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 (RAW) 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.

Dynamic = Done at runtime by CPU

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

- Dynamic scheduling: (out-of-order execution)
  - Uses a CPU-based mechanism to reorder or rearrange instruction execution order to reduce stalls dynamically at runtime.
    - Better dynamic exploitation of instruction-level parallelism (ILP).

Why?

  - 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.

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

Fully In-Order Integer Pipeline:

IF ⟶ ID ⟶ EX ⟶ MEM ⟶ WB

All instructions go through all pipelines stages in program order

Date Hazards: RAW: Possible WAR, WAW: Impossible

In-Order Pipeline with FP Support:

IF ⟶ ID ⟶ EX ⟶ MEM ⟶ WB

Up to start of EX in program order

Can be reached out of program order

Date Hazards: RAW, WAW: Possible WAR: Impossible

Out-of-Order Execution Pipelined Processor (Dynamically Scheduled):

Decode (ID)

IF ⟶ Issue ⟶ Read Operands ⟶ EX ⟶ MEM ⟶ WB

In program order

Can be reached out of program order Including:

- Read Operands (Part of ID)
- Start of Execution

Date Hazards: RAW, WAW, WAR: All Possible

All Pipelines Are Still Single Issue

Dynamic Pipeline Scheduling: *The Concept*

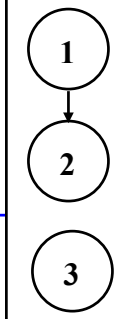
Dynamic = Done at runtime by CPU

(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).

Dependency Graph



Example:

Program Order

True Data Dependency

1 DIV.D F0, F2, F4

2 ADD.D F10, F0, F8

3 SUB.D F12, F8, F14

Does not depend on DIV.D or ADD.D

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.

- 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

Dynamic Pipeline Scheduling

- Dynamic instruction scheduling is accomplished by:
  - Dividing the Instruction Decode ID stage into two stages or steps:
    1. Issue: Decode instructions, check for structural hazards. (In-order)
       - + 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.
    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 1 in order but can be stalled or pass each other in the read operands stage 2).
  - 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.

Always done in program order

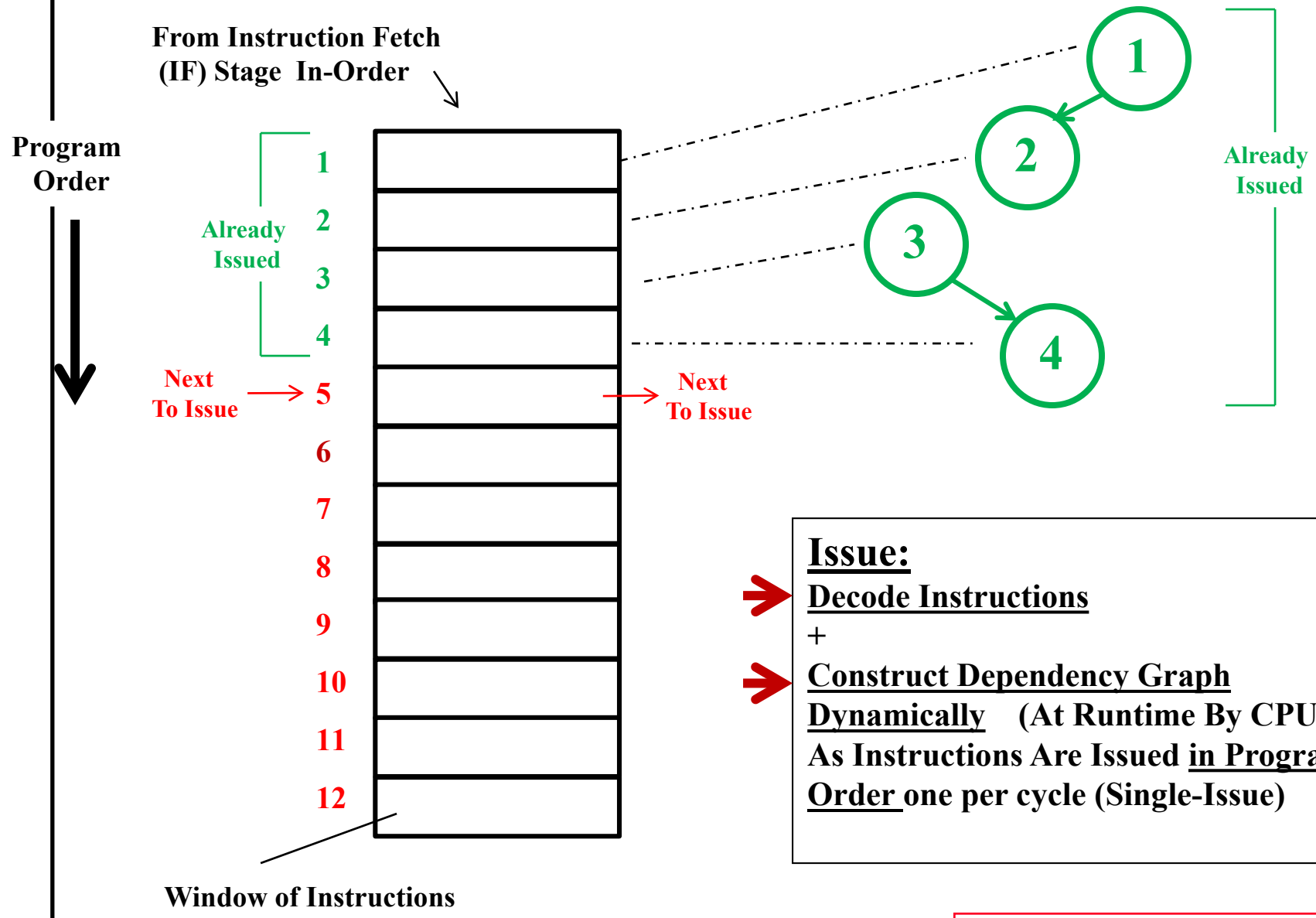
Can be done out of program order

- Two approaches to dynamic scheduling:
  - FYI 1 – Dynamic scheduling with the Scoreboard used first in CDC6600 (Control Data Corp.) (1963)
  - 2 – The Tomasulo approach pioneered by the IBM 360/91 (1966)

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

Single Instruction Issue Per Cycle (In-Order)



Issue:

