Course on: “Advanced Computer Architectures”

Instruction Level Parallelism Part II - Scoreboard

Prof. Cristina Silvano
Politecnico di Milano
email: cristina.silvano@polimi.it

Basic Assumptions

- We consider ***single-issue*** processors
- The Instruction Fetch stage might fetch either into an Instruction Register or into a queue of pending instructions
- Instructions are then issued from the IR or from the queue
- Execution stage may require ***multiple cycles,*** depending on the operation type.
- Memory stage might require ***multiple cycles*** access time due to data cache misses

Key Idea: Dynamic Scheduling

- **Problem:** Hazards due to data dependences that cannot be solved by forwarding cause **stalls** of the pipeline: no new instructions are fetched nor issued even if they are not data dependent

- **Solution: Allow data independent instructions behind a stall to proceed**
  - HW rearranges dynamically the instruction execution to reduce stalls

- **Enables out-of-order execution and completion (commit)**

- First implemented in CDC 6600 (1963).

Example

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

- RAW Hazard: ADDD stalls for F0 (waiting that DIVD commits).
- SUBD would stall even if not data dependent on anything in the pipeline without dynamic scheduling.
- **BASIC IDEA: to enable SUBD to proceed (out-of-order execution)**

Dynamic Scheduling

- **Main advantages (PROs):**
  - Enables handling cases of dependence unknown at compile time
  - Simplifies compiler
  - Allows compiled code to run efficiently on a different pipeline (code portability)

- **Disadvantages (CONs):**
  - Significant increase in hardware complexity
  - Increased power consumption
  - Could generate *imprecise* exceptions

Scoreboard Dynamic Scheduling Algorithm

Scoreboard basic scheme

- **Out-of-order execution divides ID stage:**
  1. **Issue—Decode instructions, check for structural hazards**
  2. **Read operands (RR)—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

