

# Reduction of Data Hazards Stalls with Dynamic Scheduling

i.e Current pipeline: In-order  
Single issue with FP support

- So far we have dealt with data hazards in instruction pipelines by:

- Result forwarding (register bypassing) to reduce or eliminate stalls needed to prevent RAW hazards as a result of true data dependence.
- Hazard detection hardware to stall the pipeline starting with the instruction that uses the result. *i.e forward + stall (if needed)*
- Compiler-based static pipeline scheduling to separate the dependent instructions minimizing actual hazard-prevention stalls in scheduled code.
  - Loop unrolling to increase basic block size: More ILP exposed.

i.e Start of instruction execution is not in program order

- Dynamic scheduling: (out-of-order execution)

- Uses a hardware-based mechanism to reorder or rearrange instruction execution order to reduce stalls dynamically at runtime.
  - Better dynamic exploitation of instruction-level parallelism (ILP).
- Enables handling some cases where instruction dependencies are unknown at compile time (ambiguous dependencies).
- Similar to the other pipeline optimizations above, a dynamically scheduled processor cannot remove true data dependencies, but tries to avoid or reduce stalling.

Why?

Fourth Edition: Appendix A.7, Chapter 2.4, 2.5  
(Third Edition: Appendix A.8, Chapter 3.2, 3.3)

CMPE550 - Shaaban

# Dynamic Pipeline Scheduling: *The Concept*

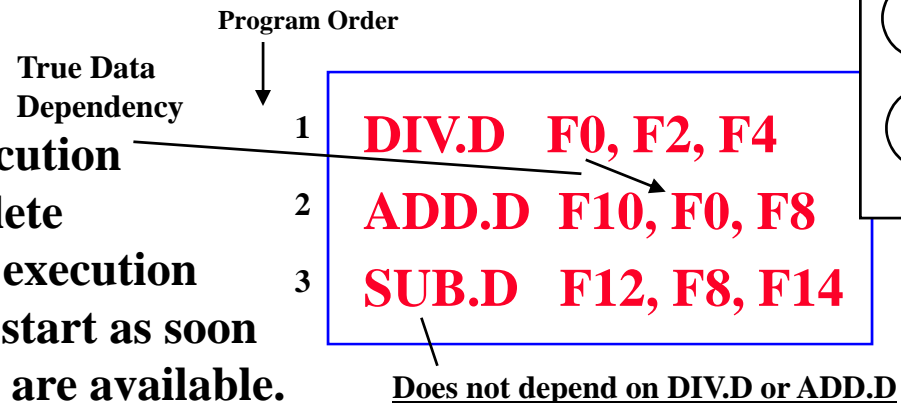
(Out-of-order execution)

i.e Start of instruction execution is not in program order

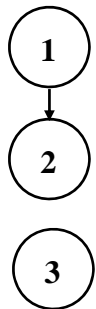
- **Dynamic pipeline scheduling overcomes the limitations of in-order pipelined execution by allowing out-of-order instruction execution.**
- **Instruction are allowed to start executing out-of-order as soon as their operands are available.**
  - **Better dynamic exploitation of instruction-level parallelism (ILP).**

## Example:

In the case of in-order pipelined execution SUB.D must wait for DIV.D to complete which stalled ADD.D before starting execution  
In out-of-order execution SUBD can start as soon as the values of its operands F8, F14 are available.



Dependency Graph



- **This implies allowing out-of-order instruction commit (completion).**
- **May lead to imprecise exceptions if an instruction issued earlier raises an exception.**
  - **This is similar to pipelines with multi-cycle floating point units.**

In Fourth Edition: Appendix A.7, Chapter 2.4  
(In Third Edition: Appendix A.8, Chapter 3.2)

Order = Program Instruction Order

**CMPE550 - Shaaban**

#2 lec # 4 Fall 2014 9-17-2014

# Dynamic Pipeline Scheduling

- Dynamic instruction scheduling is accomplished by:

- Dividing the Instruction Decode ID stage into two stages:

Always done in program order

1. **Issue:** Decode instructions, check for structural hazards.

- + – A record of data dependencies is constructed as instructions are issued
- This creates a dynamically-constructed dependency graph for the window of instructions in-flight (being processed) in the CPU.

Can be done out of program order

2. **Read operands:** Wait until data hazard conditions, if any, are resolved, then read operands when available (then start execution)

(All instructions pass through the issue stage in order but can be stalled or pass each other in the read operands stage).

- In the instruction fetch stage IF, fetch an additional instruction every cycle into a latch or several instructions into an instruction queue.
- Increase the number of functional units to meet the demands of the additional instructions in their EX stage.

- Two approaches to dynamic scheduling:

FYI

1

- Dynamic scheduling with the Scoreboard used first in CDC6600 (1963)

2

- The Tomasulo approach pioneered by the IBM 360/91 (1966)

(Control Data Corp.)

Fourth Edition: Appendix A.7, Chapter 2.4  
(Third Edition: Appendix A.8, Chapter 3.2)

CDC660 is the world's first  
"Supercomputer" Cost: \$7 million in 1963

**CMPE550 - Shaaban**

# Dynamic Scheduling With A Scoreboard

- The scoreboard is a centralized hardware mechanism that maintains an execution rate of one instruction per cycle by executing an instruction as soon as its operands are available in registers and no hazard conditions prevent it.

– e.g. Forming a single-issue out-of-order pipeline

EX Includes MEM

- It replaces ID, EX, WB with four stages: ID1, ID2, EX, WB

Includes MEM

Issue

Read Operands

No changes to Instruction Fetch (IF)

- Every instruction goes through the scoreboard where a record of data dependencies is constructed (corresponds to instruction issue).

In ID1 (Issue)

– In effect dynamically constructing the dependency graph by hardware for a window of instructions as they are issued one at a time in program order.

- A system with a scoreboard is assumed to have several functional units with their status information reported to the scoreboard.
- If the scoreboard determines that an instruction cannot execute immediately it executes another waiting instruction and keeps monitoring hardware units status and decide when the instruction can proceed to execute.
- The scoreboard also decides when an instruction can write its results to registers (hazard detection and resolution is centralized in the scoreboard).

Instruction Fetch (IF) is not changed

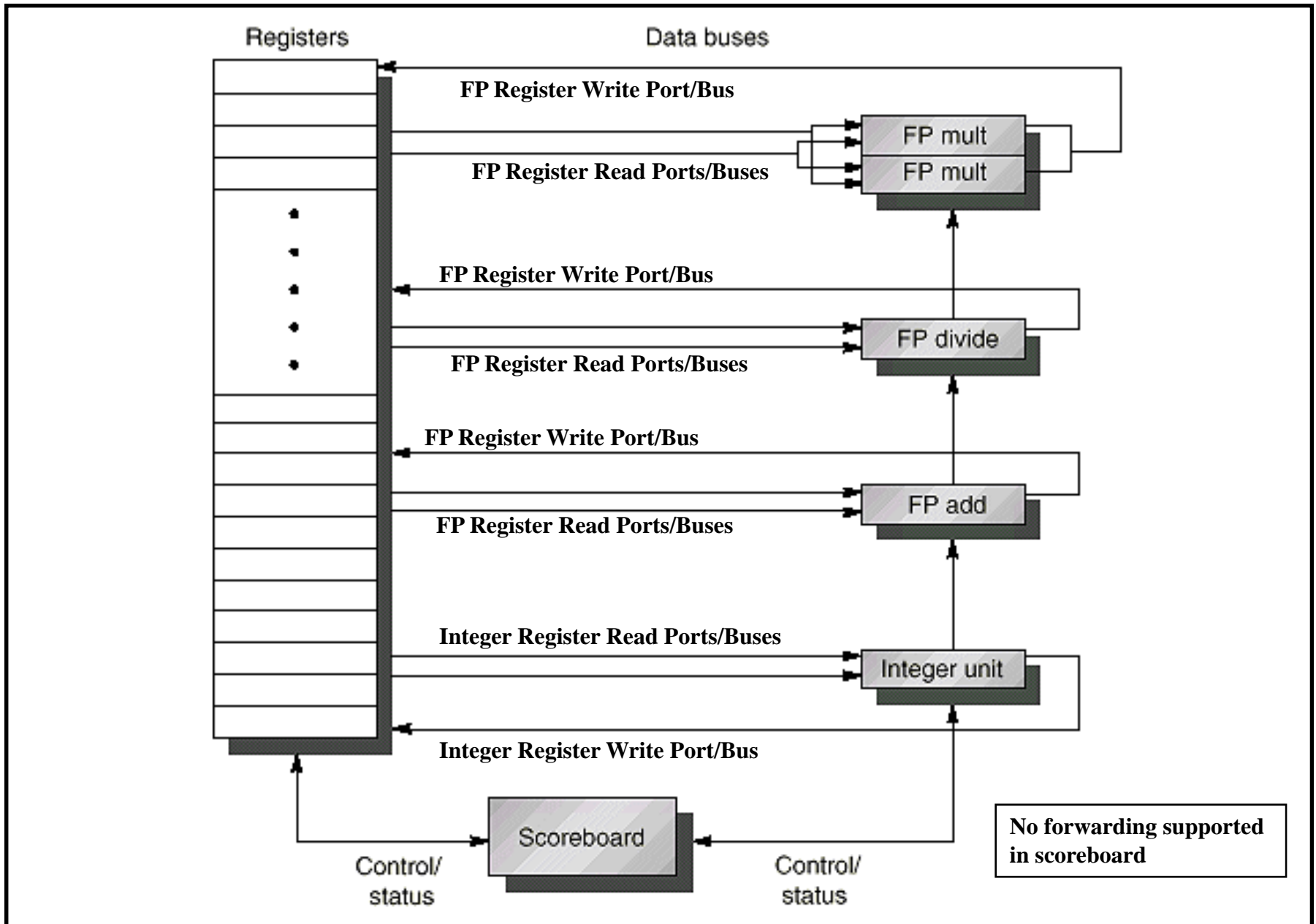
Order = Program Instruction Order

CMPE550 - Shaaban

In Fourth Edition: Appendix A.7  
(In Third Edition: Appendix A.8)

Introduced in CDC6600 (1963)

#4 lec # 4 Fall 2014 9-17-2014



The basic structure of a MIPS processor with a scoreboard

In Fourth Edition: Appendix A.7  
 (In Third Edition: Appendix A.8)

FP units are not pipelined similar to CDC6600

**CMPE550 - Shaaban**

# Instruction Execution Stages with A Scoreboard

1 Issue (ID1): An instruction is issued if:

Stage 0 Instruction Fetch (IF): No changes, in-order

Always done in program order

- A functional unit for the instruction is available (**No structural hazard**).
- The instruction result destination register is not marked for writing by an earlier active instruction (**No WAW hazard**, i.e no output dependence)
- If the above conditions are satisfied, the scoreboard issues the instruction to a functional unit and updates its internal data structures. As indicated by instruction issue requirements, **structural** and **WAW** hazards are resolved here by stalling the instruction issue. (this stage replaces part of **ID** stage in the conventional MIPS pipeline).

Can be done out of program order

2 Read operands (ID2): The scoreboard monitors the availability of the source operands. A source operand is available when no earlier active instruction will write it. When all source operands are available the scoreboard tells the functional unit to *read* all operands from the registers at once (no forwarding supported) and start execution (**RAW** hazards resolved here dynamically). This completes **ID**.

From registers (No forwarding)

3 Execution (EX): The functional unit starts execution upon receiving operands. When the results are ready it notifies the scoreboard (replaces **EX**, **MEM** in MIPS).

4 Write result (WB): Once the scoreboard senses that a functional unit completed execution, it checks for **WAR** hazards and stalls the completing instruction if needed otherwise the write back is completed. The functional unit issued to the instruction is marked as available (not busy) after WB is completed.

CMPE550 - Shaaban

In Fourth Edition: Appendix A.7  
(In Third Edition: Appendix A.8)

Stage 0: Fetch, no changes, in-order

# Three Parts of the Scoreboard

- 1 Instruction status:** Which of 4 steps the instruction is in.
- 2 Functional unit status:** Indicates the state of the functional unit (FU).

Nine fields for each functional unit:

Busy = Issued an instruction

- **Busy** Indicates whether the unit is busy or not
- **Op** Operation to perform in the unit (e.g., + or -)
- **Fi** Destination register
- **Fj, Fk** Source-register numbers i.e. Operand Registers
- **Qj, Qk** Functional units producing source (operand) registers Fj, Fk
- **Rj, Rk** Flags indicating when Fj, Fk are ready Yes or = 1 means ready  
 (set to Yes after operand is available to read  
both operands read at once from registers) i.e. when both Rj, Rk are set to yes  
 (both operands are ready)

- 3 Register result status:** Indicates which functional unit will write to each register, if one exists. Blank when no pending instructions will write that register.

Needed to check for possible WAW hazard and stall issue

e.g.