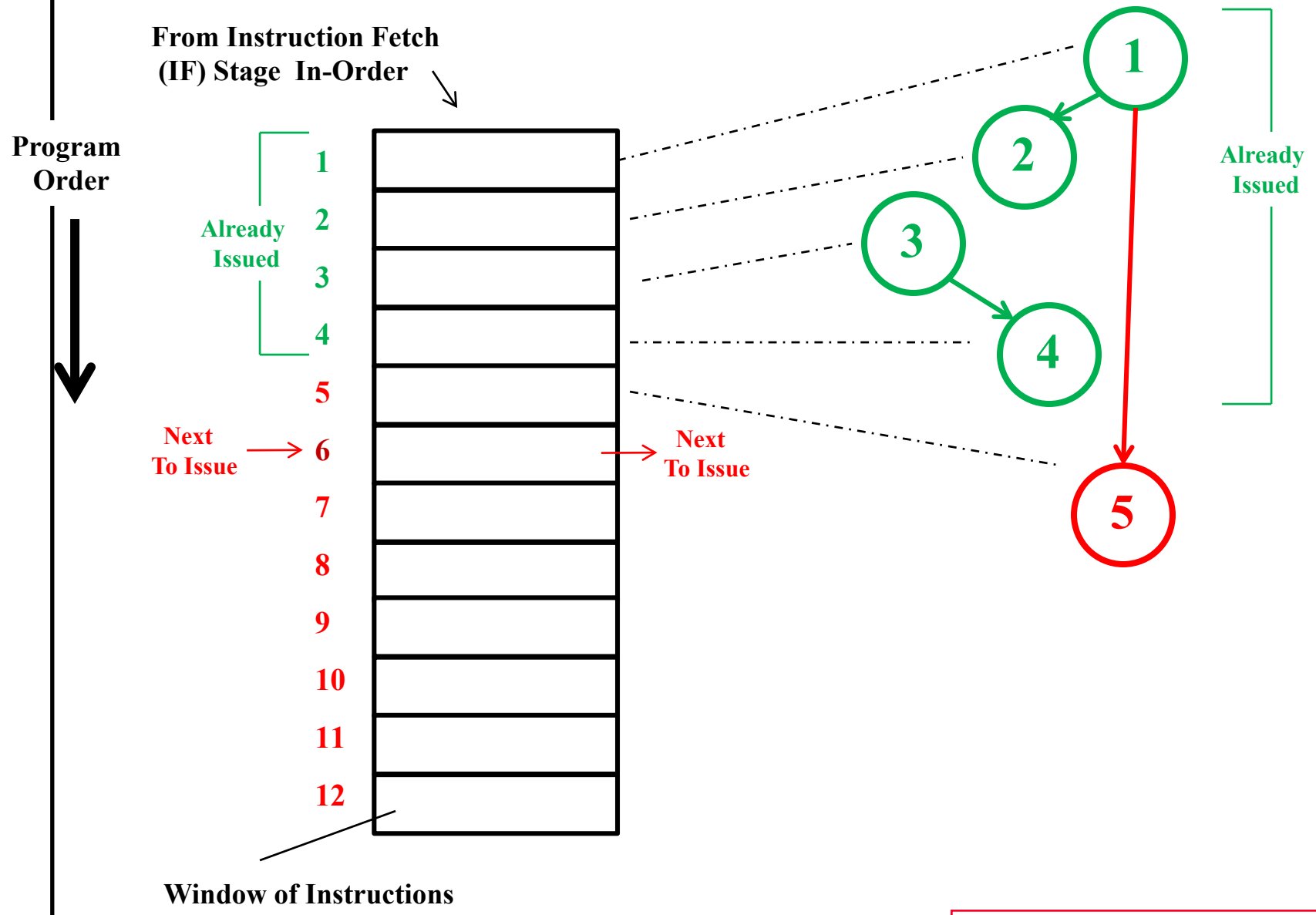
Decode Instructions

+

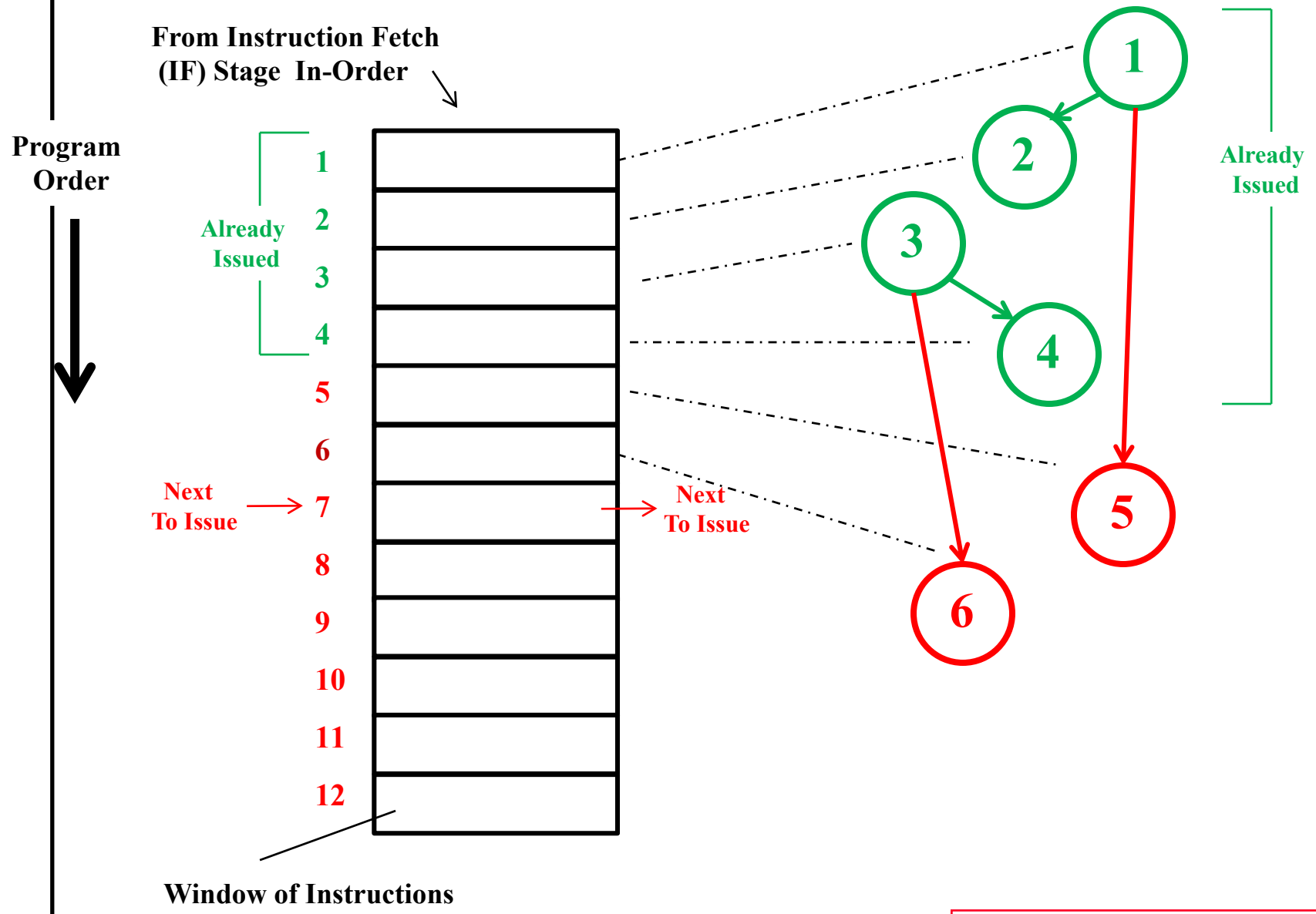
Construct Dependency Graph Dynamically (At Runtime By CPU)

