Computer Architecture (Fall 2022)

Instruction-Level Parallelism

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

Instruction-Level Parallelism

- ILP: overlap the execution of instructions
  - partially (through pipelining)
  - completely (through issuing on multiple functional units)
- Exploiting ILP
  - dynamically and hardware-intensive
    - mostly desktop and server markets (Pentium, MIPS, Alpha…)
  - static and software-intensive
    - embedded system markets (but also, Itanium...)
  - in practice these techniques are often combined
- Exploiting ILP and basic blocks
  - in a typical MIPS program, the average dynamic branch frequency is between 15% and 25%→average length of a basic block is between 4 and 7 instructions
    - necessary to exploit ILP across multiple basic blocks

Reducing CPI (or Increasing IPC)

$$CPI = CPI_{ideal} + stalls_{structural} + stalls_{data\ Hazard} + stalls_{control}$$

technique	reduces
forwarding/ bypassing	potential data-hazard stalls
delayed branches	control-hazard stalls
basic dynamic scheduling (scoreboarding)	data-hazard stalls from true dependecies
dynamic scheduling with register renaming	data-hazard, anti-dep. & output dep. stalls
dynamic branch prediction	control stalls
issuing multiple instruction per clock cycle	ideal CPI
speculation	data-hazard and control-hazard stalls
dynamic memory disambiguation	data-hazard stalls with memory
loop unrolling	control hazard stalls
basic compiler pipeline scheduling	data-hazard stalls
compiler dependency analysis	ideal CPI, data-hazard stalls
software pipelining & trace scheduling	ideal CPI, data-hazard stalls
compiler speculation	ideal CPI, data-hazard stalls

Dependences vs. Hazards

- If two instructions depend on each other
  - they must be executed in order
  - at most, they can be partially overlapped
- Dependences are a property of programs
  - a data dependency in the instruction sequence reflects a data dependency in the original source code
- Based on the given pipeline implementation a data dependency may result in an actual hazard being detected and a possible stall (which reduces or eliminates the instruction overlap)

e.g. moving the branch test for the MIPS from the EX to ID does reduce the branch delay to one, but it may cause a stall in presence of a data dependency

```
DADDIU  R1,  R8,  -8
BNE  R1,  R2,  Loop
```

Dependences vs. Hazards (cont.)

- Dependency ≡ hazard potential
  - occurrence of actual hazard and length of any stall is a property of the pipeline implementation
- Dependences determine the order in which results must be computed
- Dependences set an upper bound on the amount of instruction-level parallelism that can be exploited

- Strategies to overcome dependences
  - maintain them, but avoid hazards
  - eliminate them by transforming the code

Dynamic HW Scheduling vs. Static Pipeline Scheduling

- Dynamic Scheduling (HW)
  - tries to avoid stalls when dependences (which could generate hazards) are present
  - does not change the data flow
- Static Scheduling (SW)
  - the compiler tries to minimize stalls by separating dependent instructions so that they will not generate hazards
- The two techniques can be combined as statically scheduled code can run on a processor with a dynamically scheduled pipeline

Dynamic HW Scheduling: Motivations


```
DIV.D  F0,  F2,  F4
ADD.D  F6,  F0,  F8
SUB.D  F8,  F8,  F14
```

- SUB.D does not depend on the previous instructions but it can not execute because ADD.D is stalling due to RAW hazard
- With dynamic scheduling, the HW rearranges the instruction execution order to reduce the stalls while maintaining data flow and exception behavior
  - it simplifies compiling
    - code compiled for a given pipeline can be run efficiently on another
  - it handles cases where dependences are unknown at compile time (due to memory references)
  - it is the basis for hardware speculative execution

Out-of-Order Execution

- In-Order Execution: if an instruction is stalled in the pipeline, no later instructions can proceed.
  - Structural and Data hazards are checked at ID.
  - Even if there are later instructions that are independent and would not stall.
- Out-of-Order execution: Decode and Issue instructions in order, but execute the instructions as soon as their data operands are available.
  - It may result in out-of-order completion.
- To implement out-of-order execution, we must split the ID into two stages:
  - Issue—Decode instructions, check for structural hazards.
  - Read operands—Wait until no data hazards, then read operands
- In a dynamically scheduled pipeline:
  - All instructions pass through the issue stage in order (in-order issue);
  - However, they can be stalled or bypass each other in the second stage (read operands) and thus enter execution out-of-order.