F0	F1	F2	F3	.....	F31
Add1	--	Mult1	--	.....	--

In Fourth Edition: Appendix A.7  
 (In Third Edition: Appendix A.8)

**CMPE550 - Shaaban**

# The Scoreboard: Detailed Pipeline Control

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

DAP Spr. '98 ©UCB 30

In Fourth Edition: Appendix A.7  
(In Third Edition: Appendix A.8)

**CMPE550 - Shaaban**

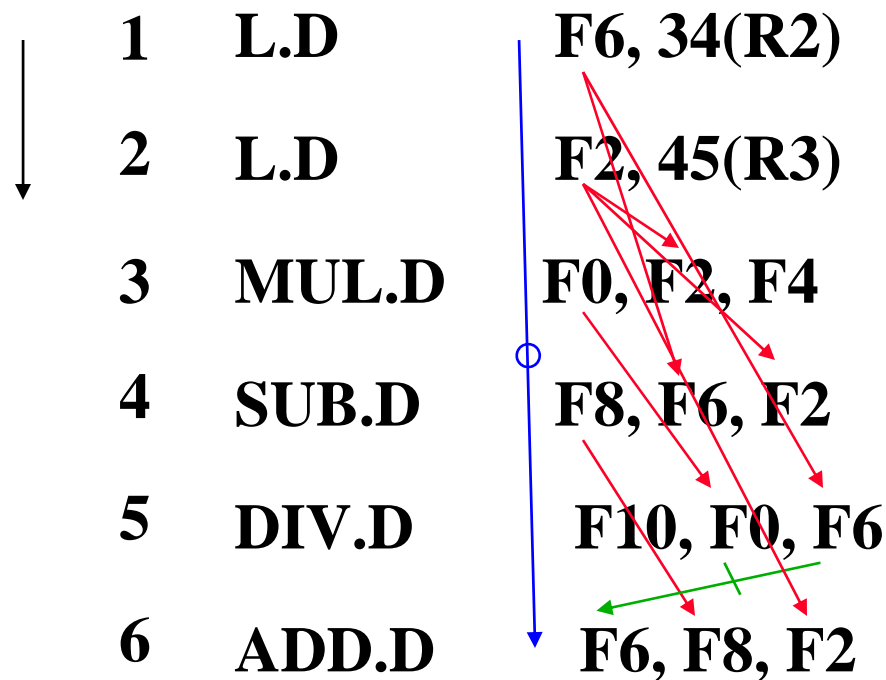
#8 lec # 4 Fall 2014 9-17-2014



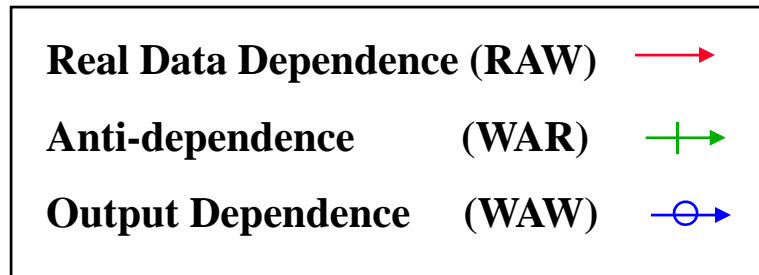
# A Scoreboard Example

The following code is run on the MIPS with a scoreboard given earlier with:

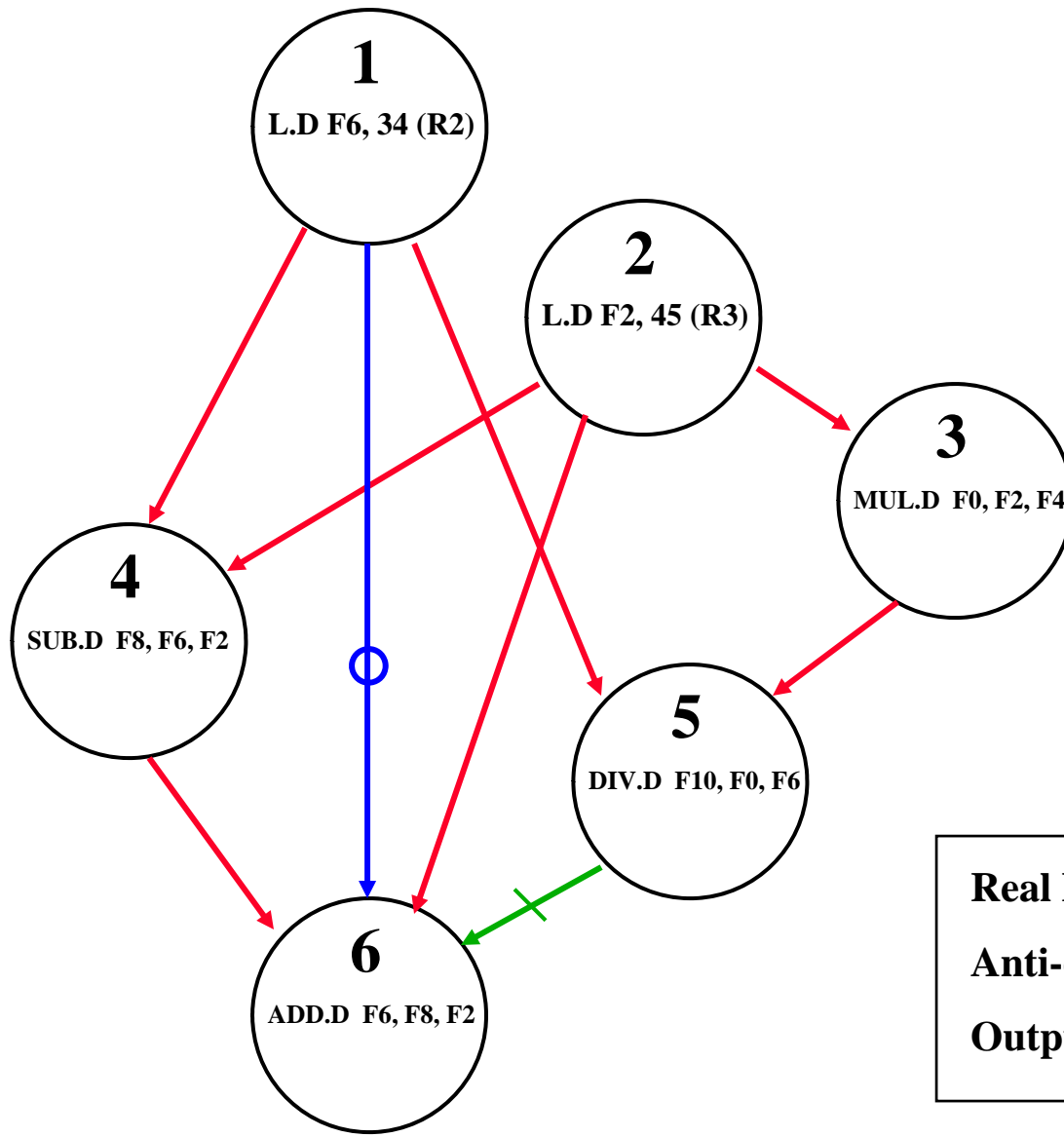
Functional Unit (FU)	# of FUs	EX cycles
Integer	1	1
Floating Point Multiply	2	10
Floating Point add/sub	1	2
Floating point Divide	1	40



All functional units  
are not pipelined  
(similar to CDC6600)



# Dependency Graph For Example Code



## Example Code

1	L.D	F6, 34(R2)
2	L.D	F2, 45(R3)
3	MUL.D	F0, F2, F4
4	SUB.D	F8, F6, F2
5	DIV.D	F10, F0, F6
6	ADD.D	F6, F8, F2

### Date Dependence:

(1, 4) (1, 5) (2, 3) (2, 4)  
(2, 6) (3, 5) (4, 6)

### Output Dependence:

(1, 6)

### Anti-dependence:

(5, 6)

Real Data Dependence (RAW)	→
Anti-dependence (WAR)	+→
Output Dependence (WAW)	⊖→

**CMPE550 - Shaaban**

# Scoreboard Example: Cycle 1

FP EX Cycles: Add = 2 cycles, Multiply = 10, Divide = 40

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

Functional unit status		Busy	Op	dest	S1	S2	FU for <i>j</i>	FU for <i>k</i>	<i>Fj?</i>	<i>Fk?</i>
Time	Name			<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				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock	FU				Integer					
1										

Means at end of Cycle 1

# Scoreboard Example: Cycle 2

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

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

Issue ? →

Functional unit status		Busy	Op	dest	S1	S2	FU for <i>j</i>	FU for <i>k</i>	F <sub><i>j</i></sub> ?	F <sub><i>k</i></sub> ?
Time	Name			<i>F<sub>i</sub></i>	<i>F<sub>j</sub></i>	<i>F<sub>k</sub></i>	<i>Q<sub>j</sub></i>	<i>Q<sub>k</sub></i>	<i>R<sub>j</sub></i>	<i>R<sub>k</sub></i>
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

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

? **Issue second L.D? No, stall on structural hazard. Single integer functional unit is busy.**

**CMPE550 - Shaaban**

# Scoreboard Example: Cycle 3

## Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read operands	Execution 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				

From Assumption  
**EX, MEM**  
 for L.D. in one cycle

Issue ? →

## Functional unit status

Time Name

Busy	Op	dest <i>Fi</i>	S1 <i>Fj</i>	S2 <i>Fk</i>	FU for <i>j</i> <i>Qj</i>	FU for <i>k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
Yes	Load	F6		R2				Yes
No								
No								
No								
No								

## Register result status

Clock

3

FU

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

- ? • **Issue MUL.D? No, cannot issue out of order (second L.D not issued yet)**

**CMPE550 - Shaaban**

# Scoreboard Example: Cycle 4

## Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read operands	Execution 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						

Issue ? →

## Functional unit status

Time Name

Actually free end of this cycle 4  
(available for instruction issue next cycle)

Busy	Op	dest <i>Fi</i>	S1 <i>Fj</i>	S2 <i>Fk</i>	FU for <i>j</i> <i>Qj</i>	FU for <i>k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
No								
No								
No								
No								
No								

## Register result status

Clock

4

FU

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

Issue second L.D?

# Scoreboard Example: Cycle 5

Issue →

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

<u>Functional unit status</u>		<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>Time</i>	<i>Name</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	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

<u>Register result status</u>		<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>
Clock	<i>FU</i>		Integer							
5										

**Issue second L.D**

# Scoreboard Example: Cycle 6

Issue →

Instruction status				Read	Execution	Write
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operands</i>	<i>complete</i>	<i>Result</i>
L.D	F6	34+	R2	1	2	3
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			

Functional unit status		<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</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>
	Integer	Yes	Load	F2		R3		
→	Mult1	Yes	Mult	F0	F2	F4	Integer	No
	Mult2	No						
	Add	No						
	Divide	No						

Register result status		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock										
6	<i>FU</i>	Mult1	Integer							

**Issue MUL.D**



# Scoreboard Example: Cycle 7

Issue →

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

## Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
→	Add
	Divide

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

- Issue SUB.D
- Read multiply operands?

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

Instruction status				Read	Execution	Write					
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operands</i>	<i>complete</i>	<i>Result</i>					
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	8							
ADD.D	F6	F8	F2								

Functional unit status		<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>Time</i>	<i>Name</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	F2		R3				Yes
	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		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock										
8	<i>FU</i>	Mult1	Integer			Add	Divide			

• Issue DIV.D

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

End of Cycle 8

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

Functional unit status		Busy	Op	dest	S1	S2	FU for <i>j</i>	FU for <i>k</i>	F <sub><i>j</i></sub> ?	F <sub><i>k</i></sub> ?
Time	Name			<i>F<sub>i</sub></i>	<i>F<sub>j</sub></i>	<i>F<sub>k</sub></i>	<i>Q<sub>j</sub></i>	<i>Q<sub>k</sub></i>	<i>R<sub>j</sub></i>	<i>R<sub>k</sub></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		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock	8									
	FU	Mult1				Add	Divide			

- Second L.D writes result to F2

# Scoreboard Example: Cycle 9

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

## Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read operands	Execution 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			?			

Issue ? →

## Functional unit status

Execution cycles Remaining (execution actually starts next cycle)

Time	Name
10	Mult1
2	Add
	Divide

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

- Read operands for MUL.D & SUB.D
- Issue ADD.D?

Ex starts next cycle for both instructions

CMPE550 - Shaaban

# Scoreboard Example: Cycle 11

## Instruction status

Instruction	<i>j</i>	<i>k</i>
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		

## Read Execution Write

Issue	operands	complete	Result
1	2	3	4
5	6	7	8
6	9		
7	9	11	
8			
?			

Issue ?

## Functional unit status

Execution cycles Remaining

Done executing

Time	Name
Integer	
8	Mult1
	Mult2
0	Add
	Divide

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>
------	----	-------------------	------------------------	------------------------	------------------------------	------------------------------	-------------------------	-------------------------

No								
Yes	Mult	F0	F2	F4			Yes	Yes
No								
Yes	Sub	F8	F6	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

## Register result status

Clock

11

FU

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

? • Issue ADD.D?

CMPE550 - Shaaban

# Scoreboard Example: Cycle 12

## Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Read operands	Execution 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	?		

Issue ?

## Functional unit status

Execution cycles  
Remaining

Actually free end  
of this cycle 12

Time	Name
	Integer
7	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest <i>F<sub>i</sub></i>	S1 <i>F<sub>j</sub></i>	S2 <i>F<sub>k</sub></i>	FU for <i>j</i> <i>Q<sub>j</sub></i>	FU for <i>k</i> <i>Q<sub>k</sub></i>	<i>F<sub>j</sub></i> ? <i>R<sub>j</sub></i>	<i>F<sub>k</sub></i> ? <i>R<sub>k</sub></i>
No								
Yes	Mult	F0	F2	F4			Yes	Yes
No								
No								
Yes	Div	F10	F0	F6	Mult1		No	Yes

## Register result status

Clock

12

FU

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

? • Read operands for DIV.D?

? • Issue ADD.D?

CMPE550 - Shaaban

# Scoreboard Example: Cycle 13

Instruction status				Read	Execution	Write	
Instruction	<i>j</i>	<i>k</i>		operands	complete	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			

Issue →

## Functional unit status

Time Name

Execution cycles Remaining

Integer  
6 Mult1  
Mult2  
→ Add  
Divide

Busy	Op	dest <i>F<sub>i</sub></i>	<i>S1</i> <i>F<sub>j</sub></i>	<i>S2</i> <i>F<sub>k</sub></i>	<i>FU for j</i> <i>Q<sub>j</sub></i>	<i>FU for k</i> <i>Q<sub>k</sub></i>	<i>F<sub>j</sub>?</i> <i>R<sub>j</sub></i>	<i>F<sub>k</sub>?</i> <i>R<sub>k</sub></i>
No								
Yes	Mult	F0	F2	F4			Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

## Register result status

Clock

13

FU

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

- Issue ADD.D, Add FP unit available at end of cycle 12 (start of 13)

CMPE550 - Shaaban

# Scoreboard Example: Cycle 17

<u>Instruction status</u>				<i>Read</i>	<i>Execution</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operands</i>	<i>complete</i>	<i>Result</i>	
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	<input type="checkbox"/>

Write result of ADD.D?  
No WAR hazard

<u>Functional unit status</u>		<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>Time</i>	<i>Name</i>			<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	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

<u>Register result status</u>		<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>
<i>Clock</i>	<i>FU</i>									
17		Mult1			Add		Divide			

- Write result of ADD.D? No, WAR hazard (DIV.D did not read any operands including F6)



# Scoreboard Example: Cycle 20

<u>Instruction status</u>				<i>Read</i>	<i>Execution</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operands</i>	<i>complete</i>	<i>Result</i>	
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	?