As Instructions Are Issued **in Program Order** one per cycle (Single-Issue)

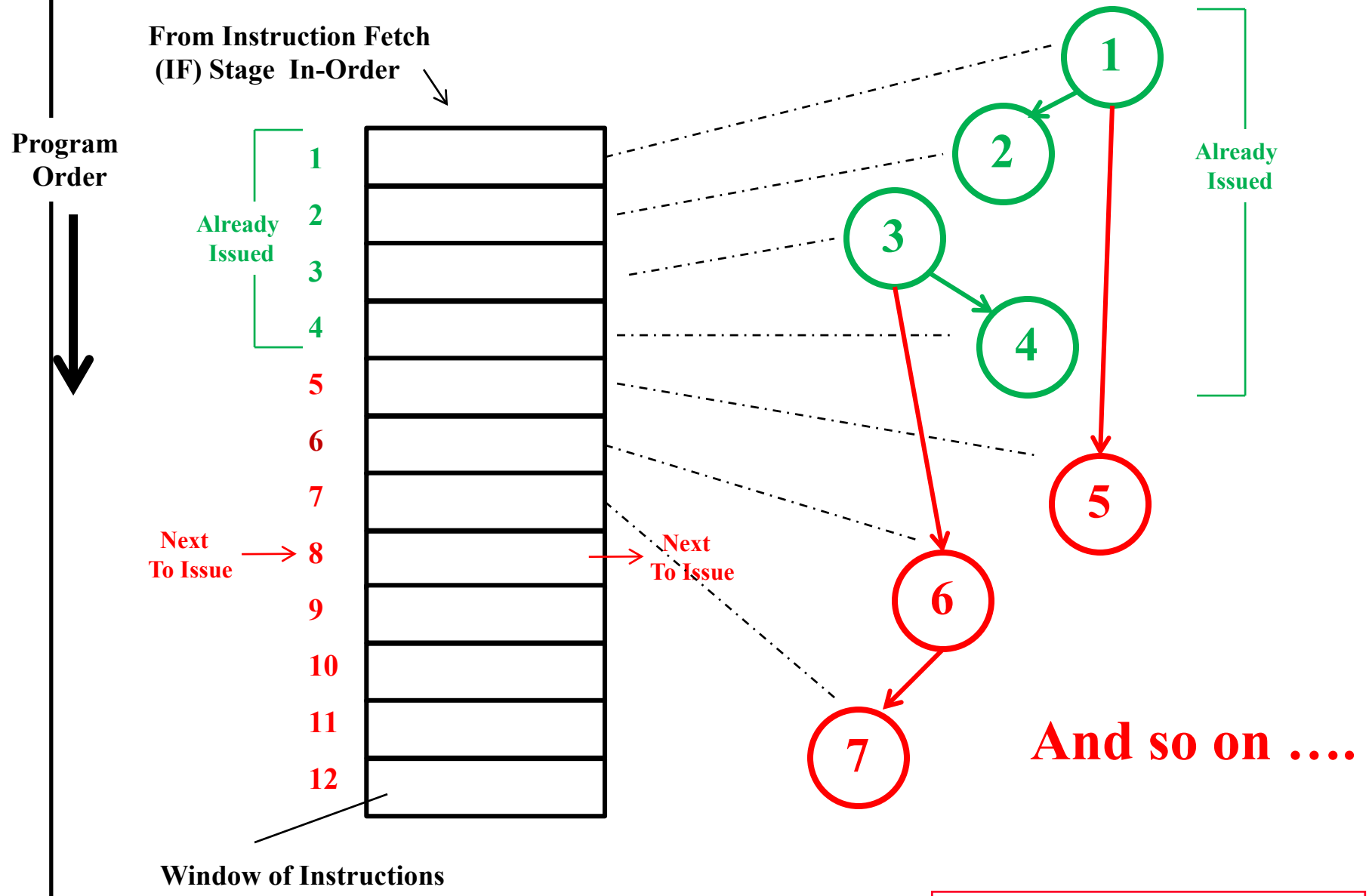
Single Instruction Issue Per Cycle (In-Order)



Single Instruction Issue Per Cycle (In-Order)



Single Instruction Issue Per Cycle (In-Order)



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

No Forwarding

EX Includes MEM

No changes to Instruction Fetch (IF)

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

Includes MEM

Issue

Read Operands

- Every instruction goes through the scoreboard where a record of data dependencies is constructed (ID1 corresponds to instruction issue).

In ID1 (Issue) In-Order

- 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

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

Introduced in CDC6600 (1963)

Instruction Execution Stages with A Scoreboard

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

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

1 Always done in program order

2 • A functional unit for the instruction is available (No structural hazard).

3 • 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.

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

Stage 0: Fetch, no changes, in-order

Scoreboard Stages

Including Instruction Fetch (IF): No changes, still in-order

In program order

Can be reached out of program order Including Read Operands (ID2) and Start of Execution

IF → ID1 → ID2 → EX → WB

ID1 Issue

- Decode instruction.

+

- Dynamic construction of data dependency graph.

- Instruction issued 1- in-order If:

2. A suitable functional unit (FU) is available (No structural hazard).
3. The instruction result destination register is not marked for writing by an earlier active instruction (No WAW hazard possible).

If any of the above three requirements are not met: ID1 (issue) is stalled.

ID2 Read Operands

- When all source operands are available/ready in registers,:

  All operands are read from registers all at once (no forwarding supported) and then start execution .

- RAW hazards resolved here.

- This completes ID.

EX Includes MEM

- Operate./execute on operands read in ID2

- Also Includes MEM stage.

