LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

MULT completes execution.

Functional unit status

dest S1 S2 FU for FU for Fj? Fk?

Time Name

Busy Op Fi Fj Fk Qj Qk Rj Rk

Integer	No							
0 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

Clock
19

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Divide			

Scoreboard example – cycle 20

Instruction	<i>j</i>	<i>k</i>	Issue	operat	compl	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

MULT writes.

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for Fj?</i>	<i>Fk?</i>
<i>Tim</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj Qk Rj Rk</i>
	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

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
20				Add		Divide			

Scoreboard example – cycle 21

Instruction	<i>j</i>	<i>k</i>	Issue	Operands	Completed	Result
LD	F6	34+	R2		1	2
LD	F2	45+	R3		5	6
MULTD	F0	F2	F4		6	9
SUBD	F8	F6	F2		7	9
DIVD	F10	F0	F6		8	21
ADDD	F6	F8	F2		13	14
						16

DIVD loads operands

Functional unit status		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for Fj?</i>	<i>Fk?</i>
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>
			<i>Rj</i>	<i>Qk</i>	<i>Rk</i>	
	Integer	No				
	Mult1	No				
	Mult2	No				
	Add	Yes	Add	F6	F8	F2
	Divide	Yes	Div	F10	F0	F6
						No
						No

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
21					Add	Divide			

Scoreboard example – cycle 22

Instruction	<i>j</i>	<i>k</i>	Issue	Operat	compl	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	22

Now ADDD can write since WAR removed.

Functional unit status

Time	Name	Busy	Op	dest	S1	S2	FU for Fj?	Fk?
		Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No						
	Mult1	No						
	Mult2	No						
	Add	No						
	Divide	Yes	Div	F10	F0	F6	No	No

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
21						Divide			

Scoreboard example – cycle 61

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operat</i>	<i>compl</i>	<i>Result</i>	
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	
ADDD	F6	F8	F2	13	14	16	22

DIVD completes execution

<u>Functional unit status</u>		<i>dest</i>	<i>S1</i>	<i>S2</i>	<i>FU for Fj?</i>	<i>Fk?</i>			
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer	No							
	Mult1	No							
	Mult2	No							
	Add	No							
	Divide	Yes	Div	F10	F0	F6		No	No

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
61	<i>FU</i>					Divide			

Scoreboard example – cycle 62

Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read operands</i>	<i>Execute</i>	<i>Write Result</i>
LD	F6	34+	R2	1	2	3 4
LD	F2	45+	R3	5	6	7 8
MULT	F0	F2	F4	6	9	19 20
SUB	F8	F6	F2	7	9	11 12
DIV	F10	F0	F6	8	21	61 62
ADD	F6	F8	F2	13	14	16 22

Functional unit status

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	0 Divide	No								

Register result status

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
62	<i>FU</i>								

Scoreboard Summary

- Dynamically schedules instructions
- Forces instructions to wait on RAW, WAR, WAW dependences and structural hazards
- First used in the CDC 6600 in 1964 and yielded performance improvements of 1.7x to 2.5x
- Hardware cost (size) of scoreboard equivalent to one of the functional units

Scoreboard Limitations

- No forwarding – read from register
 - Mitigated by a shorter “back end” of the pipeline (i.e., few stages after issue) and out-of-order execution
- Structural hazards – stall at issue
- WAW hazard – stall at issue
- WAR hazard – stall at write

Next lecture: Dynamic scheduling using Tomasulo’s algorithm

- Avoids WAW & WAR via register renaming
- Supports forwarding using a centralized result bus