## Functional unit status

<i>Time</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								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

## Register result status

<i>Clock</i>	<i>FU</i>	<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>
20					Add		Divide			

- ? • Read operands for DIV.D?
- ? • Write result of ADD.D?

# Scoreboard Example: Cycle 21

Instruction status				Read	Execution	Write	
Instruction	<i>j</i>	<i>k</i>	Issue	operands	complete	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	?

## Functional unit status

Execution cycles Remaining (execution actually starts next cycle)

Time Name  
Integer  
Mult1  
Mult2  
Add  
40 Divide

Busy	Op	dest <i>Fi</i>	S1 <i>Fj</i>	S2 <i>Fk</i>	FU for <i>j</i> <i>Qj</i>	FU for <i>k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
No								
No								
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6			Yes	Yes

## Register result status

Clock  
21

FU

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

- DIV.D reads operands, starts execution next cycle
- ? • Write result of ADD.D?

# Scoreboard Example: Cycle 22

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

<u>Functional unit status</u>		<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>Time</i>	<i>Name</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								
39	Divide	Yes	Div	F10	F0	F6			Yes	Yes

<u>Register result status</u>		<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>
Clock	22									
<i>FU</i>							Divide			

First cycle DIV.D execution (39 more ex cycles)

ADD.D writes result in F6 (No WAR, DIV.D read operands in cycle 21)  
(Possible)

CMPE550 - Shaaban

# Scoreboard Example: Cycle 61

<u>Instruction status</u>				<i>Read</i>	<i>Execution</i>	<i>Write</i>	
Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>operands</i>	<i>complete</i>	<i>Result</i>	
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	<b>61</b>	
ADD.D	F6	F8	F2	13	14	16	22

<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</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>
	Integer	No						
	Mult1	No						
	Mult2	No						
	Add	No						
	0 Divide	Yes	Div	F10	F0	F6		
							Yes	Yes

Done executing

<u>Register result status</u>		<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>
<b>61</b>	<i>FU</i>						Divide			

• DIV.D done executing

# Scoreboard Example: Cycle 62

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

**Instruction Block done**

## Functional unit status

*Time Name*

<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>
No								
No								
No								
No								
No								

## Register result status

**Clock**

**62**

*FU*

<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>

- **We have:**
  - **In-order issue,**
  - **Out-of-order execution, completion**

# Dynamic Scheduling: The Tomasulo Algorithm

- Developed at IBM and first implemented in IBM's 360/91 mainframe in 1966, about 3 years after the debut of the scoreboard in the CDC 6600.
- Dynamically schedule the pipeline in hardware to reduce stalls.
- Differences between IBM 360 & CDC 6600 ISA.
  - IBM has only 2 register specifiers/instr vs. 3 in CDC 6600.
  - IBM has 4 FP registers vs. 8 in CDC 6600 (part of ISA).
- Current CPU architectures that can be considered descendants of the IBM 360/91 which implement and utilize a variation of the Tomasulo Algorithm include:

**RISC CPUs: Alpha 21264, HP 8600, MIPS R12000, PowerPC G4 ..**

**RISC-core x86 CPUs: AMD Athlon, Intel Pentium III, 4, Xeon, ....**

**In Fourth Edition: Chapter 2.4 (In Third Edition: Chapter 3.2)**

**CMPE550 - Shaaban**

# Tomasulo Algorithm Vs. Scoreboard

- Control & buffers *distributed* with Functional Units (FUs) Vs. centralized in Scoreboard:
  - FU buffers are called “<sup>RS</sup>reservation stations” which have pending instructions and operands and other instruction status info (including data dependencies).
  - Reservations stations are sometimes referred to as “physical registers” or “renaming registers” as opposed to architecture registers specified by the ISA. — i.e. Operands
- ISA Registers in instructions are replaced by either values (if available) or pointers (renamed) to reservation stations (RS) that will supply the value later:
  - This process is called register renaming. Done in issue stage (in-order)
    - Register renaming eliminates WAR, WAW hazards (name dependence).
  - Allows for a *hardware-based* version of loop unrolling.
  - More reservation stations than ISA registers are possible, leading to optimizations that compilers can't achieve and prevents the number of ISA registers from becoming a bottleneck.
- Instruction results go (forwarded) from RSs to RSs , *not through registers*, over *Common Data Bus (CDB)* that broadcasts results to all waiting RSs (dependant instructions).
- Loads and Stores are treated as FUs with RSs as well.

1

Register Renaming

2

Data Forwarding

In Fourth Edition: Chapter 2.4 (In Third Edition: Chapter 3.2)

CMPE550 - Shaaban

Control Data Corp.

# IBM 360/91 Vs. CDC 6600

Tomasulo-based (1966)

Scoreboard-based (1963)

**Pipelined Functional Units**

**Multiple Functional Units**

**(Not pipelined)**

**(6 load, 3 store, 3 +, 2 x/÷)**

**(1 load/store, 1 + , 2 x, 1 ÷)**

**window size:  $\leq 14$  instructions**

**$\leq 5$  instructions**

**No issue on structural hazard**

**same**

**WAW: renaming avoids it**

Eliminated  
By register  
renaming

**stall issue** ID1

**WAR: renaming avoids it**

**stall completion** WB

**Broadcast results from FU** Over CDB

**Write/read registers**

**(Implements forwarding)**

**(Forwarding *not* supported)**

**Control: reservation stations  
distributed**

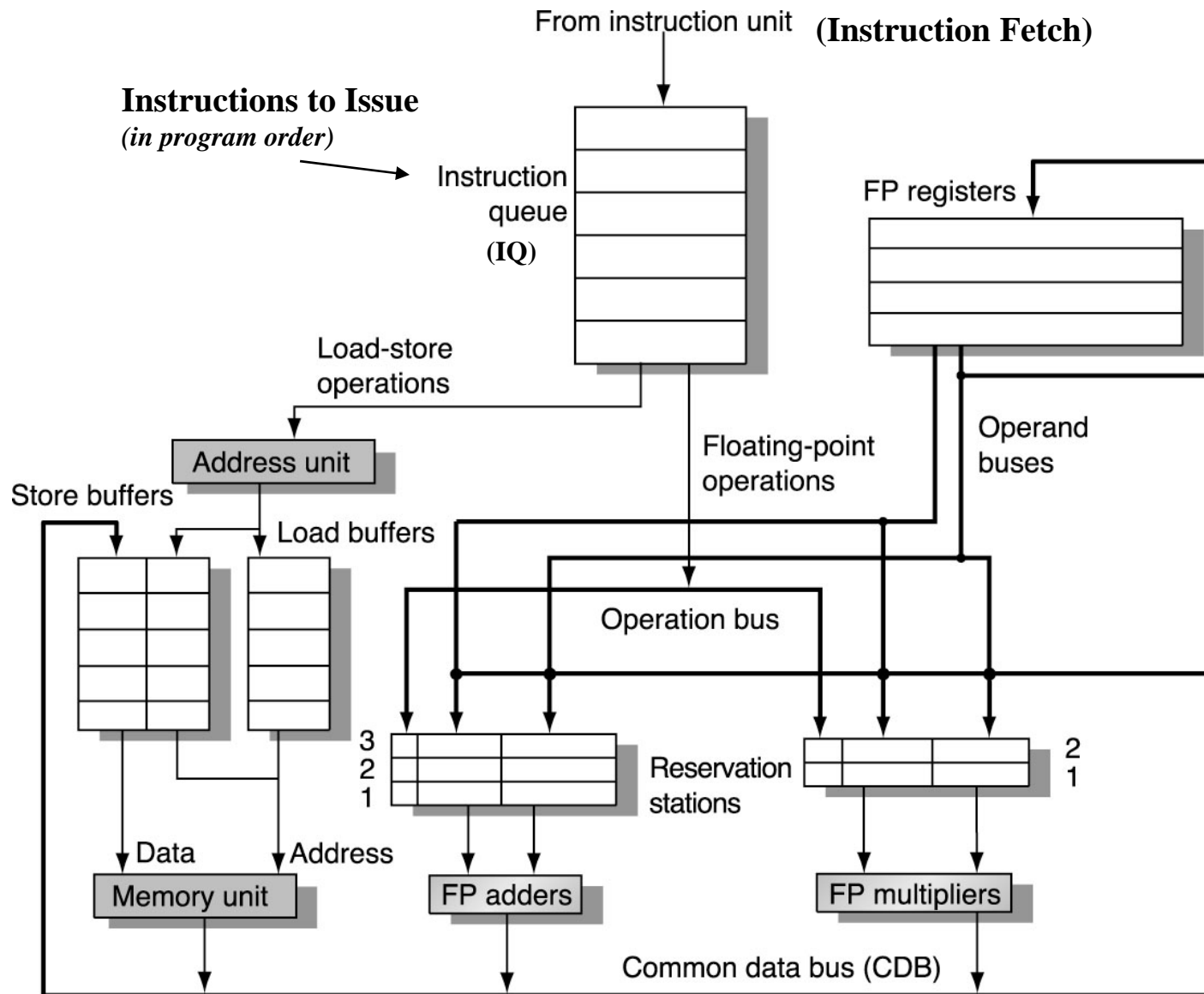
**central scoreboard**

In Fourth Edition: Chapter 2.4 (In Third Edition: Chapter 3.2)

**CMPE550 - Shaaban**



# Dynamic Scheduling: The Tomasulo Approach



The basic structure of a MIPS floating-point unit using Tomasulo's algorithm

In Fourth Edition: Chapter 2.4  
(In Third Edition: Chapter 3.2)

Pipelined FP units are used here

**CMPE550 - Shaaban**

# Reservation Station (RS) Fields

- **Op** Operation to perform in the unit (e.g., + or –)
- **Vj, Vk** **Value** of Source operands S1 and S2 When available
  - Store buffers have a single **V** field indicating result to be stored.
- **Qj, Qk** Reservation stations producing source registers. RS's (i.e. operand values needed by instruction) (value to be written).
  - No ready flags as in Scoreboard;  $Q_j, Q_k = 0 \Rightarrow$  ready.
  - Store buffers only have **Qi** for RS producing result. to be stored
- **A:** Address information for loads or stores. Initially immediate field of instruction then effective address when calculated.
- **Busy:** Indicates reservation station is busy.
- **Register result status:** **Qi** Indicates which Reservation Station will write each register, if one exists.
  - Blank (or 0) when no pending instruction (i.e. RS) exist that will write to that register.

# Three Stages of Tomasulo Algorithm

## 1 Issue: Get instruction from pending Instruction Queue (IQ).

Always done in program order

- Instruction issued to a free reservation station (RS) (no structural hazard).
- Selected RS is marked busy.
- Control sends available instruction operands values (from ISA registers) to assigned RS.
- Operands not available yet are renamed to RSs that will produce the operand (register renaming). (Dynamic construction of data dependency graph)

Stage 0 Instruction Fetch (IF): No changes, in-order

## 2 Execution (EX): Operate on operands.

Also includes waiting for operands + MEM

- When both operands are ready then start executing on assigned FU.
- If all operands are not ready, watch Common Data Bus (CDB) for needed result (forwarding done via CDB). (i.e. wait on any remaining operands, no RAW)

## 3 Write result (WB): Finish execution.

Data dependencies observed

And also to destination register

- Write result on Common Data Bus (CDB) to all awaiting units (RSs)
- Mark reservation station as available.

i.e broadcast result on CDB (forwarding)

- Normal data bus: data + destination (“go to” bus).

Note: No WB for stores or branches

## Common Data Bus (CDB): data + **source** (“**come from**” bus):

- 64 bits for data + 4 bits for Functional Unit **source** address.
- Write data to waiting RS if source matches expected RS (that produces result).
- Does the result forwarding via broadcast to waiting RSs.

Can be done out of program order

In Fourth Edition: Chapter 2.4  
(In Third Edition: Chapter 3.2)

Including destination register

**CMPE550 - Shaaban**

# Steps in The Tomsulo Approach and The Requirements of Each Step

Instruction status	Wait until	Action or bookkeeping
Issue	Station or buffer empty	<pre> if (Register['S1'].Qi ≠ 0)     {RS[r].Qj ← Register['S1'].Qi} else {RS[r].Vj ← S1; RS[r].Qj ← 0}; if (Register[S2].Qi ≠ 0)     {RS[r].Qk ← Register[S2].Qi}; else {RS[r].Vk ← S2; RS[r].Qk ← 0} RS[r].Busy ← yes; Register['D'].Qi = r; </pre>
Execute	(RS[r].Qj=0) and (RS[r].Qk=0)	None—operands are in Vj and Vk
Write result	Execution completed at r and CDB available	<pre> ∀x(if (Register[x].Qi=r) {Fx ← result; Register[x].Qi ← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj ← result; RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk ← result; RS[x].Qk ← 0}); ∀x(if (Store[x].Qi=r) {Store[x].V ← result; Store[x].Qi ← 0}); RS[r].Busy ← No </pre>