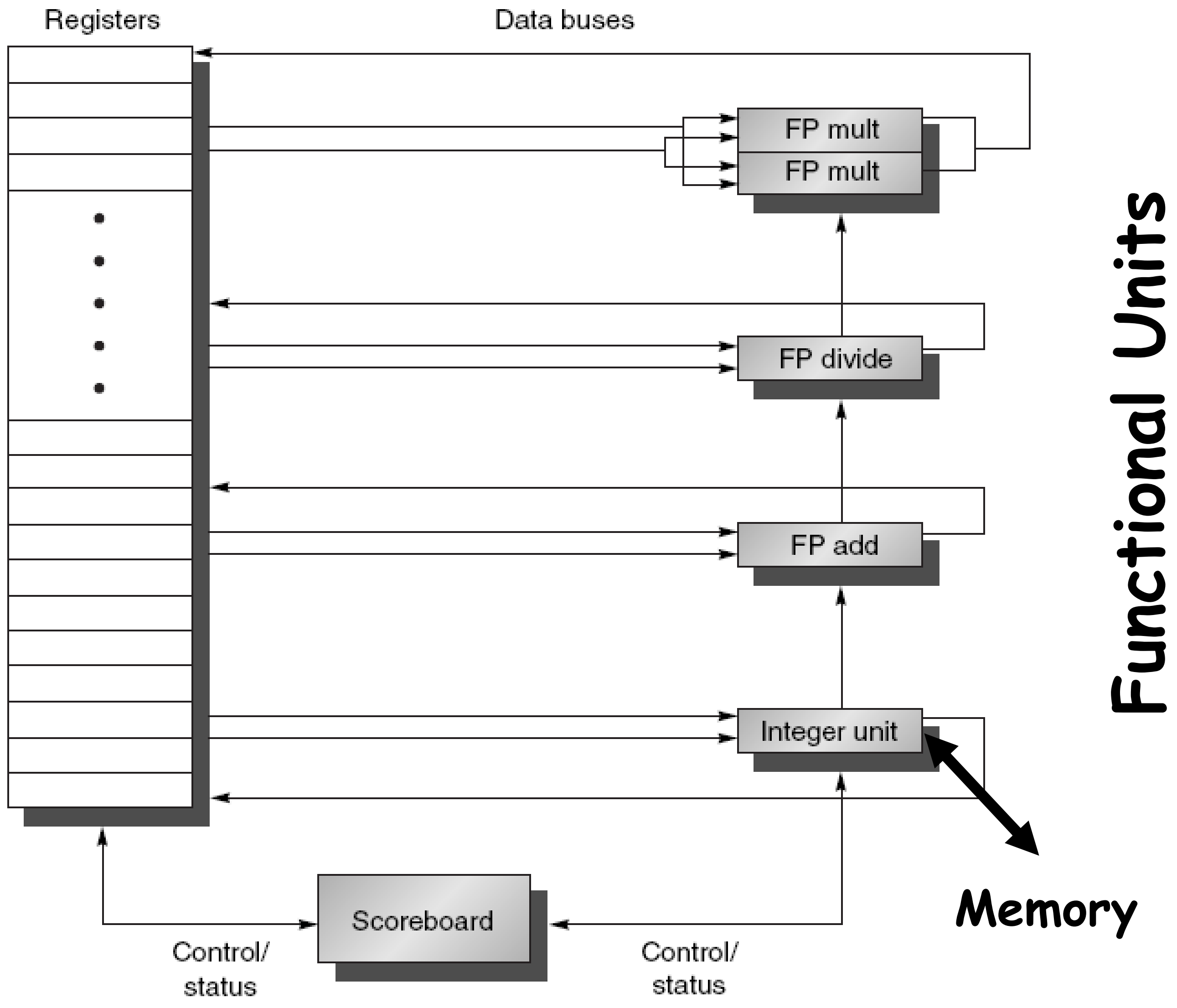
Scoreboard basic scheme

- We distinguish when an instruction begins execution and it completes execution: between the two times, the instruction is ***in execution.***

- We assume the pipeline allows multiple instructions in execution at the same time ⇒ that requires multiple functional units, pipelined functional units or both.

- **CDC 6600: In order issue, out of order execution, out of order completion (commit)**
  - *No forwarding!*
  - *Imprecise interrupt/exception model for now!*

Scoreboard Architecture



Scoreboard Pipeline

- Scoreboard replaces ID, EX, WB stages with 4 stages
- **ID stage split 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* ⇒ *out-of-order execution and completion***
- All instructions pass through the issue stage in-order, but they can be stalled or bypass each other in the read operand stage and thus enter execution out-of-order and with different latencies, which implies out-of-order completion.

Scoreboard Implications

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

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 dependencies and state of operations

Scoreboard Scheme

- **Hazard detection and resolution is centralized in the scoreboard:**
  - Every instruction goes through the Scoreboard, where a record of data dependences is constructed
  - The Scoreboard then determines when the instruction can read its operand and begin execution
  - If the scoreboard decides the instruction cannot execute immediately, it monitors every change and decides when the instruction can execute.
  - The scoreboard controls when the instruction can write its result into destination register

Four Stages of Scoreboard Control

1. **Issue**

   Decode instruction and check for structural hazards & WAW 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 (no WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure.
   - If a structural hazard 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 RAW hazards, then read operands.
   Check for structural hazards in reading RF.

- 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 solved dynamically in this step => out-of-order reading of operands then instructions are sent into execution out-of-order.**
- No forwarding of data in this model

Four Stages of Scoreboard Control

3. **Execution**

   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:
  - **Variable latency** (the effective time used to complete one operation).
  - **Load/Store latency depends on data cache HIT/MISS**

Four Stages of Scoreboard Control

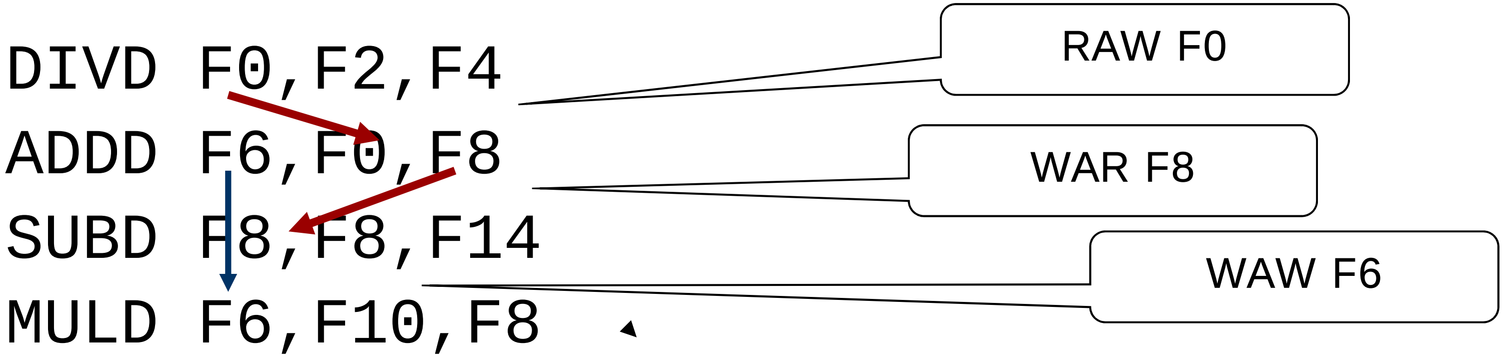
4. **Write result**

Check for WAR hazards and 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 completing instruction.