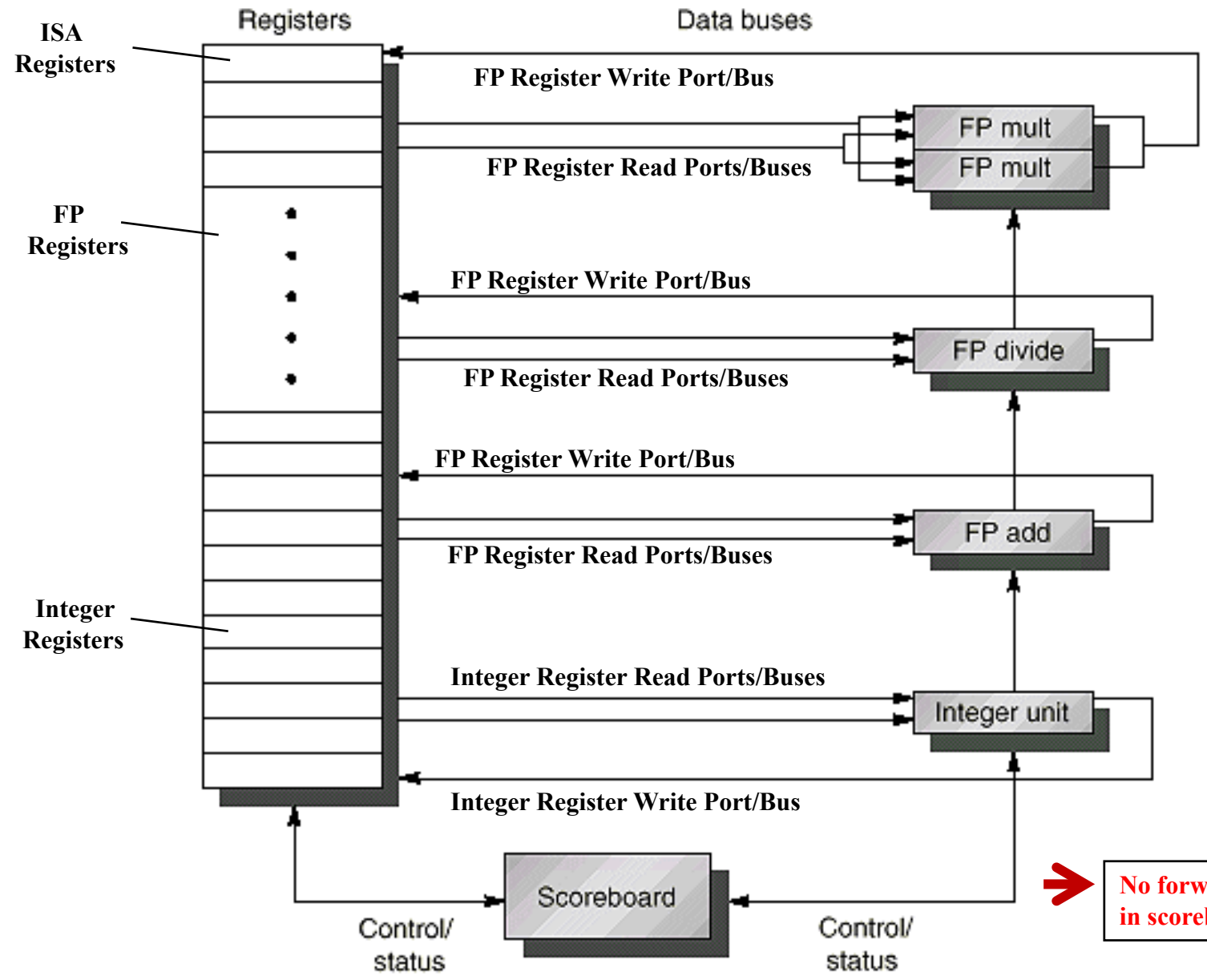
WB

- Write result to destination register.

- Once the scoreboard senses that a functional unit completed execution, it checks for WAR hazards and potentially stalls WB to prevent WAR.

- No WB for stores or branches

Date Hazards: RAW, WAW, WAR: All Possible



No forwarding supported in scoreboard

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

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 (If Any)
- **Rj, Rk** Flags indicating when Fj, Fk are ready (Yes or = 1 means ready)

Used/Needed To Track Data Dependencies

0 = No = Not ready
1 = Yes = 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 possibly stall issue

e.g

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

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

The Scoreboard: Detailed Pipeline Control

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

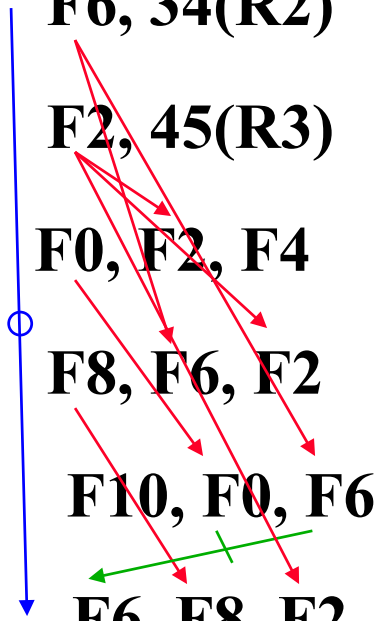
In Fourth Edition: Appendix A.7
(In Third Edition: Appendix A.8)

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

```
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
```



All functional units are not pipelined
(similar to CDC6600)

Real Data Dependence (RAW)

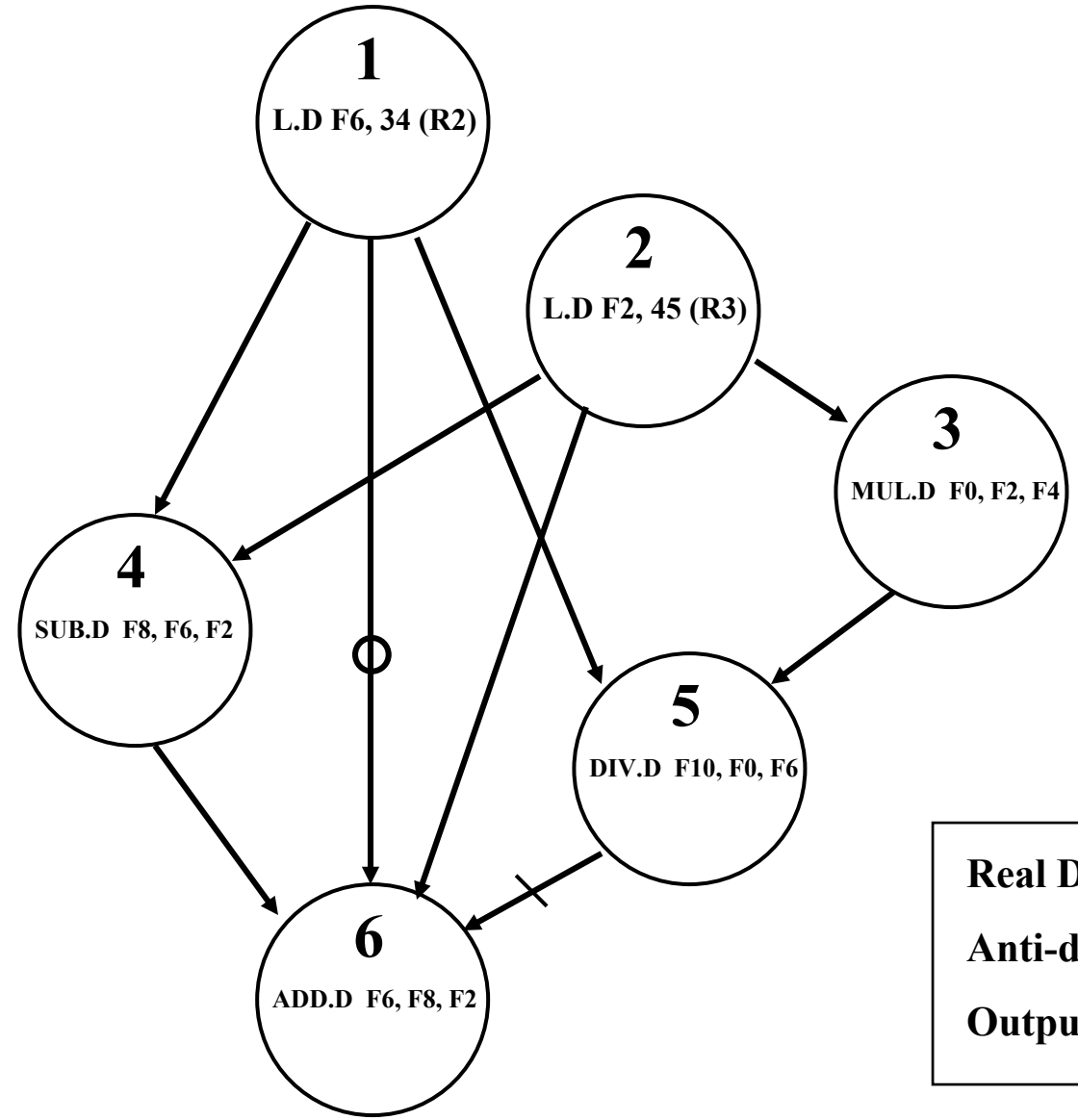
Anti-dependence (WAR)