# Drawbacks of The Tomasulo Approach

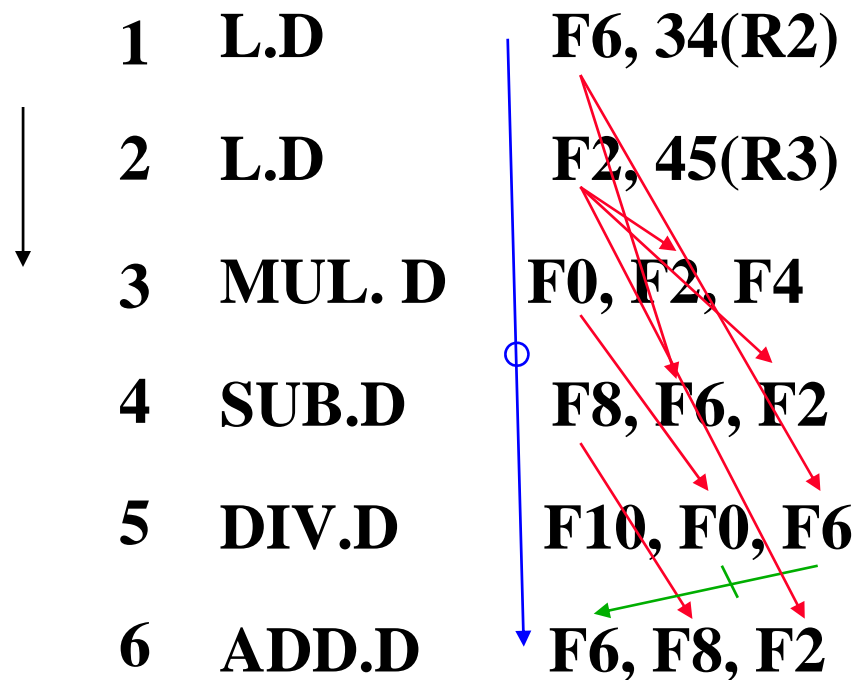
- **Implementation Complexity:**
  - **Example:** The implementation of the Tomasulo algorithm may have caused delays in the introduction of 360/91, MIPS 10000, IBM 620 among other CPUs.
- **Many high-speed associative result stores using (CDB) are required.**
- **Performance limited by one Common Data Bus**
  - **Possible solution:**  
**Multiple CDBs → more Functional Unit and RS logic needed for parallel associative stores. (Even more complexity)**

# Tomasulo Approach Example

Using the same code used in the scoreboard example to be run on the Tomasulo configuration given earlier:

RS = Reservation Stations

	# of RSs	EX Cycles
Integer	3	1
Floating Point Multiply/divide	2	10/40
Floating Point add/sub	3	2



Pipelined Functional Units

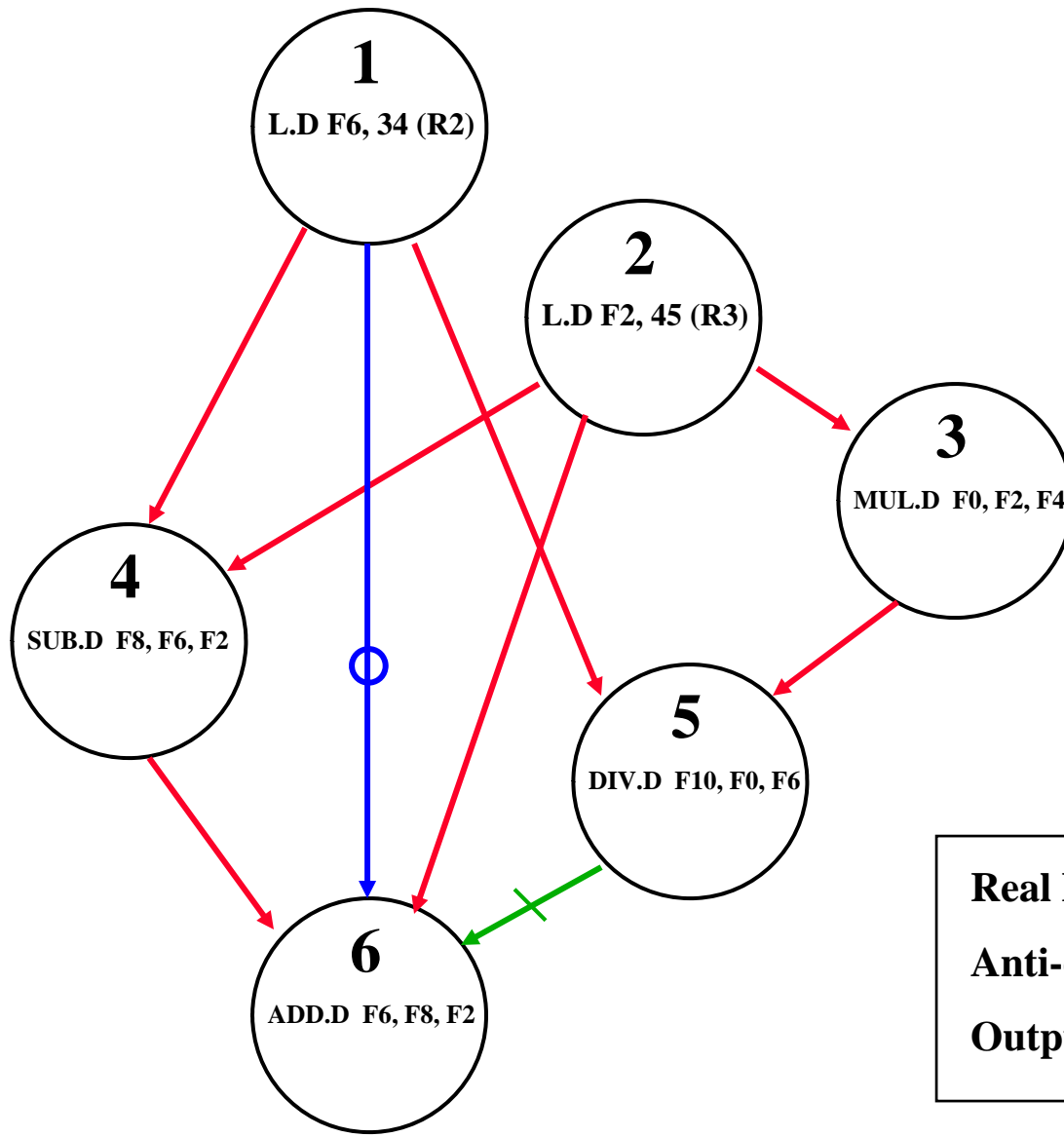
Real Data Dependence (RAW) →

Anti-dependence (WAR) +→

Output Dependence (WAW) ⊖→

L.D processing takes two cycles: EX, MEM  
(only one cycle in scoreboard example)

# Dependency Graph For Example Code



## Example Code

1	L.D	F6, 34(R2)
2	L.D	F2, 45(R3)
3	MUL.D	F0, F2, F4
4	SUB.D	F8, F6, F2
5	DIV.D	F10, F0, F6
6	ADD.D	F6, F8, F2

### Date Dependence:

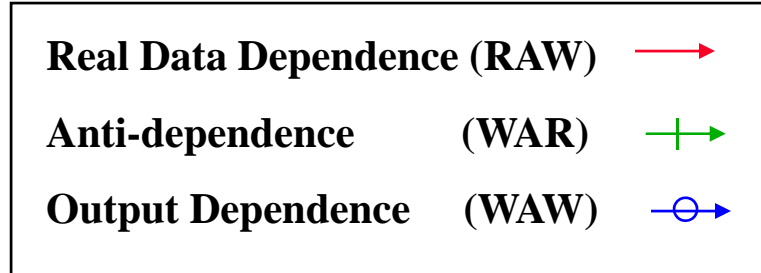
(1, 4) (1, 5) (2, 3) (2, 4)  
(2, 6) (3, 5) (4, 6)

### Output Dependence:

(1, 6)

### Anti-dependence:

(5, 6)



The same code used is the scoreboard example

**CMPE550 - Shaaban**

# Tomasulo Example: Cycle 0

(i.e at end of cycle 0)

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

## Instruction status

Instruction	<i>j</i>	<i>k</i>	Issue	Execution complete	Write Result	Busy	Address
L.D F6 34+	R2					Load1	No
L.D F2 45+	R3					Load2	No
MUL.D F0 F2	F4					Load3	No
SUB.D F8 F6	F2						
DIV.D F10 F0	F6						
ADD.D F6 F8	F2						

## Reservation Stations

Time	Name	Busy	Op	S1 V <sub>j</sub>	S2 V <sub>k</sub>	RS for <i>j</i> Q <sub>j</sub>	RS for <i>k</i> Q <sub>k</sub>
0	Add1	No					
0	Add2	No					
0	Add3	No					
0	Mult1	No					
0	Mult2	No					

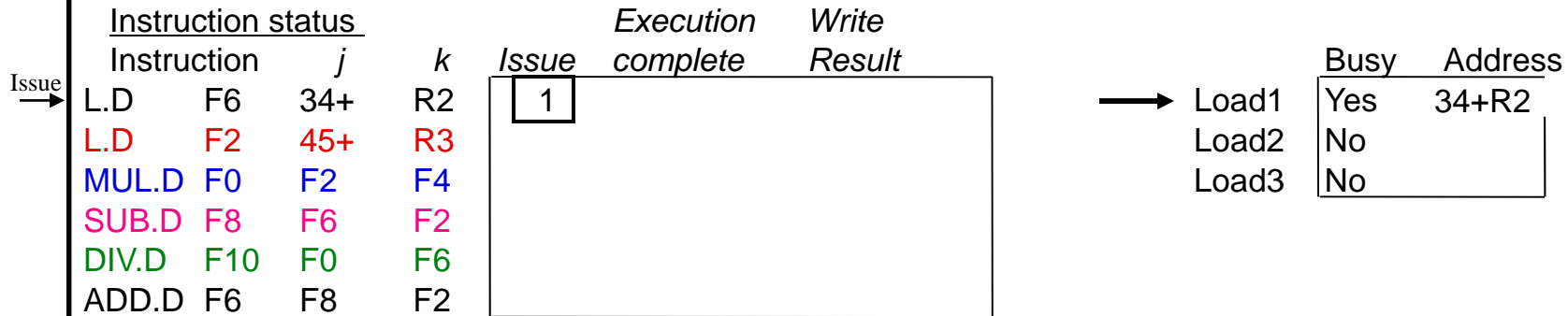
## Register result status

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



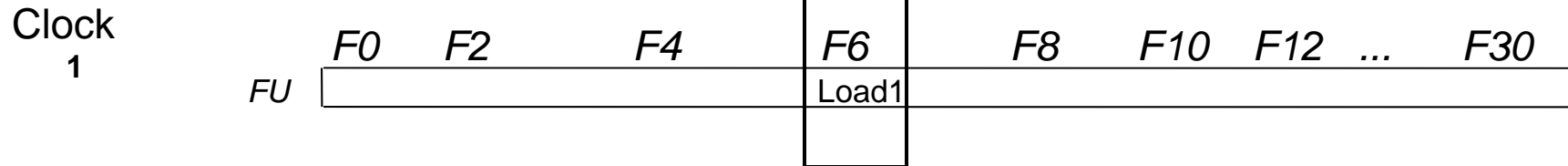
# Tomasulo Example Cycle 1

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40



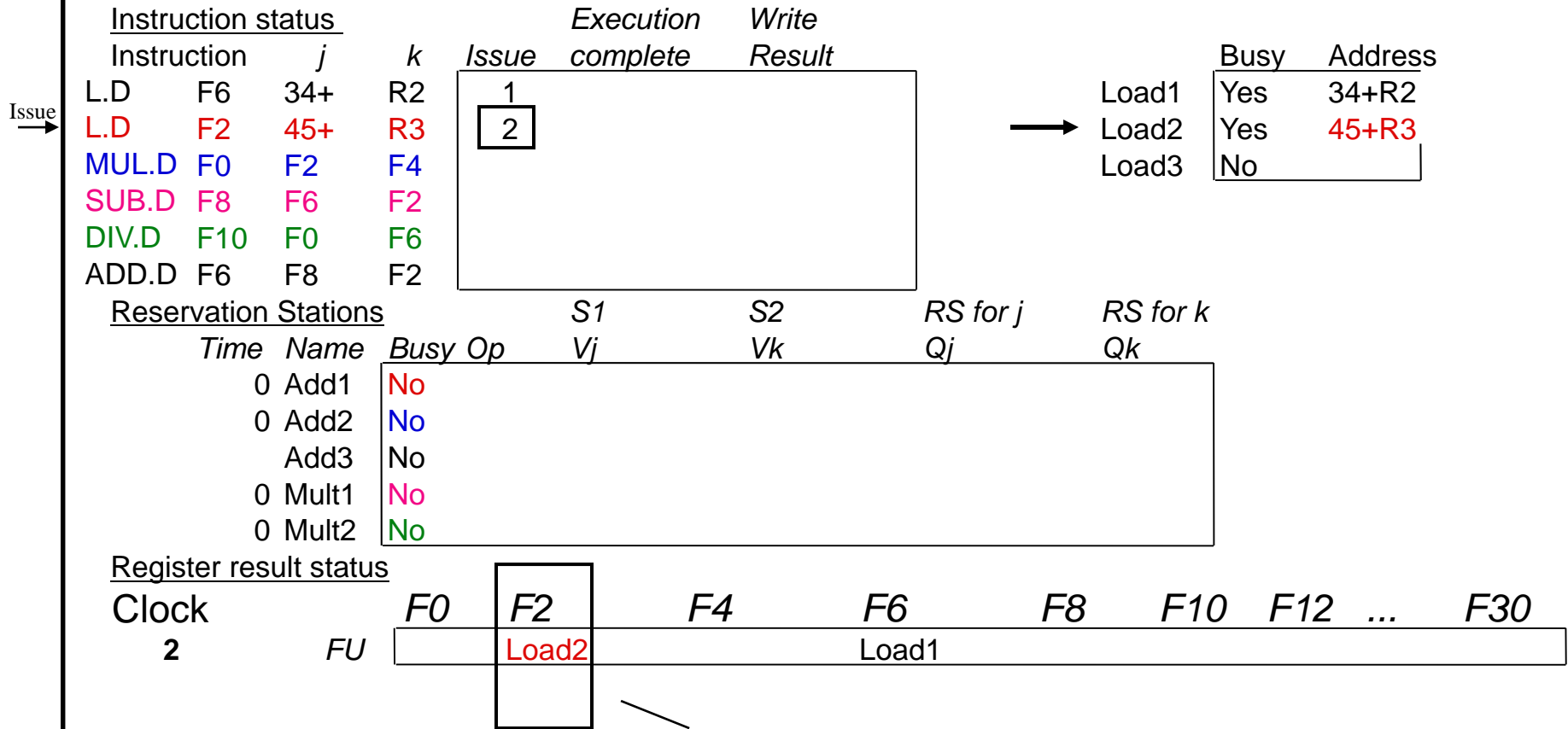
Reservation Stations			S1	S2	RS for j	RS for k
Time	Name	Busy Op	Vj	Vk	Qj	Qk
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
0	Mult2	No				