WAR/WAW Example



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

RAW F0

WAR F8

WAW F6

- The scoreboard would:
  - Stall SUBD in the WB stage, waiting for ADDD reads F0 and F8 and
  - Stall MULD in the issue stage until ADDD writes F6.
- Any WAR/WAW hazards can be solved through register renaming

SCOREBOARD BASIC SCHEME

- IN-ORDER ISSUE
- OUT-OF-ORDER READ OPERANDS
- OUT-OF-ORDER EXECUTION
- OUT-OF-ORDER COMPLETION
- NO FORWARDING
- Control is centralized into the Scoreboard

SCOREBOARD STAGES

- **ISSUE (IN-ORDER):**
  - Check for structural hazards
  - Check for WAW hazards on destination ops
- **READ OPERANDS (OUT-OF-ORDER)**
  - Check for RAW hazards
  - Check for structural hazards in reading RF
- **EXECUTION (OUT-OF-ORDER)**
  - Execution completion depends on latency of FUs
  - Execution completion of LD/ST depends on cache hit/miss latencies)
- **WRITE RESULTS (OUT-OF-ORDER)**
  - Check for WAR hazards on destionation ops
  - Check for structural hazards in writing RF

SCOREBOARD optimisations

- Check for WAW postponed to WRITE stage instead of in ISSUE stage
- Forwarding

Scoreboard Structure

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 Fj, Fk

   Rj, Rk – Flags indicating when Fj, Fk are ready.
   Flags are set to NO after operands are read.
3. Register result status.

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

Scoreboard Example:
Analysis of dependences and hazards

```
LD F6, 34(R2)
LD F2, 45(R3)
MULTD F0, F2, F4  # RAW F2
SUBD F8, F6, F2   # RAW F2,RAW F6
DIVD F10, F0, F6  # RAW F0,RAW F6
ADDD F6, F8, F2   # WAR F6,RAW F8,RAW F2
```

Scoreboard Example

Instruction status:

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

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
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 result status:

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

Scoreboard Example Cycle 2

Instruction status					Read	Executic	Write
Instruction		j	k	Issue	operand	complet	Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
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 result status									
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU				Integer					

Issue 2nd load?

Integer Pipeline Full – Cannot exec 2nd Load due to structural hazard on Integer Unit – Issue stalls

Scoreboard Example Cycle 3

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
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 result status

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

- **Issue stalls**
- **Load execution complete in one clock cycle (data cache hit)**

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:

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

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
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 result status:

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

- **The second load is issued**

Scoreboard Example: Cycle 6

Instruction status:

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*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 from LOAD (RAW Hazard on 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				Yes
	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 to ADD Functional Unit (then SUBD has to wait for RAW F2 from load)**

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) from MULTD -> DIVD has to wait for F0

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

Instruction status:

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
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 result status:

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

- **Load completes (Writes F2), and operands for MULT an SUBD are ready**

Scoreboard Example: Cycle 9

Instruction status:

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
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 result status:

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

- Read operands for MULTD & SUBD by multiple-port Register File
- Issue ADDD? No for structural hazard on ADD Functional Unit
- MULTD and SUBD are sent in execution in parallel:
  Latency of 10 cycles for MULTD and 2 cycles for SUBD

Scoreboard Example: Cycle 10

Instruction status:

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

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

Scoreboard Example: Cycle 11

Instruction status:

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
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 result status:

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

- **SUBD ends execution**

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			Yes	Yes
	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			

- SUBD writes result in F8

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			Yes	Yes
	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			

- ADDD can be issued
- DIVD still waits for operand F0 from MULTD

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			Yes	Yes
	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			

- **ADDD reads operands (out-of-order read operands: ADDD reads operands before DIVD)**

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			Yes	Yes
	Mult2	No								
1	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*
15	*FU*	Mult1			Add		Divide			

- ***ADDD starts execution***

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			Yes	Yes
	Mult2	No								
0	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*
16	*FU*	Mult1			Add		Divide			

- ADDD ends execution

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 F6 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			Yes	Yes
	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*
17	*FU*	Mult1			Add		**Divide**			

- Why not write result of ADDD??? WAR must be detected before write result of ADDD in F6
- DIVD must first read F6 (before ADDD write F6), but DIVD cannot read operands until MULTD writes F0 (RAW on F0)

Scoreboard Example: Cycle 18

Instruction status:

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
1	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 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			Yes	Yes
	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*
19	*FU*	Mult1			Add		Divide			

- **MULTD ends execution**

Scoreboard Example: Cycle 20

Instruction status:

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

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

Register result status:

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

- **MULTD writes in F0**

Scoreboard Example: Cycle 21

Instruction status:

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

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

Register result status:

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

- **DIVD can read operands**
- **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:

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

Register result status:

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

- DIVD has read its operands in previous cycle
- ADDD can now write the result in F6

(skipping some 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**			**Yes**	**Yes**

Register result status:

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

- DIVD ends execution

Scoreboard Example: Cycle 62

Instruction status:

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

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

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

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*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 ⇒ Issue Instruction & Read Operands)
  - Enables out-of-order execution => out-of-order completion
  - ID stage checked both structural and WAW hazards on destination operands.

References:

Chapter 2 of the text book: J. Hennessey, D. Patterson, "*Computer Architecture: a quantitative approach*" *4th* Edition, Morgan-Kaufmann Publishers.