POLITECNICO DI MILANO

High Performance Processors and Systems

Instruction Level Parallelism

Introduction to ILP and Scoreboard

HPPS

Outline

- Dependences and hazards
- Static and dynamic scheduling
- Hardware schemes for ILP
- Scoreboard

Definition of ILP

- **ILP = Potential overlap of execution among unrelated instructions**

- Overlapping possible if:
    - No Structural Hazards
    - No RAW, WAR of WAW Stalls
    - No Control Stalls

Some basic concepts and definitions

- To reach higher performance (for a given technology) – more parallelism must be extracted from the program. In other words...

- Dependences must be detected and solved, and instructions must be *ordered* (***scheduled***) so as to achieve highest parallelism of execution compatible with available resources.

Instruction Level Parallelism

- Two strategies to support ILP:
  - **Dynamic Scheduling:** Depend on the hardware to locate parallelism
  - **Static Scheduling:** Rely on software for identifying potential parallelism
- Hardware intensive approaches dominate desktop and server markets

Dynamic Scheduling

- The hardware reorders the instruction execution to reduce pipeline stalls while maintaining data flow and exception behavior.
- Main advantages:
  - It enables handling some cases where dependences are unknown at compile time
  - It simplifies the compiler complexity
  - It allows compiled code to run efficiently on a different pipeline.
- Those advantages are gained at a cost of a significant increase in hardware complexity and power consumption.

Detection and resolution of dependences: Dynamic Scheduling

- Simple pipeline: hazards due to data dependences that cannot be hidden by forwarding *stall* the pipeline – no new instructions are fetched nor issued.
- ***Dynamic scheduling***: Hardware reorders instruction execution so as to reduce stalls, maintaining data flow and exception behaviour.

Detection and resolution of dependences: Dynamic Scheduling (2)

- Basically: Instructions are *fetched* and *issued* ***in program order*** (in-order-issue)
- Execution begins ***as soon as operands are available*** - possibly, ***out of order*** (implies ***out of order completion***) – note: ***possible even with pipelined scalar architectures***.
- Out-of order execution introduces possibility of WAR, WAW data hazards.

Getting higher performance: Hardware-based techniques (2)

Technique	Reduces
Dynamic scheduling	Data hazard stalls
Dynamic branch pred.	Control stalls
Multiple issue	CPI_{ideal}
Speculation	Data and control stalls

Key Idea: dynamic scheduling

- Problem:
    - data dependences that cannot be hidden with bypassing or forwarding cause hardware stalls of the pipeline
- Solution: allow instructions behind a stall to proceed
    - Hw rearranges the instruction execution to reduce stalls
- Enables out-of-order execution and completion (commit)
- First implemented in CDC 6600 (1963).

Dynamic scheduling

- Advantages:
  - Enables handling cases of dependence unknown at compile time
  - Simplifies compiler
  - Allows code compiled for one pipeline to run efficiently on a different pipeline
- Cost: significant increase in hw complexity

Example

```
DIVD F0,F2,F4
ADDD F10,F0,F8
SUBD F12,F8,F14
```

ADDD stalls for F0 (waiting that DIVD commits)

SUBD would stall even if not data dependent on anything in the pipeline without dynamic scheduling.

Problems?

- How do we prevent WAR and WAW hazards?
- How do we deal with variable latency?
  - Forwarding for RAW hazards harder.

	Clock Cycle Number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD F6,34(R2)	IF	ID	EX	MEM	WB												
LD F2,45(R3)		IF	ID	EX	MEM	WB											
MULTD F0,F2,F4			IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	MEM	WB
SUBD F8,F6,F2				IF	ID	A1	A2	MEM	WB								
DIVD F10,F0,F6					IF	ID	stall	stall	stall	stall	stall	stall	stall	stall	stall	D1	D2
ADDD F6,F8,F2						IF	ID	A1	A2	MEM	WB						