Register result status



- Issue first load to load1 reservation station

# Tomasulo Example: Cycle 2



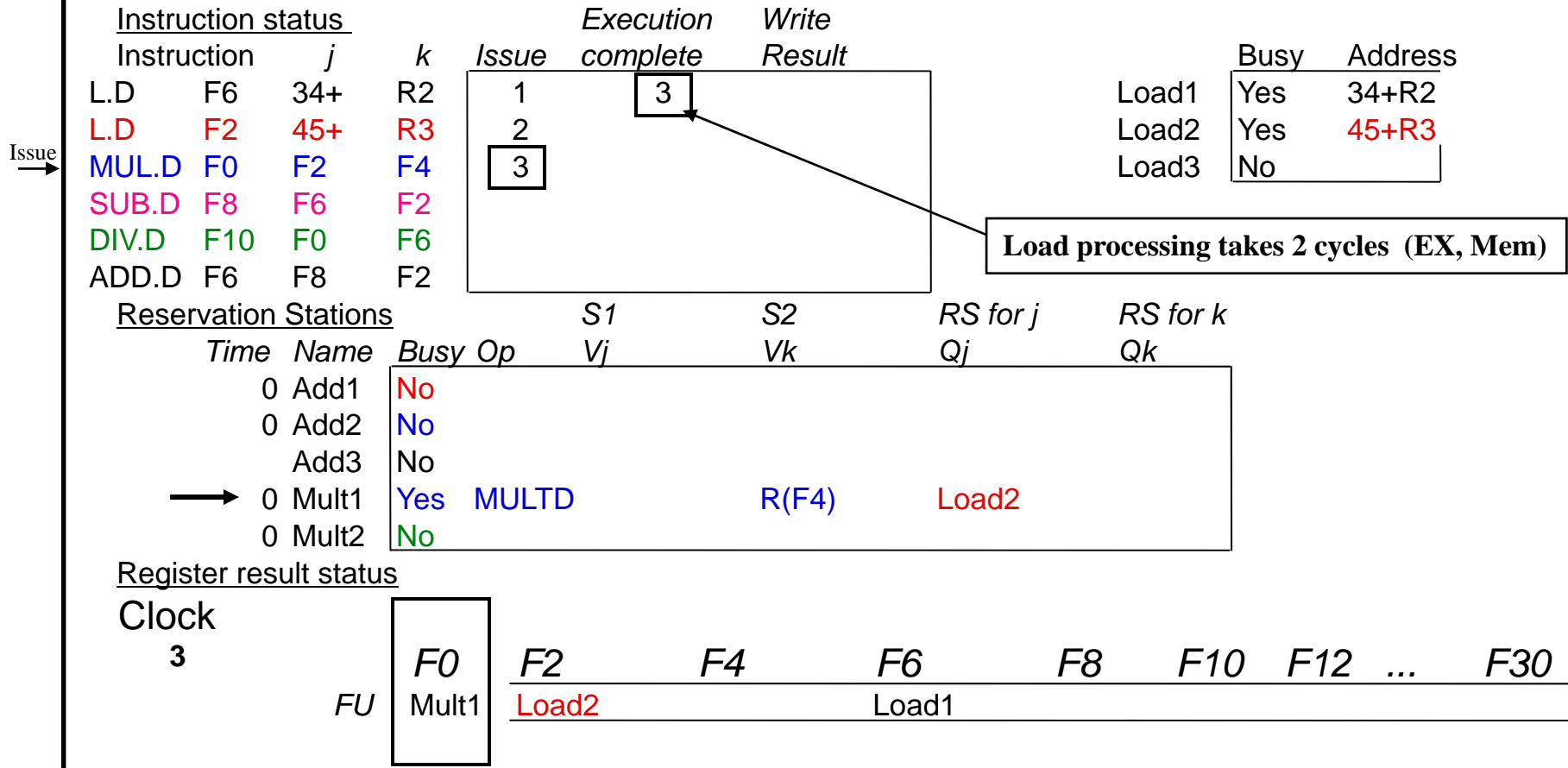
- Issue second load to load2 reservation station

Note: Unlike 6600, can have multiple loads outstanding

(CDC6600 only has one integer FU)

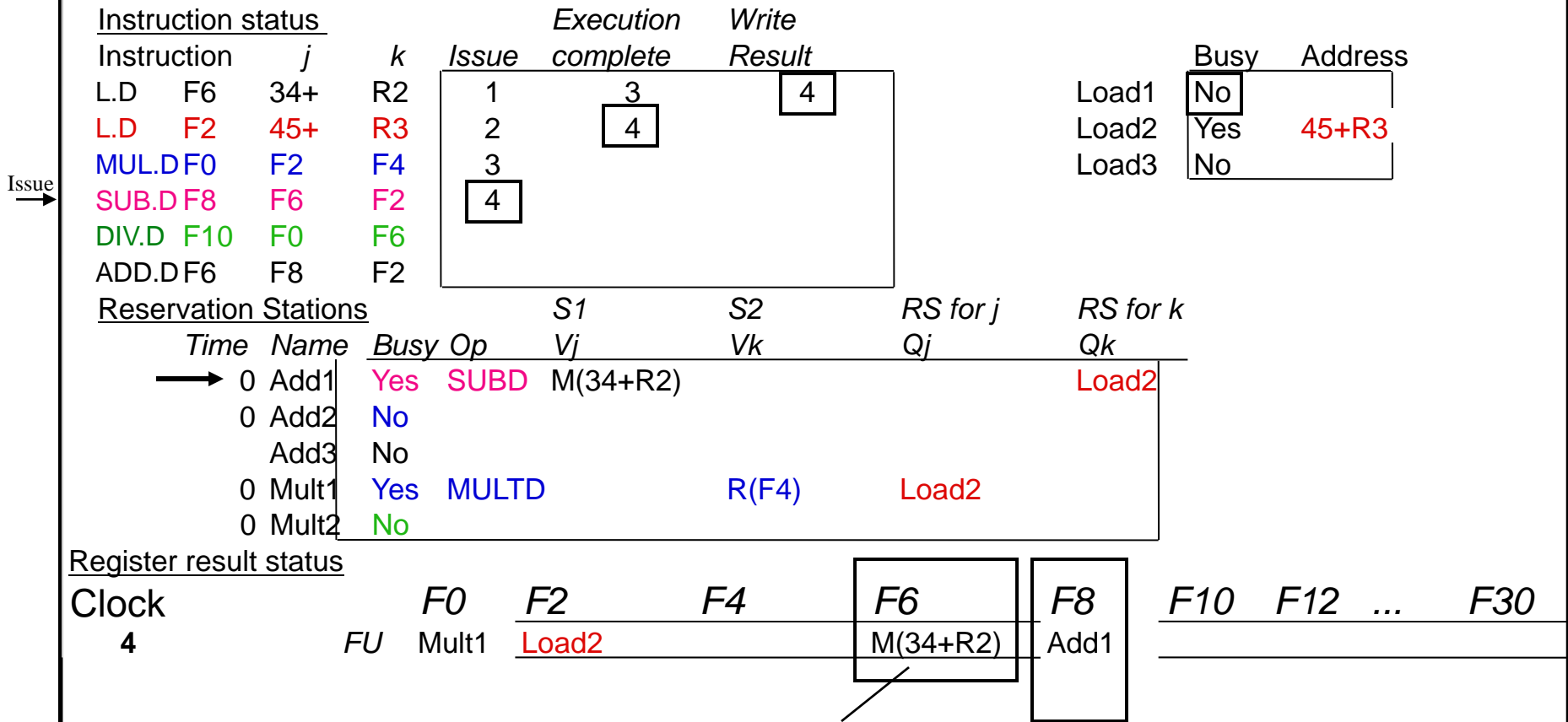
**CMPE550 - Shaaban**

# Tomasulo Example: Cycle 3



• Issue MUL.D to reservation station Mult1

# Tomasulo Example: Cycle 4



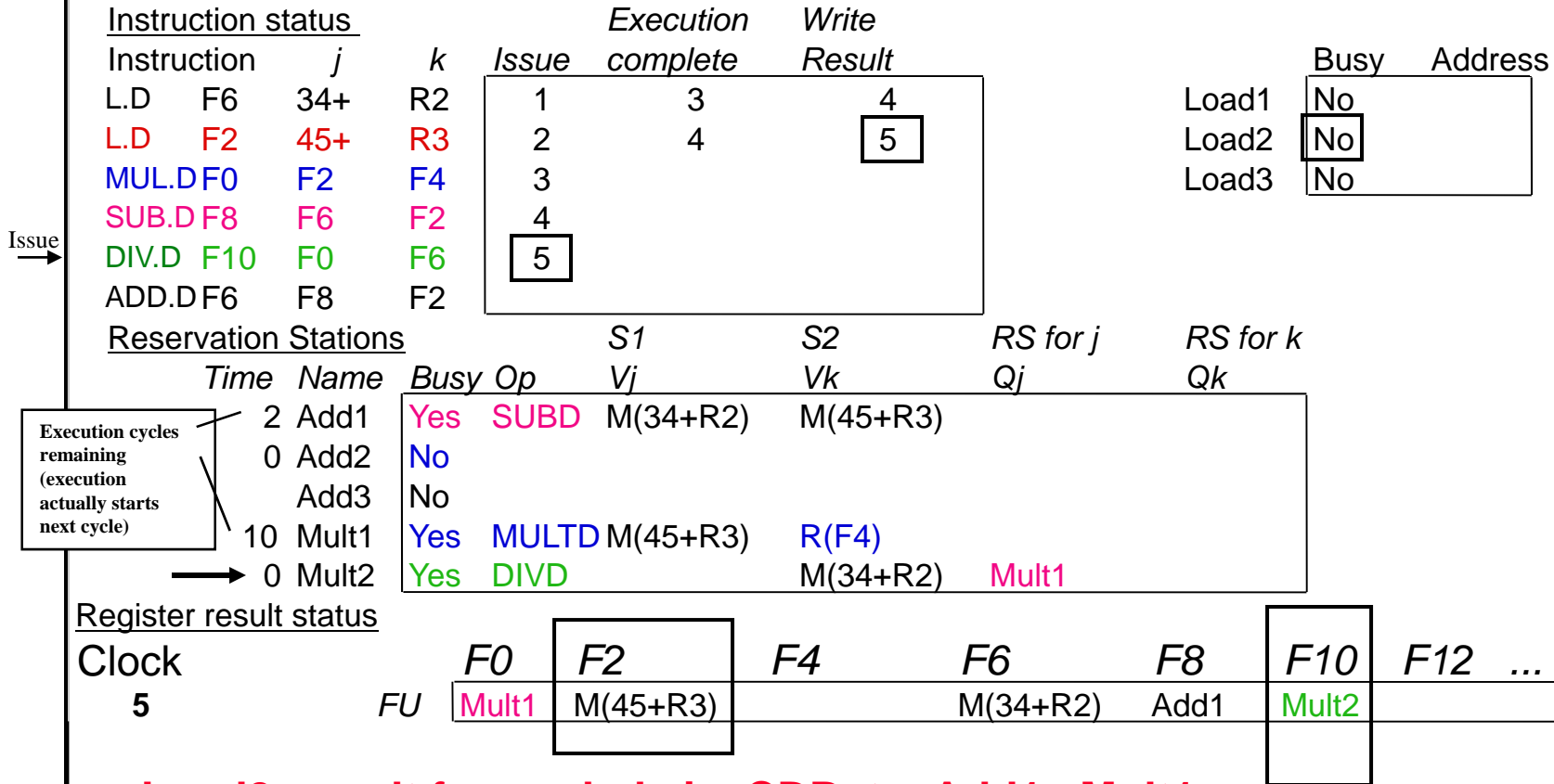
• Issue SUB.D

i.e. register F6 has the loaded value from memory

• Load2 completing; what is waiting for it?

# Tomasulo Example: Cycle 5

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40



- Load2 result forwarded via CDB to Add1, Mult1 (SUB.D, MUL.D execution will start execution next cycle 6)
- Issue DIV.D to Mult2 reservation station

# Tomasulo Example Cycle 6

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

Instruction status				Execution	Write		
Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result	Busy	Address
L.D F6	34+	R2	1	3	4	Load1	No
L.D F2	45+	R3	2	4	5	Load2	No
MUL.DF0	F2	F4	3			Load3	No
SUB.DF8	F6	F2	4				
DIV.D F10	F0	F6	5				
ADD.DF6	F8	F2	6				

Issue →

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Busy Op	V <sub>j</sub>	V <sub>k</sub>	Q <sub>j</sub>	Q <sub>k</sub>
1	Add1	Yes SUBD	M(34+R2)	M(45+R3)		
0	Add2	Yes ADDD		M(45+R3)	Add1	
	Add3	No				
9	Mult1	Yes MULTD	M(45+R3)	R(F4)		
0	Mult2	Yes DIVD		M(34+R2)	Mult1	

Execution cycles remaining

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
6		Mult1	M(45+R3)		Add2	Add1	Mult2		

- **ADD.D is issued here vs. scoreboard (in cycle 16)**

# Tomasulo Example: Cycle 7

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

Instruction status				Execution	Write	Busy	Address	
Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result			
L.D	F6	34+	R2	1	3	4	Load1	No
L.D	F2	45+	R3	2	4	5	Load2	No
MUL.D	F0	F2	F4	3			Load3	No
SUB.D	F8	F6	F2	4	7			
DIV.D	F10	F0	F6	5				
ADD.D	F6	F8	F2	6				

Reservation Stations				S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Busy	Op	V <sub>j</sub>	V <sub>k</sub>	Q <sub>j</sub>	Q <sub>k</sub>
0	Add1	Yes	SUBD	M(34+R2)	M(45+R3)		
0	Add2	Yes	ADDD		M(45+R3)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Done executing

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(45+R3)		Add2	Add1	Mult2		

- RS Add1 completing; what is waiting for it?