Name Dependences and the New Hazards due to Out-of-Order Execution

- True data dependency: an instruction produces a value for a following instruction
  - may lead to a RAW hazard
- Name dependence: no real value must be transmitted between two instructions, but they both use the same register or memory location
  - anti-dependence
    - may lead to a WAR hazard
  - output dependence
    - may lead to a WAW hazard

```
DIV.D  F0,  F2,  F4
ADD.D  F6,  F0,  F8
```

```
ADD.D  F6,  F0,  F8
SUB.D  F8,  F8,  F14
```

```
ADD.D  F6,  F0,  F8
MUL.D  F6,  F10,  F9
```

Dynamic Scheduling & Data Hazards: The Scoreboard Approach

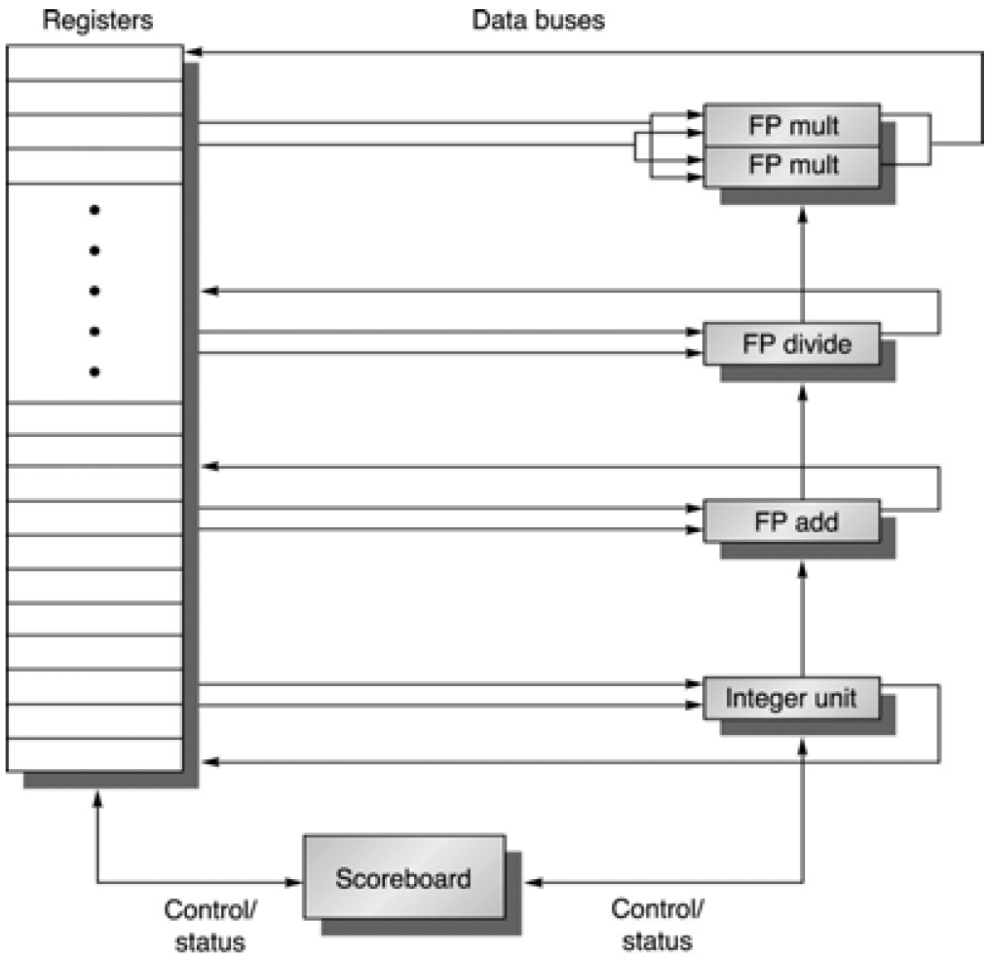

```
DIV.D   F0,  F2,  F4
ADD.D   F6,  F0,  F8
SUB.D   F8,  F8,  F14
MUL.D   F6,  F10, F9
```

- anti-dependency between ADD.D and SUB.D (may lead to WAR hazard)
- output dependency between MUL.D and ADD.D (may lead to WAW hazard)

- Goal: to maintain CPI≈1 with out-of-order execution by
  - issuing an instruction ASAP
  - taking care of issuing/execution, including all hazard detections
    1. checking for structural hazards
    2. waiting for absence of data hazards
- Scoreboard was introduced in 1963 with CDC 6600
  - 7 units for integer operations
  - 5 memory reference units
  - 4 FP units