RAW

WAR

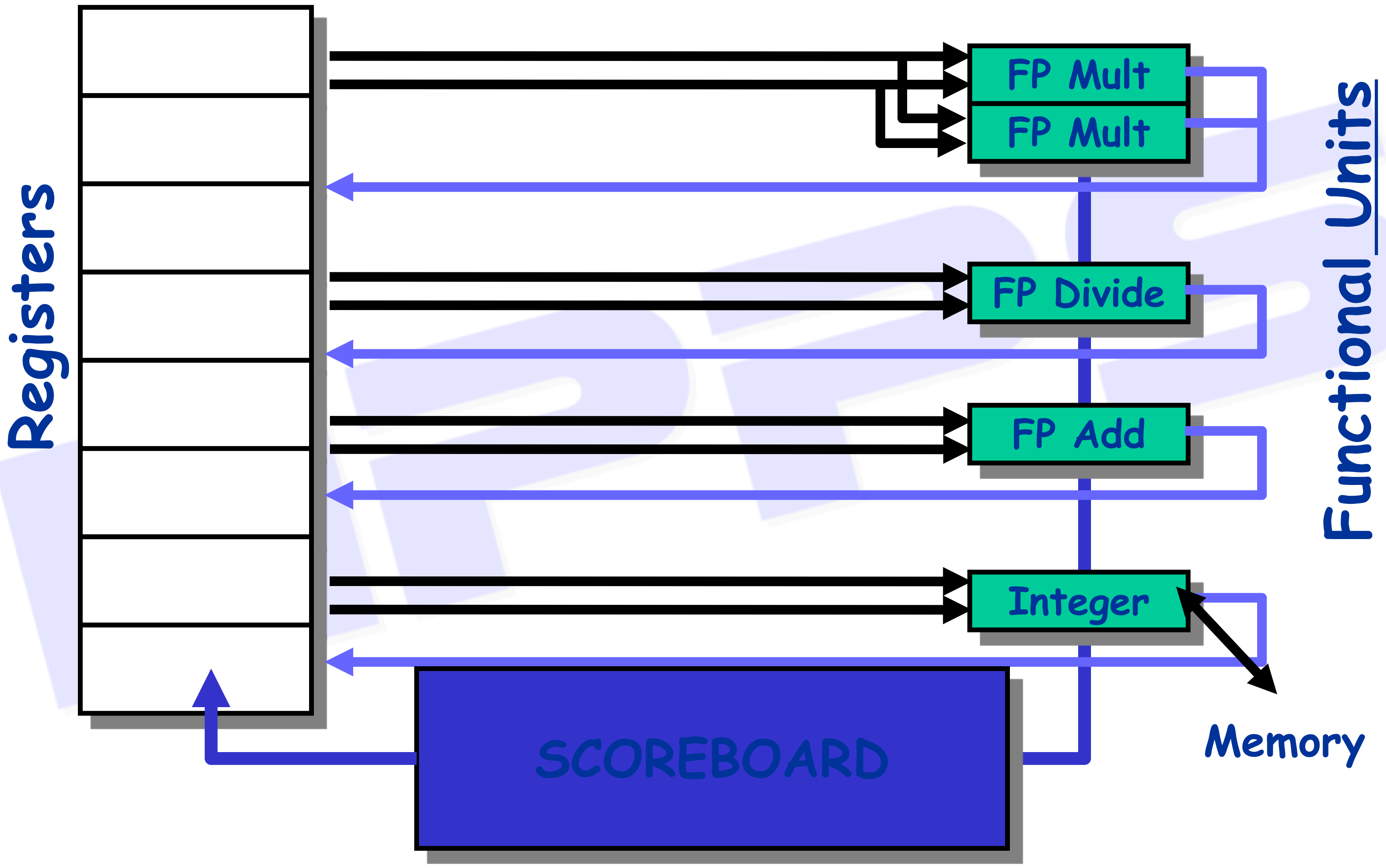
Scoreboard basic scheme

- Out-of-order execution divides ID stage:

  1. Issue—decode instructions, check for structural hazards
  2. Read operands—wait until no data hazards, then read operands

- Instructions execute whenever not dependent on previous instructions and no hazards
- Scoreboard allows instructions to execute whenever 1 & 2 hold, not waiting for prior instructions
- CDC 6600 (1963): In order issue, out of order execution, out of order completion (commit)
  - No forwarding!
  - Imprecise interrupt/exception model for now!