# Tomasulo Example: Cycle 10

Instruction status				Execution	Write		Busy	Address
Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result			
L.D F6	34+	R2	1	3	4		Load1	No
L.D F2	45+	R3	2	4	5		Load2	No
MUL.D F0	F2	F4	3				Load3	No
SUB.D F8	F6	F2	4	7	8			
DIV.D F10	F0	F6	5					
ADD.D F6	F8	F2	6	10				

## Reservation Stations

Time	Name	Busy	Op	S1 V <sub>j</sub>	S2 V <sub>k</sub>	RS for <i>j</i> Q <sub>j</sub>	RS for <i>k</i> Q <sub>k</sub>
0	Add1	No					
0	Add2	Yes	ADD	M()-M()	M(45+R3)		
0	Add3	No					
5	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIV		M(34+R2)	Mult1	

Done executing

## Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2			

- RS Add2 completed execution



# Tomasulo Example: Cycle 11

Instruction status				Issue	Execution complete	Write Result	Busy	Address
Instruction	<i>j</i>	<i>k</i>						
L.D	F6	34+	R2	1	3	4	Load1	No
L.D	F2	45+	R3	2	4	5	Load2	No
MUL.D	F0	F2	F4	3			Load3	No
SUB.D	F8	F6	F2	4	7	8		
DIV.D	F10	F0	F6	5				
ADD.D	F6	F8	F2	6	10	11		

Reservation Stations			<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>
Time	Name	Busy Op	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
0	Add1	No				
0	Add2	No				
0	Add3	No				
4	Mult1	Yes	MULTD	M(45+R3)	R(F4)	
0	Mult2	Yes	DIVD		M(34+R2)	Mult1

## Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

- Write back result of ADD.D in this cycle  
(What about anti-dependence over F6 with DIV.D ?)

# Tomasulo Example: Cycle 15

Instruction status				Execution	Write		Busy	Address
Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result			
L.D	F6	34+	R2	1	3	4	Load1	No
L.D	F2	45+	R3	2	4	5	Load2	No
MUL.D	F0	F2	F4	3	15		Load3	No
SUB.D	F8	F6	F2	4	7	8		
DIV.D	F10	F0	F6	5				
ADD.D	F6	F8	F2	6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Busy Op	V <sub>j</sub>	V <sub>k</sub>	Q <sub>j</sub>	Q <sub>k</sub>
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	Yes	MULTD	M(45+R3)	R(F4)	
0	Mult2	Yes	DIVD		M(34+R2)	Mult1

Done executing

## Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
15	Mult1	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

- Mult1 completed execution; what is waiting for it?

# Tomasulo Example: Cycle 16

FP EX Cycles : Add = 2 cycles, Multiply = 10, Divide = 40

Instruction status			Execution	Write		Busy	Address
Instruction	<i>j</i>	<i>k</i>	Issue	complete	Result		
L.D F6	34+	R2	1	3	4	Load1	No
L.D F2	45+	R3	2	4	5	Load2	No
MUL.D F0	F2	F4	3	15	16	Load3	No
SUB.D F8	F6	F2	4	7	8		
DIV.D F10	F0	F6	5				
ADD.D F6	F8	F2	6	10	11		

Reservation Stations			S1	S2	RS for <i>j</i>	RS for <i>k</i>
Time	Name	Busy Op	V <sub><i>j</i></sub>	V <sub><i>k</i></sub>	Q <sub><i>j</i></sub>	Q <sub><i>k</i></sub>
0	Add1	No				
0	Add2	No				
	Add3	No				
0	Mult1	No				
40	Mult2	Yes DIVD	M*F4	M(34+R2)		

Execution cycles remaining (execution actually starts next cycle)

Register result status

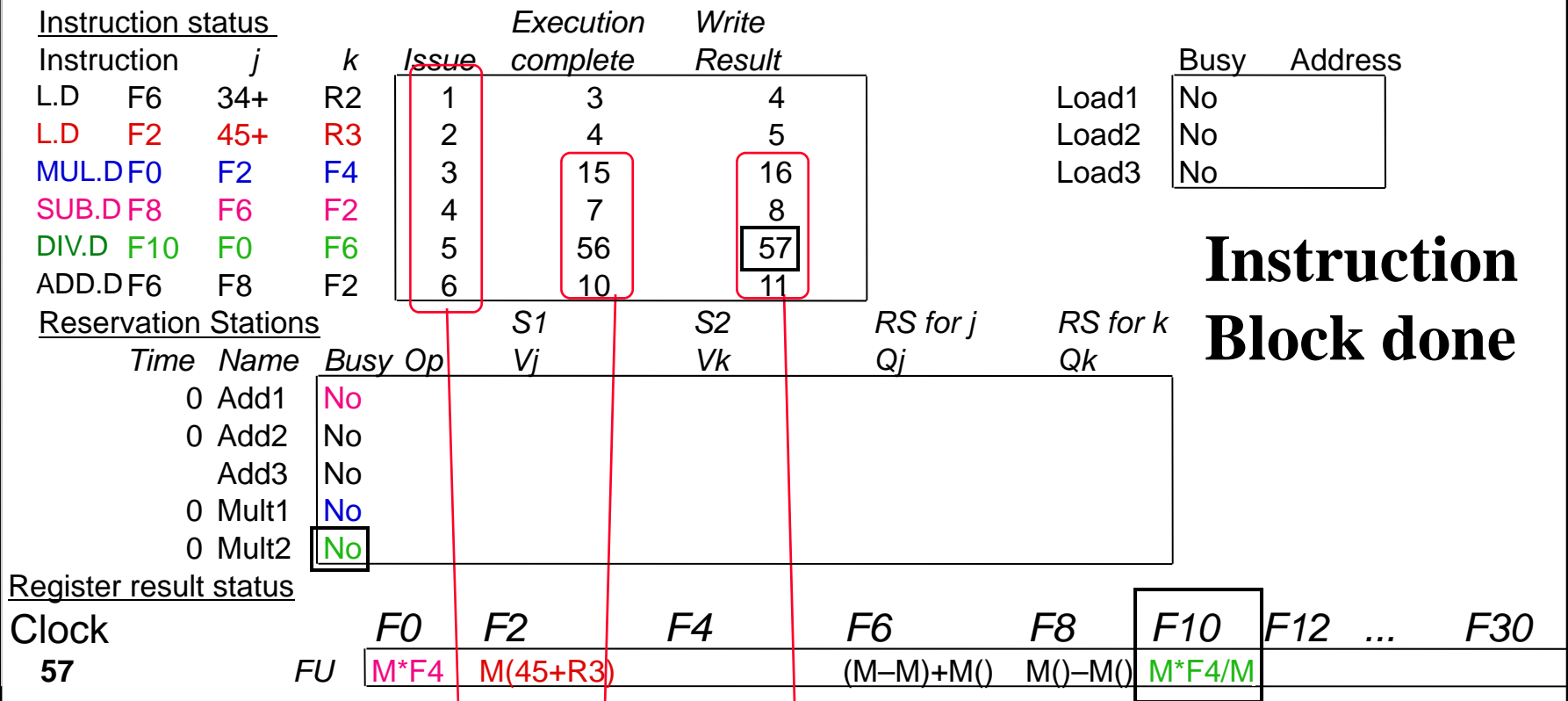
Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
16	M*F4	M(45+R3)		(M-M)+M()	M()-M()	Mult2			

**Only Divide instruction remains**  
**DIV.D execution will start next cycle (17)**

**CMPE550 - Shaaban**

# Tomasulo Example: Cycle 57

(vs 62 cycles for scoreboard)



**Instruction Block done**

- Again we have:
  - In-order issue,
  - Out-of-order execution, completion

# Tomasulo Loop Example

(Hardware-Based Version of Loop-Unrolling)

Loop: L.D	F0, 0(R1)
MUL.D	F4, F0, F2
S.D	F4, 0(R1)
DADDUI	R1, R1, # -8
BNE	R1, R2, Loop ; branch if R1 ≠ R2

Note independent loop iterations  
(the same loop used in loop unrolling example)

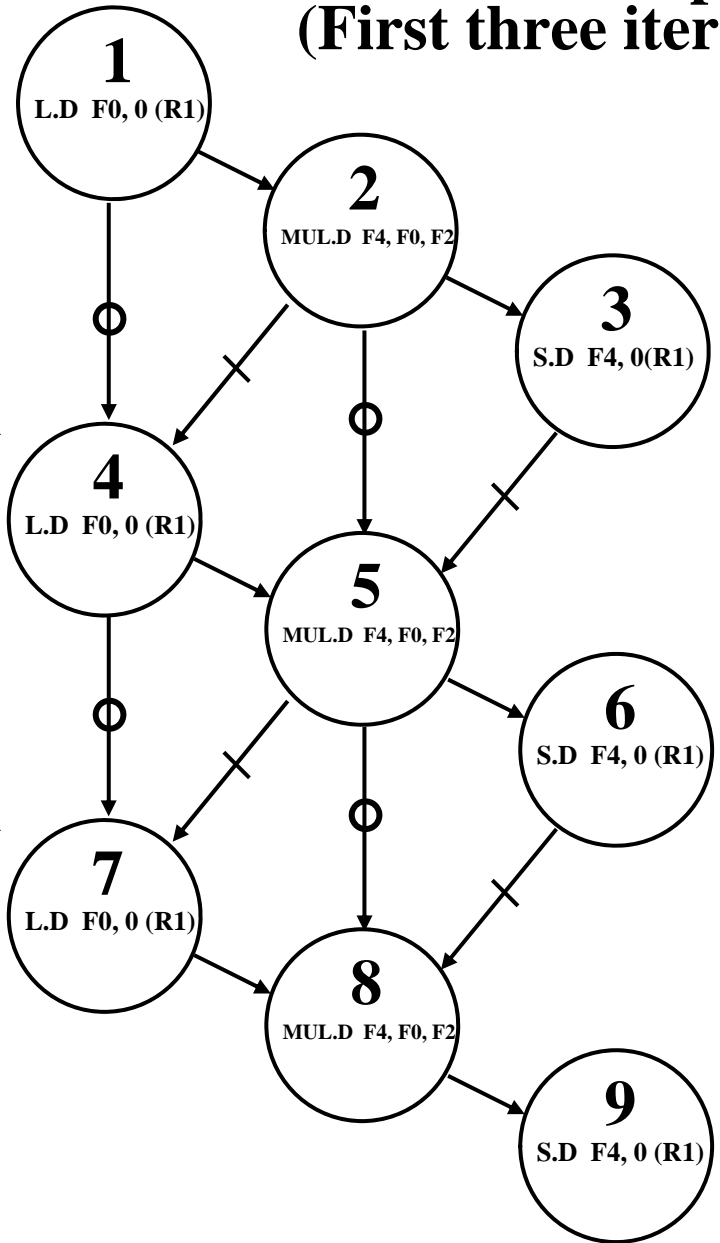
- Assume FP Multiply takes 4 execution clock cycles.
- Assume first load takes 8 cycles (possibly due to a cache miss), second load takes 4 cycles (cache hit). 3rd ....
- Assume R1 = 80 initially.
- Assume DADDUI only takes one cycle (issue) i.e. Perfect branch prediction. How?
- Assume branch resolved in issue stage (no EX or CDB write) Target?
- Assume branch is predicted taken and no branch misprediction. What if prediction Is wrong?
- No branch delay slot is used in this example.
- Stores take 4 cycles (ex, mem) and do not write on CDB
- We'll go over the execution to complete first two loop iterations.

Expanded from loop example in Chapter 2.5 (Third Edition Chapter 3.3)

**CMPE550 - Shaaban**

# Tomasulo Loop Example Dependency Graph (First three iterations shown)

First  
Iteration



Second  
Iteration

Third  
Iteration

Example Code

First Iteration	{	1	L.D	F0, 0 (R1)
		2	MUL.D	F4, F0, F2
		3	S.D	F4, 0(R1)
Second Iteration	{	4	L.D	F0, 0(R1)
		5	MUL.D	F4, F0, F2
		6	S.D	F4, 0(R1)
Third Iteration	{	7	L.D	F0, 0(R1)
		8	MUL.D	F4, F0, F2
		9	S.D	F4, 0(R1)

Loop maintenance (DADDUI)  
and branches (BNE) not shown

Name dependencies between iteration 3 instructions and iteration 1 instructions are not shown in graph

# Loop Example Cycle 0

(i.e at end of cycle 0)

Instruction status				Execution Write			Busy Address	
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	Issue	complete	Result	Busy	Address
L.D	F0	0 R1	1			Load1	No	Qi
MUL.D	F4	F0 F2	1			Load2	No	
S.D	F4	0 R1	1			Load3	No	
L.D	F0	0 R1	2			Store1	No	
MUL.D	F4	F0 F2	2			Store2	No	
S.D	F4	0 R1	2			Store3	No	