Output Dependence (WAW)

Assume EX and MEM for L.D. done in one cycle

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

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)

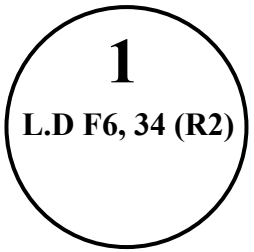
Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

Dependency Graph For Example Code

Cycle 1



Issue Instruction 1 to Only Integer Unit

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:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 1

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

Issue →

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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				

Means at end of Cycle 1

Producer of Result Needed | **Ready Flags**

Functional unit status

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

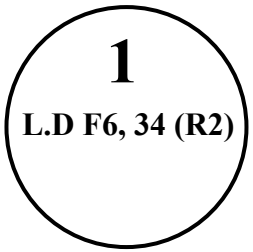
Register result status

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

→ **Issue first L.D**

Dependency Graph For Example Code

Cycle 2



Read Operands (ID2)

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

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

Date Dependence:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 2

End of

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

Issue ?

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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				

Functional unit status

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

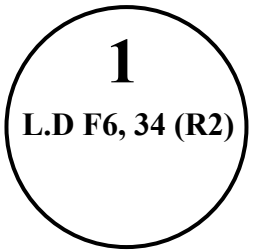
Register result status

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

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

Dependency Graph For Example Code

Cycle 3



EX + MEM Access Done

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
```

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

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

Date Dependence:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 3

End of

Instruction status

Instruction		j	k	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				

Issue ? → MUL.D F0 F2 F4

?

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