Scoreboard Architecture (CDC 6600)



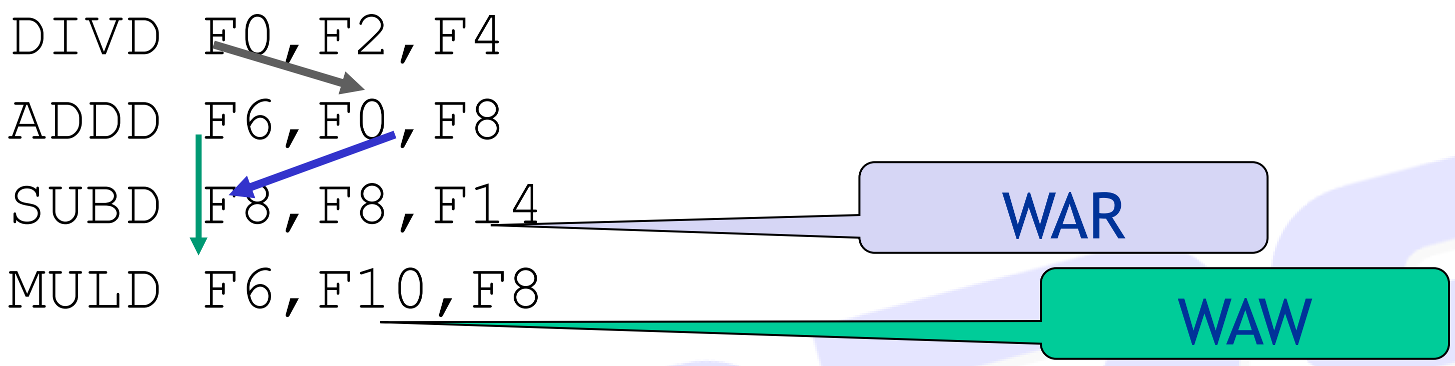
Scoreboard Scheme

- Scoreboard replaces ID, EX, WB with 4 stages
- ID stage splitted in two parts:
  - Issue (decode and check structural hazard)
  - Read Operands (wait until no data hazards)
- Scoreboard allows instructions without dependencies to execute
- In-order *issue* BUT out-of-order *read-operands*

Scoreboard Implications

- Out-of-order completion -> WAR and WAW hazards?
- Solutions for WAR:
  - Stall write back until registers have been read.
  - Read registers only during Read Operands stage.

WAR/WAW Example



```
DIVD F0,F2,F4
ADDD F6,F0,F8
SUBD F8,F8,F14
MULD F6,F10,F8
```

The scoreboard would stall:

- SUBD in the WB stage, waiting that ADDD reads F0 and F8 and
- MULD in the issue stage until ADDD writes F6.

Can be solved through register renaming

Scoreboard Implications

- Solution for WAW:
  - Detect hazard and stall issue of new instruction until the other instruction completes
- No register renaming
- Need to have multiple instructions in execution phase ➔ Multiple execution units or pipelined execution units
- Scoreboard keeps track of dependences and state of operations

Exception handling

- Problem with out-of order completion
  - Must preserve exception behavior as in-order execution

- Solution:
  ensure that no instruction can generate an exception until the processor knows that the instruction raising the exception will be executed

Imprecise exceptions

- An exception is imprecise if the processor state when an exception is raised does not look exactly as if the instructions were executed in-order.
  - The pipeline may have ***already*** completed instructions that are *later* in program order than the instruction causing the exception
  - The pipeline may have ***not yet*** completed some instructions that are *earlier* in program order than the instruction causing the exception
- Imprecise exception make it difficult to restart execution after handling

Four Stages of Scoreboard Control

1. **Issue**
   Decode instructions & check for structural hazards.

- ✓ Instructions issued in program order (for hazard checking)
- ✓ If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure.
- ✓ If a structural or a WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.

Four Stages of Scoreboard Control

2. Read Operands

Wait until no data hazards, then read operands

A source operand is available if:

- no earlier issued active instruction will write it or
- A functional unit is writing its value in a register

When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution.
RAW hazards are resolved dynamically in this step, and instructions may be sent into execution out of order.

No forwarding of data in this model

Four Stages of Scoreboard Control

3. **Execution**

Operate on operands

The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

FUs are characterized by:

- **latency** (the effective time used to complete one operation)
- **Initiation interval** (the number of cycles that must elapse between issuing two operations to the same functional unit).

Four Stages of Scoreboard Control

4. **Write result**

   Finish execution

   Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction.

   Assume we can overlpa issue and write

Scoreboard structure: three parts

1. **Instruction status**
2. **Functional Unit status**

   Indicates the state of the functional unit (FU):

   Busy – Indicates whether the unit is busy or not
   Op - The operation to perform in the unit (+,-, etc.)
   Fi - Destination register
   Fj, Fk – Source register numbers
   Qj, Qk – Functional units producing source registers
   Rj, Rk – Flags indicating when Fj, Fk are ready
3. **Register result status**

   Indicates which functional unit will write each register.
   Blank if no pending instructions will write that register.