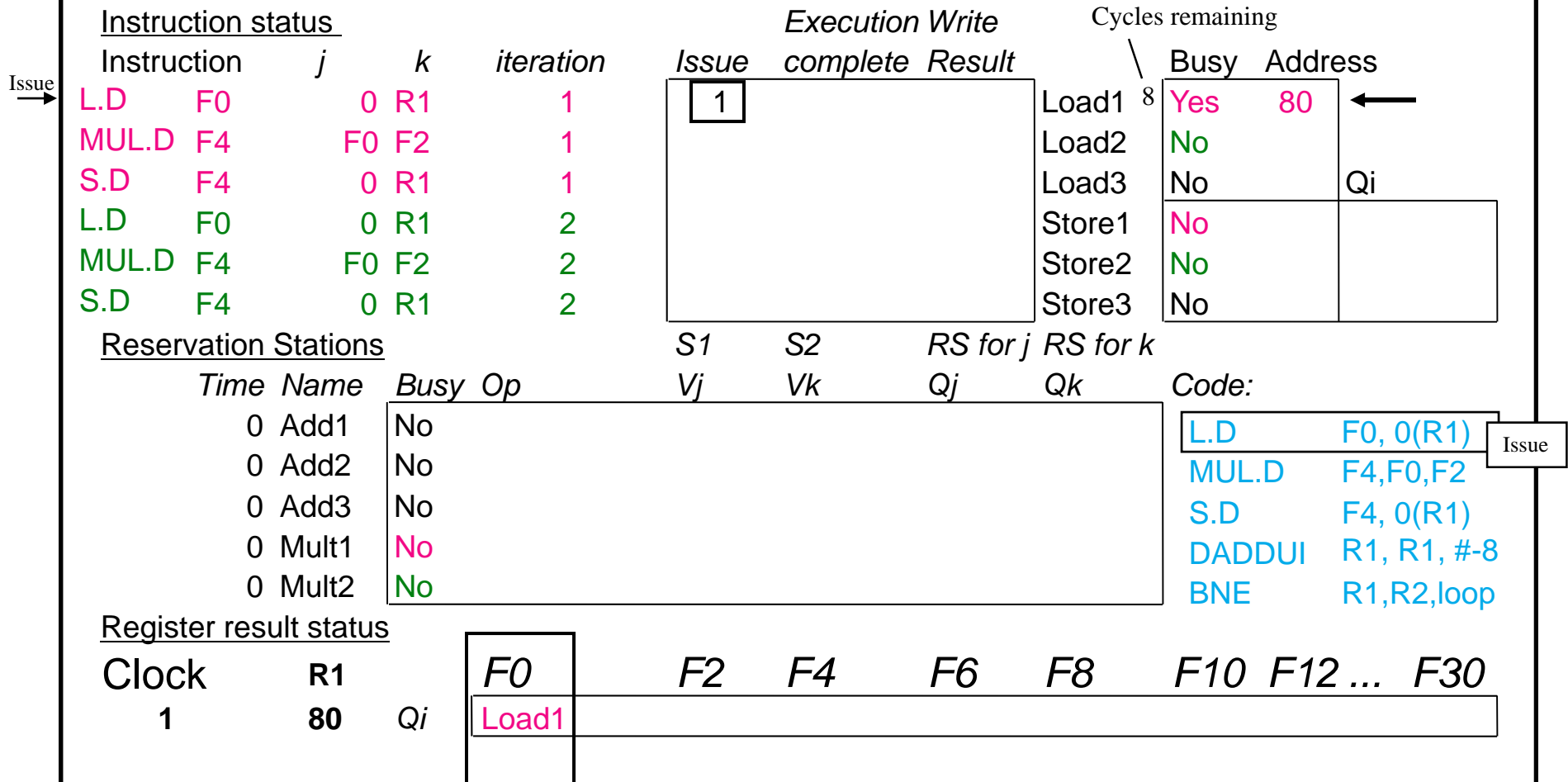
## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4,F0,F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	No						DADDUI R1, R1, #-8
0	Mult2	No						BNE R1,R2,loop

## Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30	
0	80	Qi										

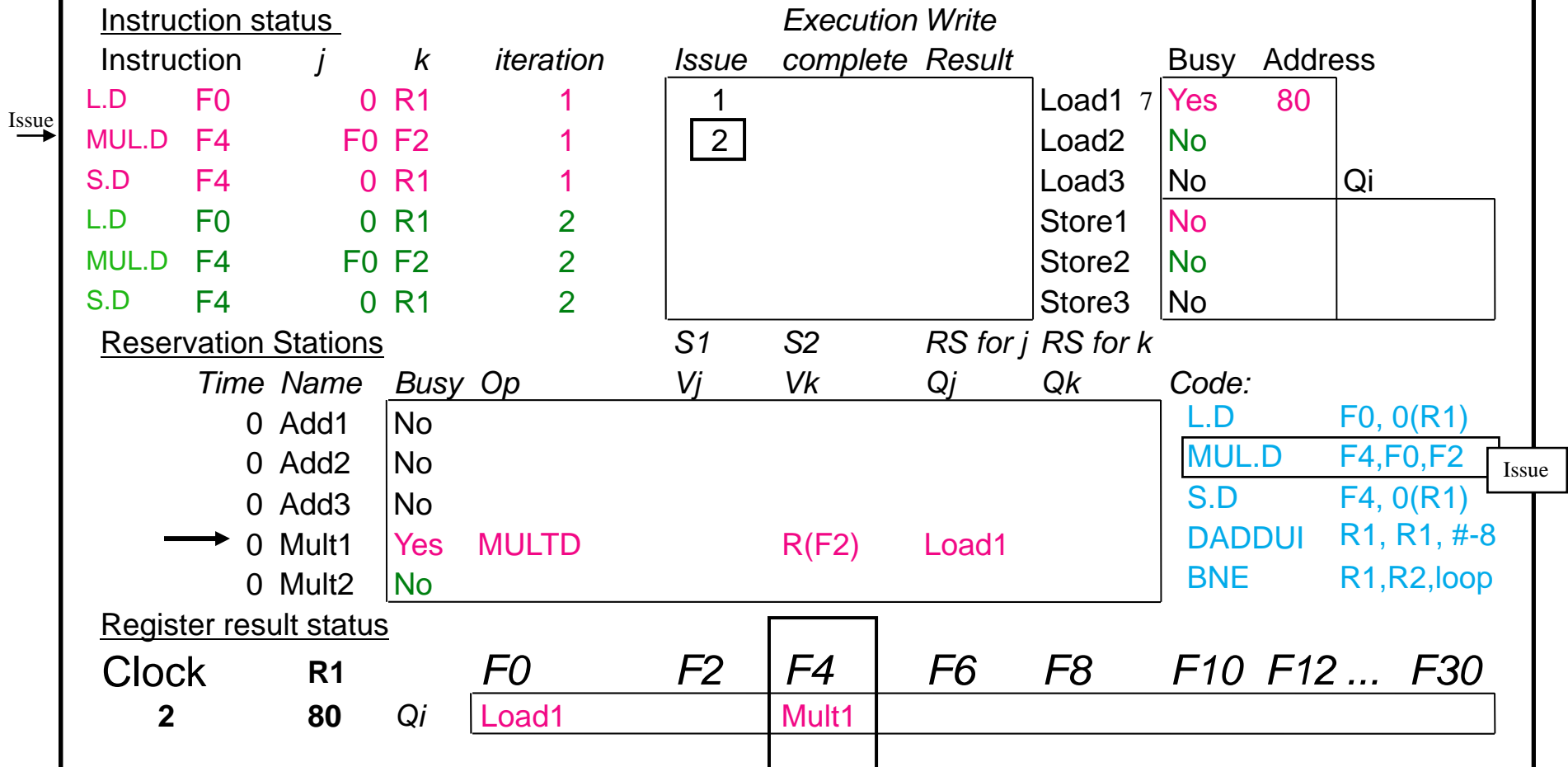
# Loop Example Cycle 1



First L.D issues, takes 8 cycles to complete execution (including mem access)



# Loop Example Cycle 2



First MUL.D issues, wait on first L.D (Load1) to write on CDB

# Loop Example Cycle 3

Instruction status				Execution Write			Busy Address		
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>complete</i>	<i>Result</i>	<i>Busy</i>	<i>Address</i>	
L.D	F0	0 R1	1	1		Load1	Yes	80	
MUL.D	F4	F0 F2	1	2		Load2	No		
S.D	F4	0 R1	1	3		Load3	No		Qi
L.D	F0	0 R1	2			Store1	Yes	80	Mult1
MUL.D	F4	F0 F2	2			Store2	No		
S.D	F4	0 R1	2			Store3	No		

Reservation Stations				<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>	<i>Code:</i>
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4,F0,F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD		R(F2)	Load1		DADDUI R1, R1, #-5
0	Mult2	No						BNE R1,R2,loop

Register result status										
Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
3	80	Qi	Load1							Mult1

First S.D issues, wait on first MUL.D (Mult1) to write on CDB

# Loop Example Cycle 4

## Instruction status

Instruction	<i>j</i>	<i>k</i>	iteration	Issue	Execution complete	Write Result	Busy	Address
L.D F0	0	R1	1	1		Load1	Yes	80
MUL.D F4	F0	F2	1	2		Load2	No	
S.D F4	0	R1	1	3		Load3	No	Qi
L.D F0	0	R1	2			Store1	Yes	80
MUL.D F4	F0	F2	2			Store2	No	
S.D F4	0	R1	2			Store3	No	

## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD		R(F2)	Load1		DADDUI R1, R1, #-1
0	Mult2	No						BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
4	72	Qi	Load1		Mult1					

First DADDUI issues (not shown)

CMPE550 - Shaaban

# Loop Example Cycle 5

Instruction status				Execution Write			Busy	Address
Instruction	<i>j</i>	<i>k</i>	iteration	Issue	complete	Result		
L.D	F0	0 R1	1	1		Load1	Yes	80
MUL.D	F4	F0 F2	1	2		Load2	No	
S.D	F4	0 R1	1	3		Load3	No	Qi
L.D	F0	0 R1	2			Store1	Yes	80
MUL.D	F4	F0 F2	2			Store2	No	
S.D	F4	0 R1	2			Store3	No	

## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD		R(F2)	Load1		DADDUI R1, R1, #-8
0	Mult2	No						BNE R1, R2, loop

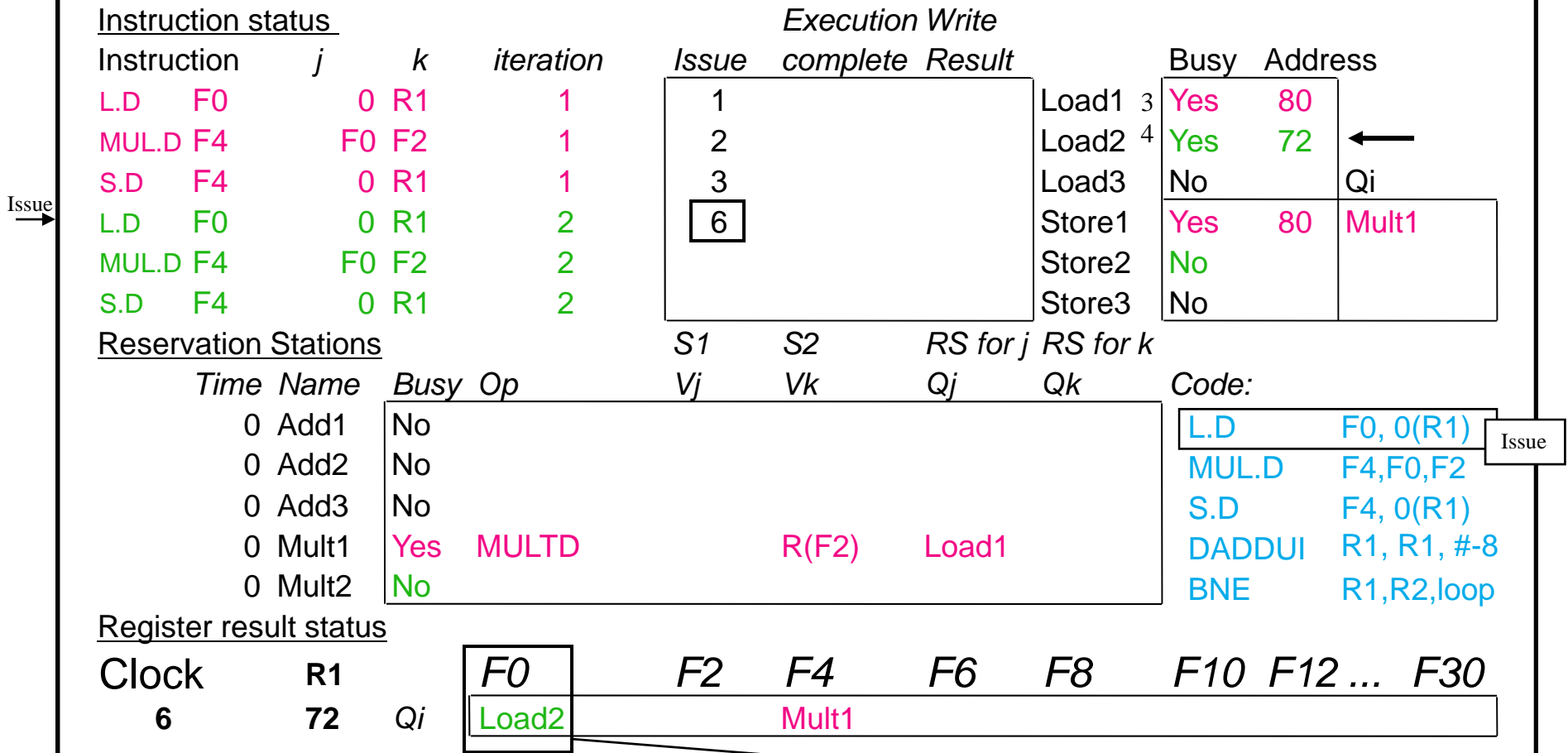
## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
5	72	Qi	Load1		Mult1				

First BNE issues (not shown), assumed predicted taken

CMPE550 - Shaaban

# Loop Example Cycle 6



- Second L.D. issues (will take four ex cycles) Note: F0 never sees Load1 result
- WAW between first and second L.D on F0 eliminated by register renaming

# Loop Example Cycle 7

Instruction status				Execution Write			Busy Address	
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>complete</i>	<i>Result</i>	<i>Busy</i>	<i>Address</i>
L.D F0	0	R1	1	1		Load1	Yes	80
MUL.D F4	F0	F2	1	2		Load2	Yes	72
S.D F4	0	R1	1	3		Load3	No	Qi
L.D F0	0	R1	2	6		Store1	Yes	80
MUL.D F4	F0	F2	2	7		Store2	No	
S.D F4	0	R1	2			Store3	No	

Reservation Stations				<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>	<i>Code:</i>
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>	
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD		R(F2)	Load1		DADDUI R1, R1, #-8
0	Mult2	Yes	MULTD		R(F2)	Load2		BNE R1, R2, loop

Register result status										
Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Qi	Load2	Mult2						

- Second MUL.D issues (to RS Mult2) Note: F4 never sees Mult1 result
- WAW between first and second MUL.D on F4 eliminated by register renaming

# Loop Example Cycle 8

Instruction status				Execution Write			Busy Address	
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>complete</i>	<i>Result</i>		
L.D	F0	0 R1	1	1		Load1	1	Yes 80
MUL.D	F4	F0 F2	1	2		Load2	2	Yes 72
S.D	F4	0 R1	1	3		Load3		No Qi
L.D	F0	0 R1	2	6		Store1		Yes 80 Mult1
MUL.D	F4	F0 F2	2	7		Store2		Yes 72 Mult2
S.D	F4	0 R1	2	8		Store3		No

Reservation Stations			<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>	
<i>Time</i>	<i>Name</i>	<i>Busy Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>	<i>Code:</i>
0	Add1	No					L.D F0, 0(R1)
0	Add2	No					MUL.D F4, F0, F2
0	Add3	No					S.D F4, 0(R1)
0	Mult1	Yes MULTD		R(F2)	Load1		DADDUI R1, R1, #-5
0	Mult2	Yes MULTD		R(F2)	Load2		BNE R1, R2, loop

Register result status		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
Clock	R1									
8	72	Qi	Load2	Mult2						

- Second S.D issues (to RS Store2)

# Loop Example Cycle 9

Instruction status				Execution Write			First Load EX Done	
Instruction	<i>j</i>	<i>k</i>	iteration	Issue	complete	Result	Busy	Address
L.D	F0	0 R1	1	1	9	Load1	0 Yes	80
MUL.D	F4	F0 F2	1	2		Load2	1 Yes	72
S.D	F4	0 R1	1	3		Load3	No	Qi
L.D	F0	0 R1	2	6		Store1	Yes	80 Mult1
MUL.D	F4	F0 F2	2	7		Store2	Yes	72 Mult2
S.D	F4	0 R1	2	8		Store3	No	

## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD		R(F2)	Load1		DADDUI R1, R1, #-
0	Mult2	Yes	MULTD		R(F2)	Load2		BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
9	64	Qi	Load2	Mult2					

- Issue second DADDUI (not shown)
- Load1 completing; what is waiting for it?