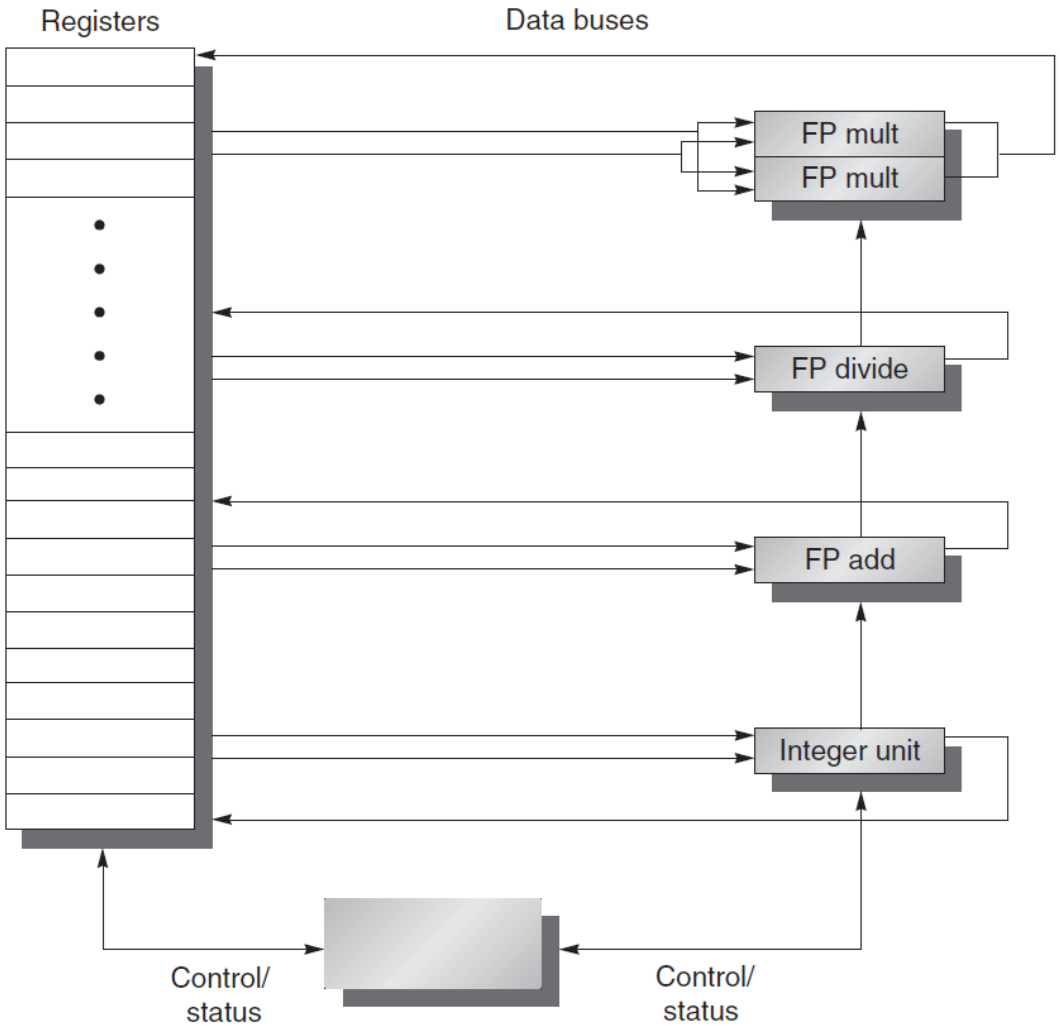
Assumption: Basic Architecture of MIPS Processor with Scoreboard

- Simpler structure than CDC 6600 original one
- For MIPS, focus on FP units (other units have small latencies)
- Assume
  - 1 integer unit
    - integer operations
    - memory references
    - branches
  - 1 FP adder (2 cycles)
  - 2 FP multipliers (10 cycles)
  - 1 FP divider (40 cycles)



Scoreboard

- All instructions go through Scoreboard, and Scoreboard record all the data dependencies.
- It can determine when an instruction can read its operands and begin execution.
  - If an instruction has to be stalled (RAW hazards), scoreboard can also find the earliest time when its operands are available.
- The scoreboard also controls when an instruction can write its result into the destination register.
  - To avoid WAW and WAR