Detailed Scoreboard Pipeline Control

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

Scoreboard Example

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
	FU									

Scoreboard Example: Cycle 1

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock
1

	F0	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
FU				Integer					

µ-LAB POLITECNICO DI MILANO

Scoreboard Example: Cycle 2

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*				Integer					

- Issue 2nd LD?
- Integer Pipeline Full – Cannot exec 2nd Load – Issue stalls

Scoreboard Example: Cycle 3

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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				

Functional unit status:

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

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*				Integer					

- **Issue MULT?** Issue stalls

Scoreboard Example: Cycle 4

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*				Integer					

Issue stalls

Write F6

Scoreboard Example: Cycle 5

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*		Integer							

The 2nd load is issued

Scoreboard Example: Cycle 6

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				Yes
	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*	Mult1	Integer							

MULT is issued but has to wait for F2

Scoreboard Example: Cycle 7

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	S1 Fj	S2 Fk	FU Qj	FU 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:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	Integer			Add				

Read multiply operands?

Now SUBD can be issued but has to wait for operands

Scoreboard Example: Cycle 8a (First half of clock cycle)

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU 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	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	*FU*	Mult1	Integer			Add	Divide			

DIVD is issued but there is another RAW hazard (F0) then DIVD has to wait for F0

µ-LAB POLITECNICO DI MILANO

Scoreboard Example: Cycle 8b (Second half of clock cycle)

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU 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:

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

Load completes, and operands for MULT an SUBD are ready

Scoreboard Example: Cycle 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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				

Functional unit status:

Note Remaining

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? 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 result status:

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

Read operands for MULTD & SUBD

Issue ADDD? No for structural hazard on ADD Functional Unit

MULTD and SUBD are sent in execution in parallel

Scoreboard Example: Cycle 10

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*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				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
9	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU 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:

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

SUBD ends

Scoreboard Example: Cycle 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU 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:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	*FU*	Mult1					Divide			

Read operands for DIVD?

Scoreboard Example: Cycle 13

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU 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:

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

SUBD writes results and ADDD can be issued

Scoreboard Example: Cycle 14

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*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		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 15

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 16

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU 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:

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

Scoreboard Example: Cycle 17

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	

WAR Hazard!

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU 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:

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

Why not write result of ADD???

DIVD must first read F6 but cannot read until MULTD writes F0

Scoreboard Example: Cycle 18

Instruction status:

Instruction	j	k		*Issue*	*Read Oper*	*Exec Comp*	*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	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
1	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		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
18	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 19

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	
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	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? 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		F0	F2	F4	F6	F8	F10	F12	...	F30
19	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 20

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU 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:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
20	*FU*				Add		Divide			

Scoreboard Example: Cycle 21

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	S1 Fj	S2 Fk	FU Qj	FU 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:

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

WAR Hazard is now gone...

Scoreboard Example: Cycle 22

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	22

Functional unit status:

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

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
22	*FU*						Divide			

Now DIVD has read its operands, ADDD can write the result in F6

Faster than light computation (skip a couple of cycles)

Scoreboard Example: Cycle 61

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	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	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
61	*FU*						Divide			

DIVD ends execution

Scoreboard Example: Cycle 62

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
62	*FU*									

DIVD writes in F10

Review: Scoreboard Example: Cycle 62

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*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*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
62	*FU*									

In-order issue; out-of-order execute & commit

CDC 6600 Scoreboard

- Speedup of 2.5 w.r.t. no dynamic scheduling
- Speedup 1.7 by reorganizing instructions from compiler
- BUT slow memory (no cache) limits benefit
- **Limitations of 6600 scoreboard:**
  - No forwarding hardware
  - Limited to instructions in basic block (small *window*)
  - Small number of functional units (structural hazards), especially integer/load store units
  - Do not issue on structural hazards
  - Wait for WAR hazards
  - Prevent WAW hazards

Summary

- Instruction Level Parallelism (ILP) in SW or HW
- Loop level parallelism is easiest to see
- SW parallelism dependencies defined for program, hazards if HW cannot resolve
- SW dependencies/compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine
- HW exploiting ILP
  - Works when can't know dependence at run time
  - Code for one machine runs well on another
- **Key idea of Scoreboard:** Allow instructions behind stall to proceed (Decode $\Rightarrow$ Issue Instruction & Read Operands)
  - Enables out-of-order execution => out-of-order completion
  - ID stage checked both structural and WAW hazards