# Loop Example Cycle 10

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2			Yes	72
S.D F4	0	R1	1	3			No	Qi
L.D F0	0	R1	2	6	10		Yes	80
MUL.D F4	F0	F2	2	7			Yes	72
S.D F4	0	R1	2	8			No	

Second Load EX Done

## Reservation Stations

Time Name Busy Op S1 S2 RS for *j* RS for *k* Code:

Time	Name	Busy	Op	V <sub>j</sub>	V <sub>k</sub>	Q <sub>j</sub>	Q <sub>k</sub>
0	Add1	No					
0	Add2	No					
0	Add3	No					
4	Mult1	Yes	MULTD	M(80)	R(F2)		
0	Mult2	Yes	MULTD		R(F2)	Load2	

L.D F0, 0(R1)  
 MUL.D F4, F0, F2  
 S.D F4, 0(R1)  
 DADDUI R1, R1, #-8  
 BNE R1, R2, loop

Execution cycles remaining (execution actually starts next cycle)

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
10	64	Qi	Load2	Mult2					

- Load1 result forwarded via CDB to Mult1, execution will start next cycle 11
- Issue second BNE (not shown)
- Load2 completing; what is waiting for it?

CMPE550 - Shaaban

# Loop Example Cycle 11

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2			No	
S.D F4	0	R1	1	3			Yes	64 Qi ←
L.D F0	0	R1	2	6	10	11	Yes	80 Mult1
MUL.D F4	F0	F2	2	7			Yes	72 Mult2
S.D F4	0	R1	2	8			No	

## Reservation Stations

Time	Name	Busy	Op	S1 V <sub>j</sub>	S2 V <sub>k</sub>	RS for <i>j</i> Q <sub>j</sub>	RS for <i>k</i> Q <sub>k</sub>	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
3	Mult1	Yes	MULTD	M(80)	R(F2)			DADDUI R1, R1, #-8
4	Mult2	Yes	MULTD	M(72)	R(F2)			BNE R1, R2, loop

Execution cycles remaining (execution actually starts next cycle)

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
11	64	Qi	Load3						
				Mult2					

**Load2 result forwarded via CDB to Mult2, execution will start next cycle 12**  
**Third iteration L.D. issues (to RS Load3)**

**CMPE550 - Shaaban**

# Loop Example Cycle 12

Instruction status				Execution Write				
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>complete</i>	<i>Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2			No	
S.D F4	0	R1	1	3			Yes	64 Qi
L.D F0	0	R1	2	6	10	11	Yes	80 Mult1
MUL.D F4	F0	F2	2	7			Yes	72 Mult2
S.D F4	0	R1	2	8			No	

Reservation Stations				<i>S1</i>	<i>S2</i>	<i>RS for j</i>	<i>RS for k</i>	
<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>	<i>Code:</i>
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2 – Issue?
0	Add3	No						S.D F4, 0(R1)
2	Mult1	Yes	MULTD	M(80)	R(F2)			DADDUI R1, R1, #-8
3	Mult2	Yes	MULTD	M(72)	R(F2)			BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
12	64 Qi	Load3		Mult2					

Issue third iteration MUL.D ?

CMPE550 - Shaaban

# Loop Example Cycle 13

Instruction status				Execution			Write	
Instruction	<i>j</i>	<i>k</i>	iteration	Issue	complete	Result	Busy	Address
L.D F0	0	R1	1	1	9	10	Load1	No
MUL.D F4	F0	F2	1	2			Load2	No
S.D F4	0	R1	1	3			Load3	Yes 64 Qi
L.D F0	0	R1	2	6	10	11	Store1	Yes 80 Mult1
MUL.D F4	F0	F2	2	7			Store2	Yes 72 Mult2
S.D F4	0	R1	2	8			Store3	No

Reservation Stations				S1	S2	RS for <i>j</i>	RS for <i>k</i>	Code:
Time	Name	Busy	Op	V <sub>j</sub>	V <sub>k</sub>	Q <sub>j</sub>	Q <sub>k</sub>	
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1) <span style="font-size: 2em;">}</span> Issue?
1	Mult1	Yes	MULTD	M(80)	R(F2)			DADDUI R1, R1, #-8
2	Mult2	Yes	MULTD	M(72)	R(F2)			BNE R1, R2, loop

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock	R1									
13	64 Qi	Load3		Mult2						

**Issue third iteration MUL.D, S.D ?**

# Loop Example Cycle 14

Instruction status				Execution Write			Busy	Address
Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>complete</i>	<i>Result</i>		
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2	14		No	
S.D F4	0	R1	1	3			Yes	64 Qi
L.D F0	0	R1	2	6	10	11	Yes	80 Mult1
MUL.D F4	F0	F2	2	7			Yes	72 Mult2
S.D F4	0	R1	2	8			No	

## Reservation Stations

Time	Name	Busy	Op	S1 <i>V<sub>j</sub></i>	S2 <i>V<sub>k</sub></i>	RS for <i>j</i> <i>Q<sub>j</sub></i>	RS for <i>k</i> <i>Q<sub>k</sub></i>	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4,F0,F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD	M(80)	R(F2)			DADDUI R1, R1, #-8
1	Mult2	Yes	MULTD	M(72)	R(F2)			BNE R1,R2,loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
14	64 Qi	Load3		Mult2					

- Mult1 completing; what is waiting for it?

# Loop Example Cycle 15

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3			Yes	64 Qi
L.D F0	0	R1	2	6	10	11	Yes	80 M(80)*R(F2)
MUL.D F4	F0	F2	2	7	15		Yes	72 Mult2
S.D F4	0	R1	2	8			No	

Third Load EX Done

## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2 Issue?
0	Add3	No						S.D F4, 0(R1)
0	Mult1	No						DADDUI R1, R1, #-8
0	Mult2	Yes	MULTD	M(72)	R(F2)			BNE R1, R2, loop

Available by end of this cycle

EX Done

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Qi	Load3	Mult2						

- Mult2 completing; what is waiting for it?
  - Third iteration L.D done execution
- Issue third multiply?

CMPE550 - Shaaban

# Loop Example Cycle 16

## Instruction status

Instruction	<i>j</i>	<i>k</i>	iteration	Issue	Execution complete	Write Result	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3			No	Qi
L.D F0	0	R1	2	6	10	11	Yes	80
MUL.D F4	F0	F2	2	7	15	16	Yes	72
S.D F4	0	R1	2	8			No	

## Reservation Stations

Time	Name	Busy	Op
0	Add1	No	
0	Add2	No	
0	Add3	No	
0	Mult1	Yes	MULTD
0	Mult2	No	

S1 S2 RS for j RS for k  
Vj Vk Qj Qk

M(64) R(F2)

Code:

L.D F0, 0(R1)  
 MUL.D F4, F0, F2  
 S.D F4, 0(R1)  
 DADDUI R1, R1, #-8  
 BNE R1, R2, loop

Issue

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
16	64	Qi		Mult1					

**Issue third iteration MUL.D (to RS Mult1)**

**CMPE550 - Shaaban**

# Loop Example Cycle 17

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3			No	Qi
L.D F0	0	R1	2	6	10	11	Yes	80
MUL.D F4	F0	F2	2	7	15	16	Yes	72
S.D F4	0	R1	2	8			Yes	64

## Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
------	------	------	----	----------	----------	----------------	----------------	-------

Execution cycles remaining (execution actually starts next cycle) Delayed one cycle

0	Add1	No						
0	Add2	No						
0	Add3	No						
4	Mult1	Yes	MULTD	M(64)	R(F2)			
0	Mult2	No						

L.D F0, 0(R1)  
 MUL.D F4, F0, F2  
 S.D F4, 0(R1)  
 DADDUI R1, R1, #-8  
 BNE R1, R2, loop

Issue

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
17	64	Qi		Mult1					

**Third iteration L.D writes on CDB (delayed one cycle due to CDB conflict)  
 Issue third iteration S.D (to RS Store3)**

**CMPE550 - Shaaban**



# Loop Example Cycle 18

## Instruction status

Instruction	<i>j</i>	<i>k</i>	iteration	Issue	Execution complete	Write Result	Busy	Address
L.D F0	0	R1	1	1	9	10	Load1	No
MUL.D F4	F0	F2	1	2	14	15	Load2	No
S.D F4	0	R1	1	3			Load3	No
L.D F0	0	R1	2	6	10	11	Store1 <sup>1</sup>	Yes 80
MUL.D F4	F0	F2	2	7	15	16	Store2 <sup>2</sup>	Yes 72
S.D F4	0	R1	2	8			Store3	Yes 64

## Reservation Stations

Time	Name	Busy	Op	S1 V <sub>j</sub>	S2 V <sub>k</sub>	RS for <i>j</i> Q <sub>j</sub>	RS for <i>k</i> Q <sub>k</sub>	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
3	Mult1	Yes	MULTD	M(64)	R(F2)			DADDUI R1, R1, #-8
0	Mult2	No						BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
18	56	Qi	Mult1						

**Issue third iteration DADDUI**

**CMPE550 - Shaaban**

(First Loop Iteration Done)

# Loop Example Cycle 19

Instruction status

Instruction	<i>j</i>	<i>k</i>	iteration	Issue	Execution complete	Write Result	Busy	Address
L.D F0	0	R1	1	1	9	10	No	
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3	19		No	Qi
L.D F0	0	R1	2	6	10	11	No	
MUL.D F4	F0	F2	2	7	15	16	Yes	72 M(72)*R(72)
S.D F4	0	R1	2	8			Yes	64 Mult1

First Store Done

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	No					
0	Add3	No					
2	Mult1	Yes	MULTD	M(64)	R(F2)		
0	Mult2	No					

Code:

L.D	F0, 0(R1)
MUL.D	F4, F0, F2
S.D	F4, 0(R1)
DADDUI	R1, R1, #-8
BNE	R1, R2, loop

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
19	56	Qi	Mult1						

First S.D done (No write on CDB for stores) First loop iteration done  
Issue third iteration BNE

**CMPE550 - Shaaban**

(First Two Loop Iterations Done)

# Loop Example Cycle 20

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	Yes	54
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3	19		No	Qi
L.D F0	0	R1	2	6	10	11	No	
MUL.D F4	F0	F2	2	7	15	16	No	
S.D F4	0	R1	2	8	20		Yes	64 Mult1

## Reservation Stations

Time	Name	Busy	Op	S1 <i>Vj</i>	S2 <i>Vk</i>	RS for <i>j</i> <i>Qj</i>	RS for <i>k</i> <i>Qk</i>	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
1	Mult1	Yes	MULTD	M(64)	R(F2)			DADDUI R1, R1, #-8
0	Mult2	No						BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
20	56	Qi	Load1	Mult1					

Second S.D done (No write on CDB for stores) Second loop iteration done  
Issue fourth iteration L.D (to RS Load1)

# Loop Example Cycle 21

## Instruction status

Instruction	<i>j</i>	<i>k</i>	<i>iteration</i>	<i>Issue</i>	<i>Execution complete</i>	<i>Write Result</i>	Busy	Address
L.D F0	0	R1	1	1	9	10	Yes	54
MUL.D F4	F0	F2	1	2	14	15	No	
S.D F4	0	R1	1	3	19		No	Qi
L.D F0	0	R1	2	6	10	11	No	
MUL.D F4	F0	F2	2	7	15	16	No	
S.D F4	0	R1	2	8	20		Yes	64 Mult1

## Reservation Stations

Time	Name	Busy	Op	S1 <i>Vj</i>	S2 <i>Vk</i>	RS for <i>j</i> <i>Qj</i>	RS for <i>k</i> <i>Qk</i>	Code:
0	Add1	No						L.D F0, 0(R1)
0	Add2	No						MUL.D F4, F0, F2
0	Add3	No						S.D F4, 0(R1)
0	Mult1	Yes	MULTD	M(64)	R(F2)			DADDUI R1, R1, #-8
0	Mult2	Yes	MULTD		R(F2)	Load1		BNE R1, R2, loop

## Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
21	56	Qi	Load3						Mult1

**Mult1 (third iteration MUL.D) completing; what is waiting for it?  
Issue fourth iteration MUL.D (to RS Mult2)**

**CMPE550 - Shaaban**

# Tomasulo Loop Example Timing Diagram

Cycle

Iteration

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	L.D.	I	E	E	E	E	E	E	E	W											
	MUL.D		I								E	E	E	E	W						
	S.D.			I												E	E	E	E		
	DADDUI				I																
	BNE					I															
2	L.D.					I	E	E	E	E	W										
	MUL.D						I					E	E	E	E	W					
	S.D.							I									E	E	E	E	
	DADDUI								I												
	BNE									I											
3	L.D.										I	E	E	E	E		W				
	MUL.D															I		E	E	E	E
	S.D.																I				
	DADDUI																	I			
	BNE																		I		
4	L.D.																			I	E
	MUL.D																				I
	S.D.																				
	DADDUI																				
	BNE																				

3rd L.D write delayed one cycle

3rd MUL.D issue delayed until mul RS is available

I = Issue    E = Execute    W = Write Result on CDB