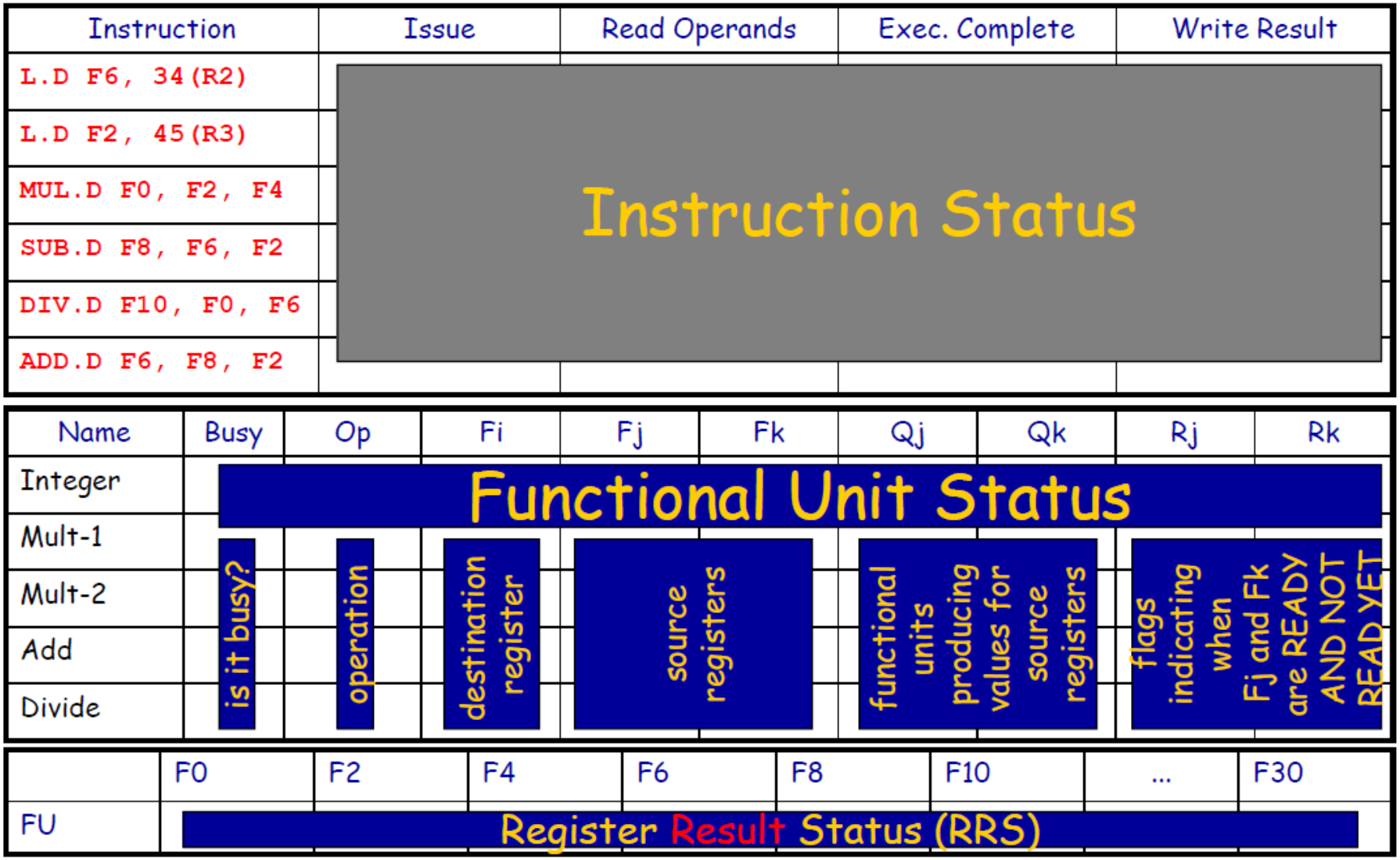


MIPS Pipeline Stages with Scoreboard

- All hazard detection and resolution is centralized in the *scoreboard module*
  - when operands can be read
  - when execution can start
  - when result can be written
- In-order issuing, out-of-order execution and commit
- The ID, EX, and WB stages are replaced by four new stages

1. Issue (ID-1)
   - in-order instruction decoding
   - check/stall for structural hazards
   - check/stall for WAW hazards
2. Read Operands (ID-2)
   - wait for each operand availability
   - resolve RAW hazards dynamically
   - no forwarding, but wait for WB
3. Execution (EX)
   - out-of-order execution
   - inform scoreboard upon completion
4. Write Result (WB)
   - check/stall for WAR hazards
   - write result into register asap
   - no statically-assigned write slot

MIPS Scoreboard Component



Scoreboard:Checking & Bookkeeping per Instruction(op D, S1, S2)

Instruction Status	Wait until	Bookkeeping
1. Issue	not Busy[FU] and not RRS[D]	Busy[FU] ← true; Op[FU] ←op; Fi[FU] ← D; Fj[FU] ←S1; Fk[FU] ←S2; Qj ← RRS[S1]; Qk ← RRS[S2]; Rj ← (Qj==0); Rk ← (Qk==0); RRS[D] ← FU
2. Read Operands	Rj=true and Rk=true	Rj ← false; Rk ← false; Qj ← 0; Qk ← 0
3. Execution Complete	FU is done executing	
4. Write Result	∀f { (Fj[f] ≠ Fi[FU] or Rj[f] = false) & (Fk[f] ≠ Fi[FU] or Rk[f] = false) }	∀f((Qj[f] = FU) ⇒ Rj[f] ← true) ∀f((Qk[f] = FU) ⇒ Rk[f] ← true) RSS[Fi[FU]] ← 0; Busy[FU] ← false

Scoreboard Example: Clock Cycle = 1

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				true
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- issue first instruction

Scoreboard Example: Clock Cycle = 2

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2		
L.D F2, 45(R3)				
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				false
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- read operand; second load instruction cannot be issued (struct. hazard)

Scoreboard Example: Clock Cycle = 3

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	
L.D F2, 45(R3)				
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				false
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- execute first load instruction; MUL.D waits too (issuing stage is stalled)

Scoreboard Example: Clock Cycle = 4

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- first load instruction commits (writing F6)

Scoreboard Example: Clock Cycle = 5

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				true
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- issue second load instruction

Scoreboard Example: Clock Cycle = 6

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				false
Mult-1	yes	MUL.D	F0	F2	F4	Integer		false	true
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1	Integer						

- read operands of second load instruction; issue multiply instruction

Scoreboard Example: Clock Cycle = 7

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				false
Mult-1	yes	MUL.D	F0	F2	F4	Integer		false	true
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2		Integer	true	false
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1	Integer			Add			

- execute second load instruction; stall multiply; issue subtraction

Scoreboard Example: Clock Cycle = 8

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4	Integer		true	true
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2		Integer	true	true
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- issue division; second load instruction is done; F2 ready (but not read yet)

Scoreboard Example: Clock Cycle = 9

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- reading operands of multiplication and subtraction; addition not issued

Scoreboard Example: Clock Cycle = 10

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- multiplication (10 cycles) and subtraction (2 cycles) are executing

Scoreboard Example: Clock Cycle = 11

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- subtraction execution is completed

Scoreboard Example: Clock Cycle = 12

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1					Divide		

- subtraction is done and result is written into F8

Scoreboard Example: Clock Cycle = 13

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			true	true
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition is issued because FP adder is now available

Scoreboard Example: Clock Cycle = 14

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition operands are read and execution starts (will take two cycles)

Scoreboard Example: Clock Cycle = 15

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- executing addition (1 more cycle) and multiplication (still 4 more to go)

Scoreboard Example: Clock Cycle = 16

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition execution is completed, but multiplication has still 3 more to go

Scoreboard Example: Clock Cycle = 17

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition stalls (result is not written) to avoid WAR hazard on F6

Scoreboard Example: Clock Cycle = 18

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- multiplication is completing, division waits for it, addition waits for division

Scoreboard Example: Clock Cycle = 19

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- multiplication execution is finally completed

Scoreboard Example: Clock Cycle = 20

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6			true	true

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

- multiplication has committed (writing F0)

Scoreboard Example: Clock Cycle = 21

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6			false	false

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

- reading division operands

Scoreboard Example: Clock Cycle = 22

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6			false	false

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

- No more WAR hazards, addition can write result into F6

Scoreboard Example: Clock Cycle = 61

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6			false	false

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

- After 40 cycles division execution is completed

Scoreboard Example: Clock Cycle = 62

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- Division commits and the whole program is done

Scoreboard Example: Clock Cycle = 62

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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- 62 cycles necessary (in-order issue; out-of-order execution & commit)

Scoreboard: Benefits & Costs

- The CDC 6600 designers measured
  - 1.7x performance improvement for FORTRAN programs
  - 2.5x performance improvement for assembly programs
    - but before breakthroughs in DRAM, SW scheduling, caches…
    - dynamic scheduling now popular with multiple-issue & speculation
- CDC 6600 scoreboard had about as much logic as one of the functional units
    - but four times as many buses as a simple in-order implementation
- In eliminating stalls, a scoreboard is limited by
    - # of scoreboard entries ≥ window size
    - amount of parallelism among the instructions
      - recall that CDC 6600 windows do not extend beyond a basic block
    - # and type of functional units (structural hazards)
    - # of buses ≥ # functional units proceeding in steps 2 & 4
    - presence of anti-dependences and output dependences