Functional unit status

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

Register result status

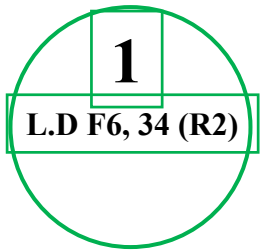
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU				Integer					

? • **Issue MUL.D? No, cannot issue out of order**

(second L.D not issued yet)

Dependency Graph For Example Code

Cycle 4



WB Done

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

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

Date Dependence:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 4

End of

Instruction status

Instruction		j	k	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	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Actually free end of this cycle 4

(available for instruction issue next cycle)

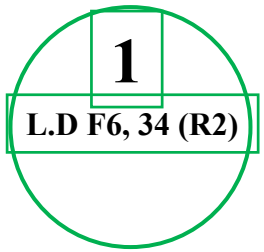
Register result status

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

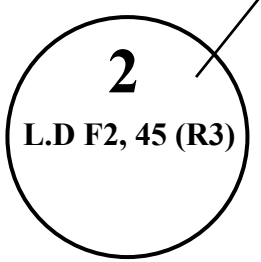
→ Issue second L.D? Next Cycle →

Dependency Graph For Example Code

Cycle 5



Issue Instruction 2 to Integer Unit



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:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 5

End of

Issue →

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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				

Functional unit status

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

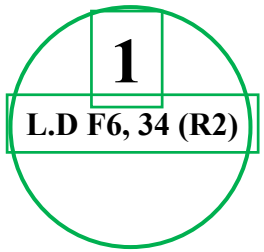
Register result status

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

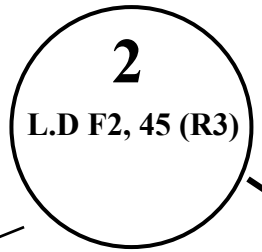
→ **Issue second L.D**

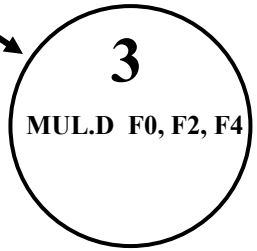
Dependency Graph For Example Code

Cycle 6



Issue Instruction 3 to Mult1 Functional Unit





Read Operands (ID2)

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:
(2, 3)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 6

End of

Issue →

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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				

Producer of Result Needed

Ready Flags

Functional unit status

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

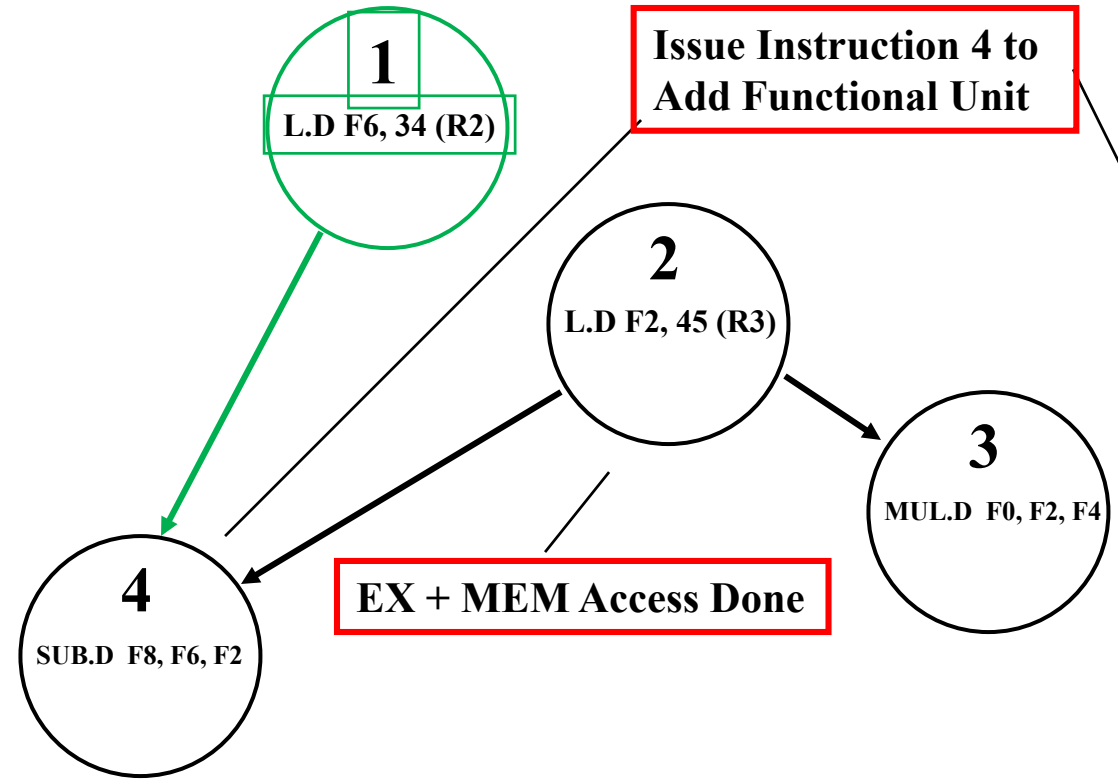
Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	**Mult1**	Integer							

→ **Issue MUL.D**

Dependency Graph For Example Code

Cycle 7



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) (2, 3) (2, 4)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 7

End of

Issue →

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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				

Functional unit status

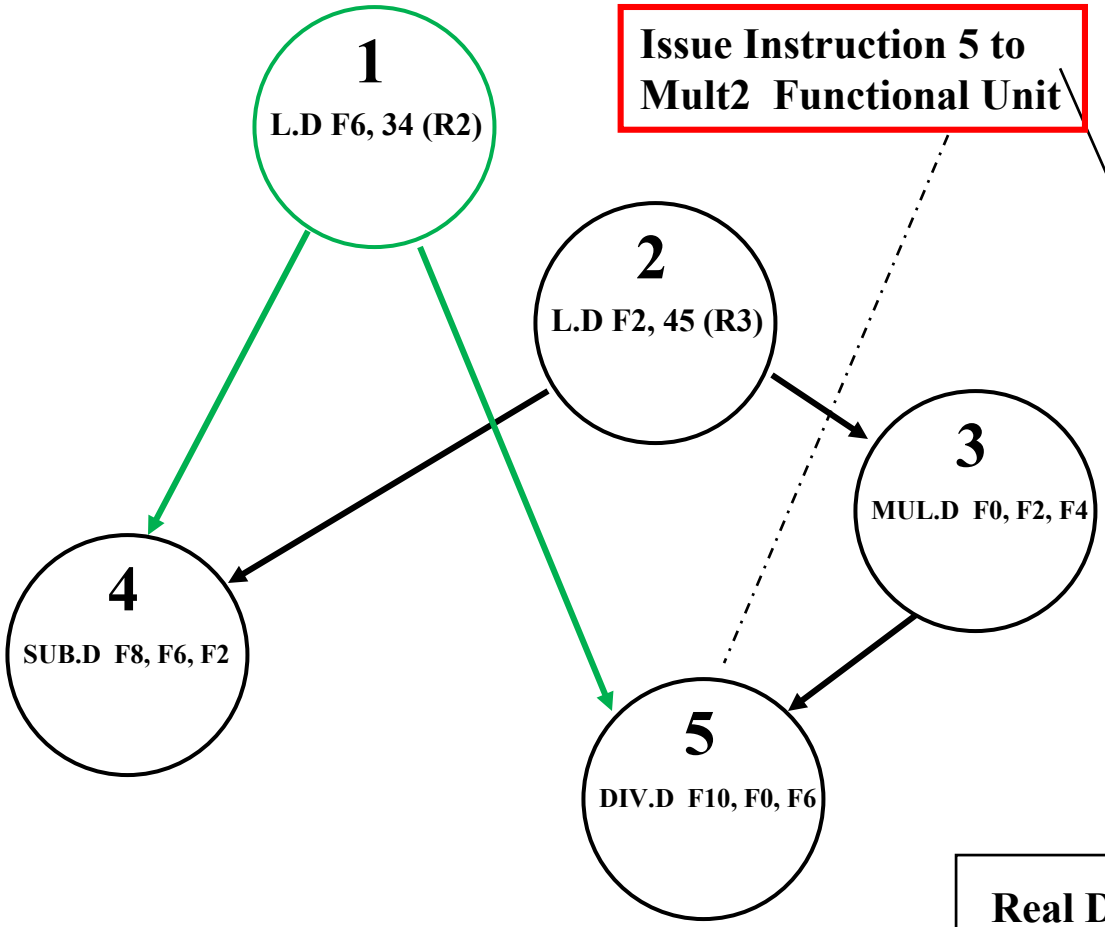
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
→	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

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

- Issue SUB.D
- Read multiply operands?

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)
(3, 5)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

Cycle 8a (First half of cycle 8)

ID1 ⟶ ID2 ⟶ EX ⟶ WB

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

Instruction status

Instruction		j	k	Issue	Read operands	Execution 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	8			
ADD.D	F6	F8	F2				

Issue →

Functional unit status

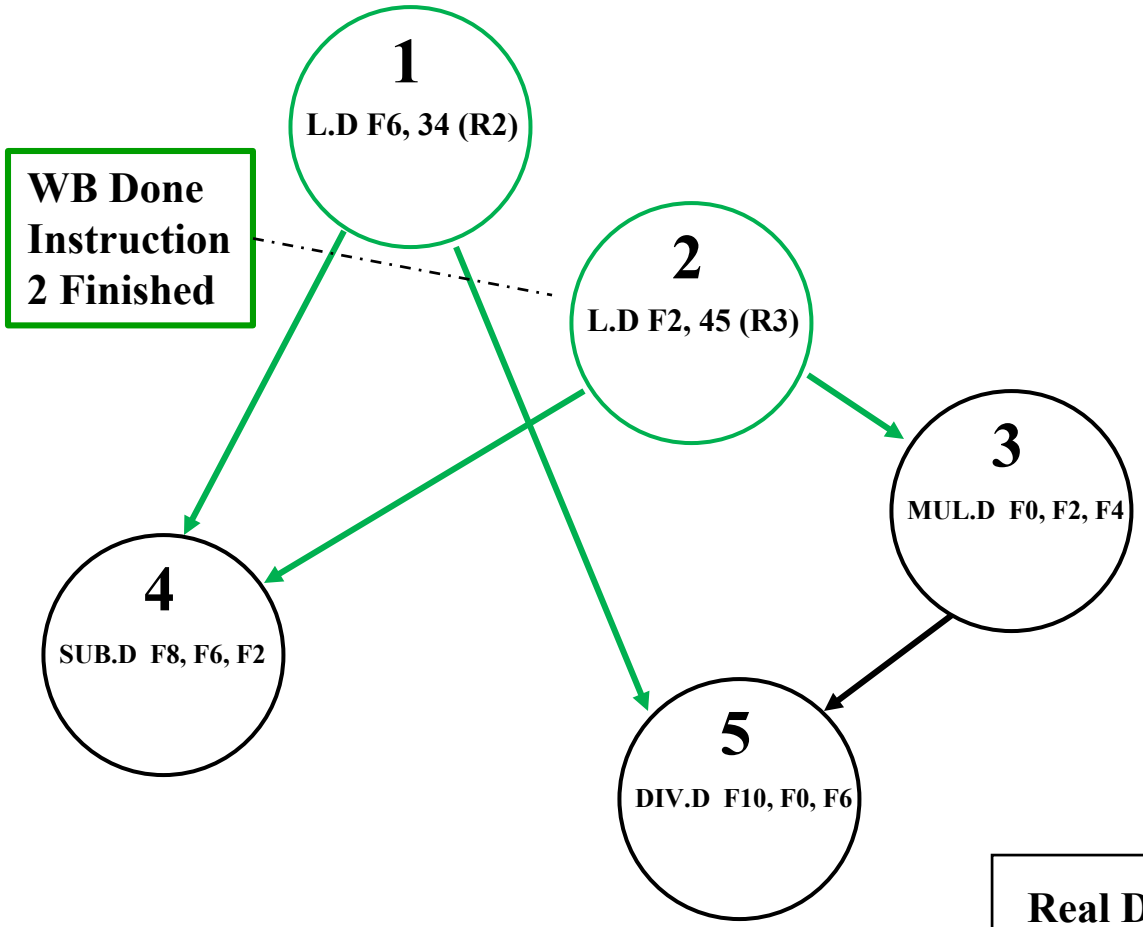
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
→	Divide	**Yes**	**Div**	**F10**	**F0**	**F6**	**Mult1**		**No**	**Yes**

Register result status

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

➔ • **Issue DIV.D**

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)
(3, 5)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

Cycle 8b (Second half of cycle 8)

ID1 → ID2 → EX → WB

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

End of Cycle 8

Instruction status

Instruction		j	k	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			
SUB.D	F8	F6	F2	7			
DIV.D	F10	F0	F6	8			
ADD.D	F6	F8	F2				

Functional unit status

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

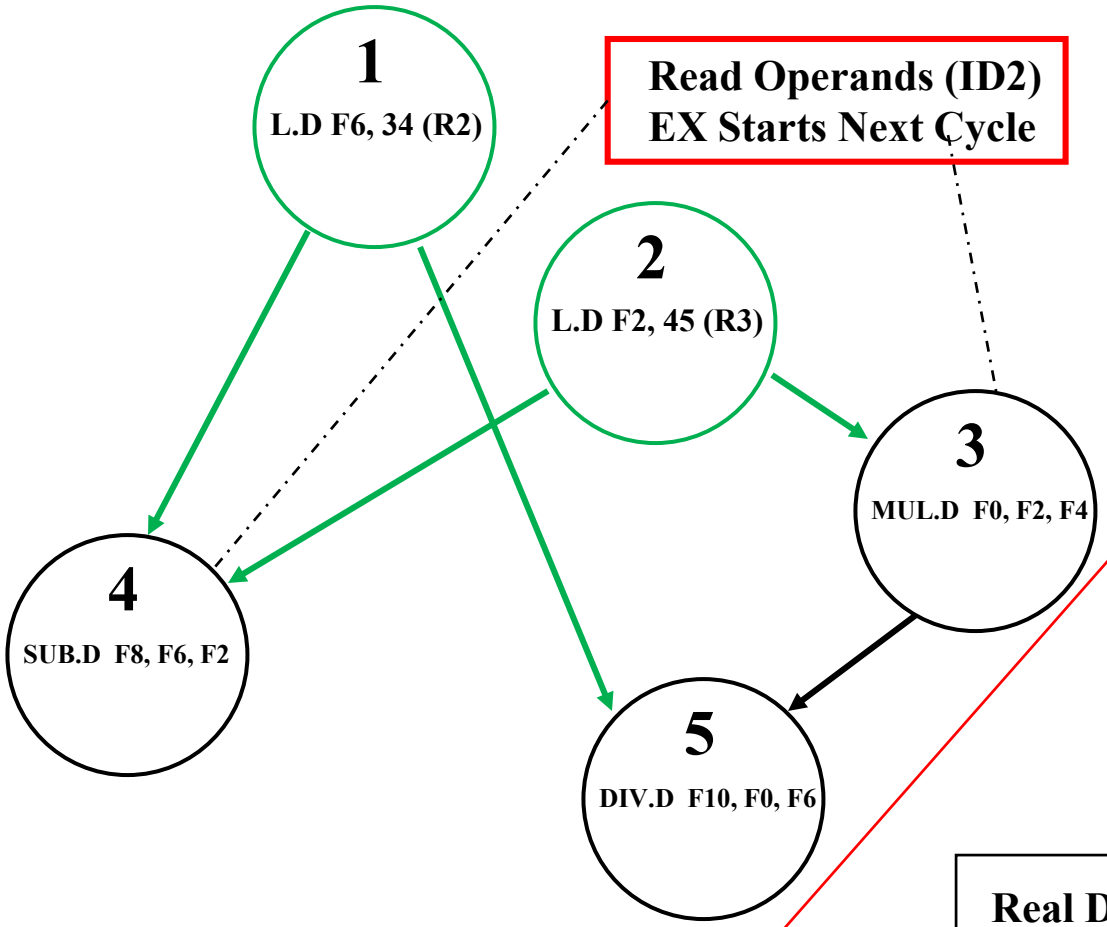
Register result status

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

- Second L.D writes result to F2

Dependency Graph For Example Code

Cycle 9



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)
(3, 5)

Output Dependence:

Anti-dependence:

Issue ADD.D?
No, stall on structural hazard.
Single Add functional unit is busy.

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

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 9

End of

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

Instruction status

Instruction		j	k	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 ? → ADD.D

Functional unit status

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

Execution cycles Remaining (execution actually starts next cycle)

Register result status

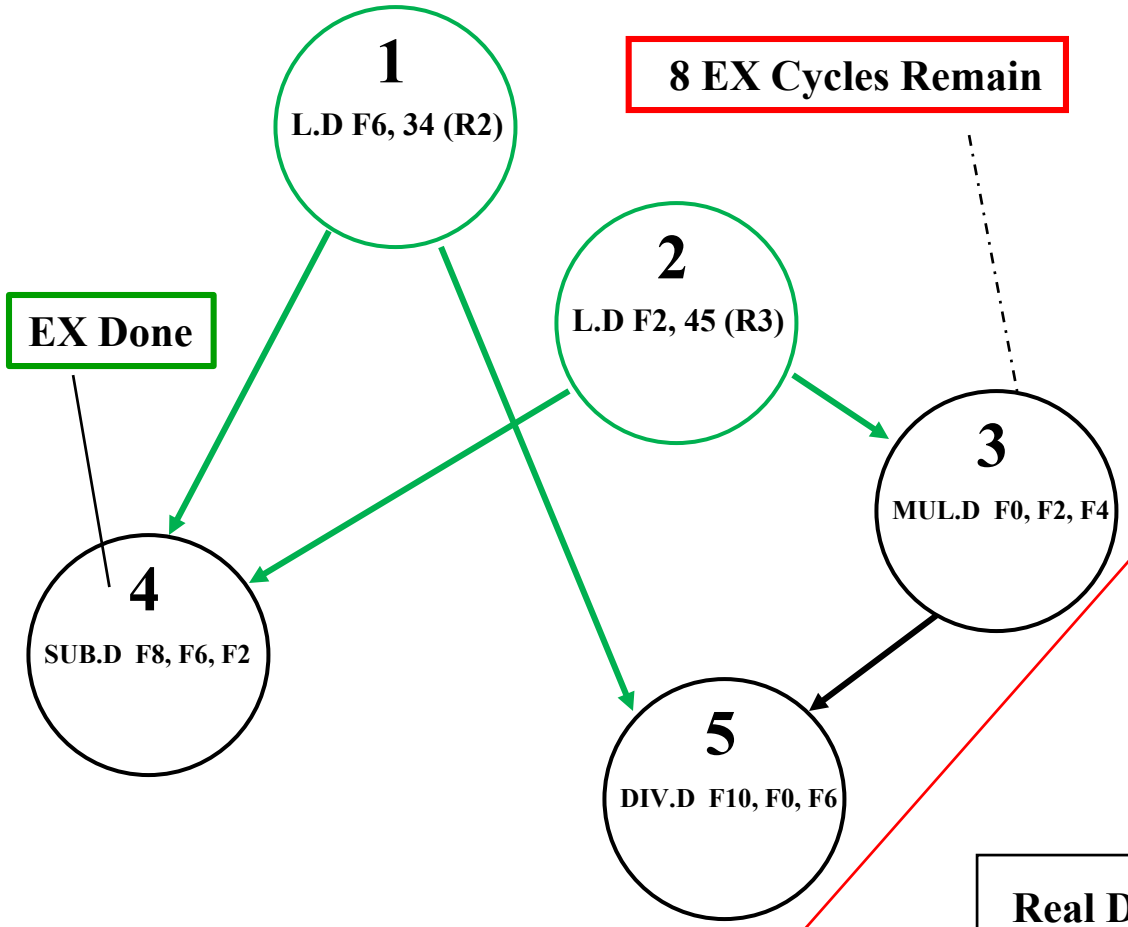
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1				Add	Divide			

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

Ex starts next cycle for both instructions

Dependency Graph For Example Code

Cycle 11



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)
(3, 5)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

Issue ADD.D?
No, stall on structural hazard.
Single Add functional unit is still busy.

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 11

End of

Instruction status

Instruction		j	k	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	
DIV.D	F10	F0	F6	8			
ADD.D	F6	F8	F2	?			

Issue ? → ADD.D

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? 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

Execution cycles Remaining

Done executing

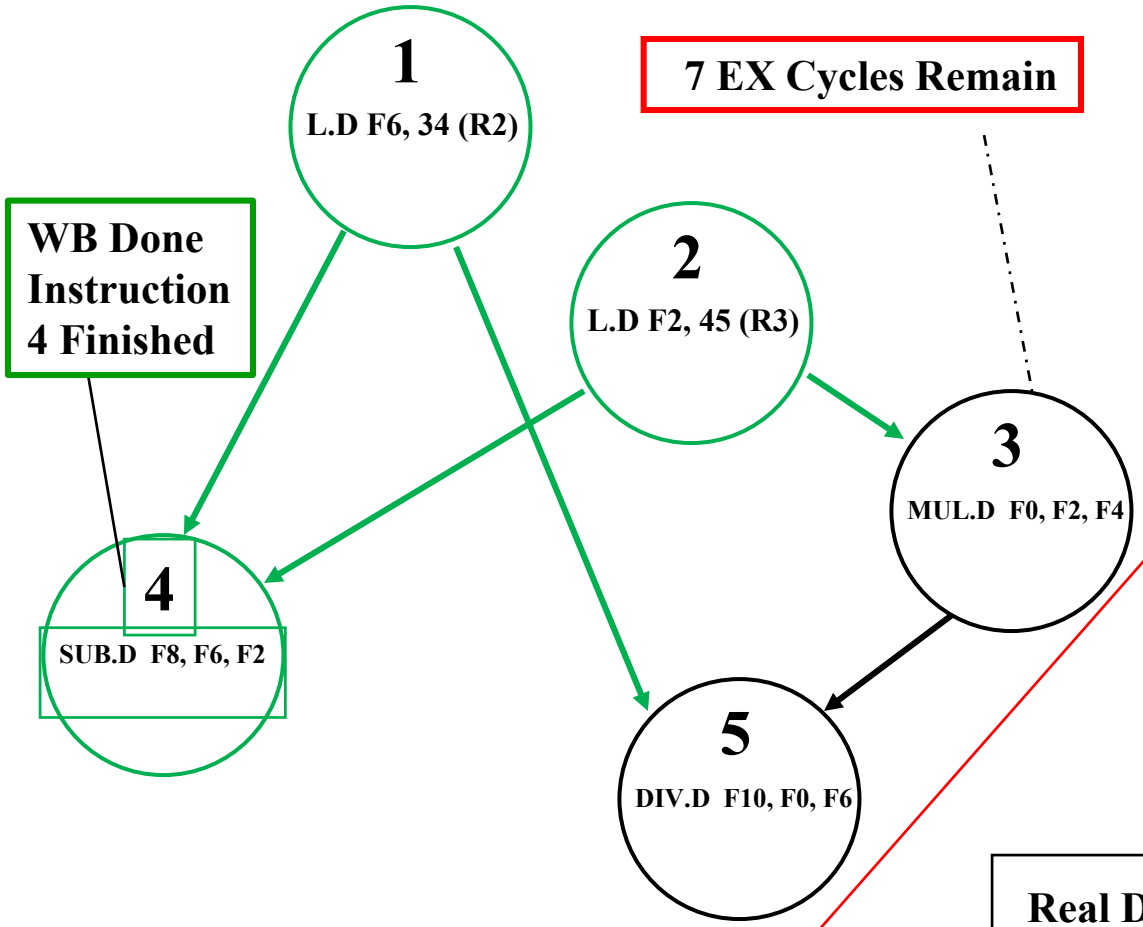
Register result status

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

? • Issue ADD.D?

Dependency Graph For Example Code

Cycle 12



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)
(3, 5)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

Issue ADD.D? Next Cycle
No, stall on structural hazard.
Single Add functional unit is still busy.

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 12

End of

Instruction status

Instruction		j	k	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

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k 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

Execution cycles Remaining

Actually free end of this cycle 12

Register result status

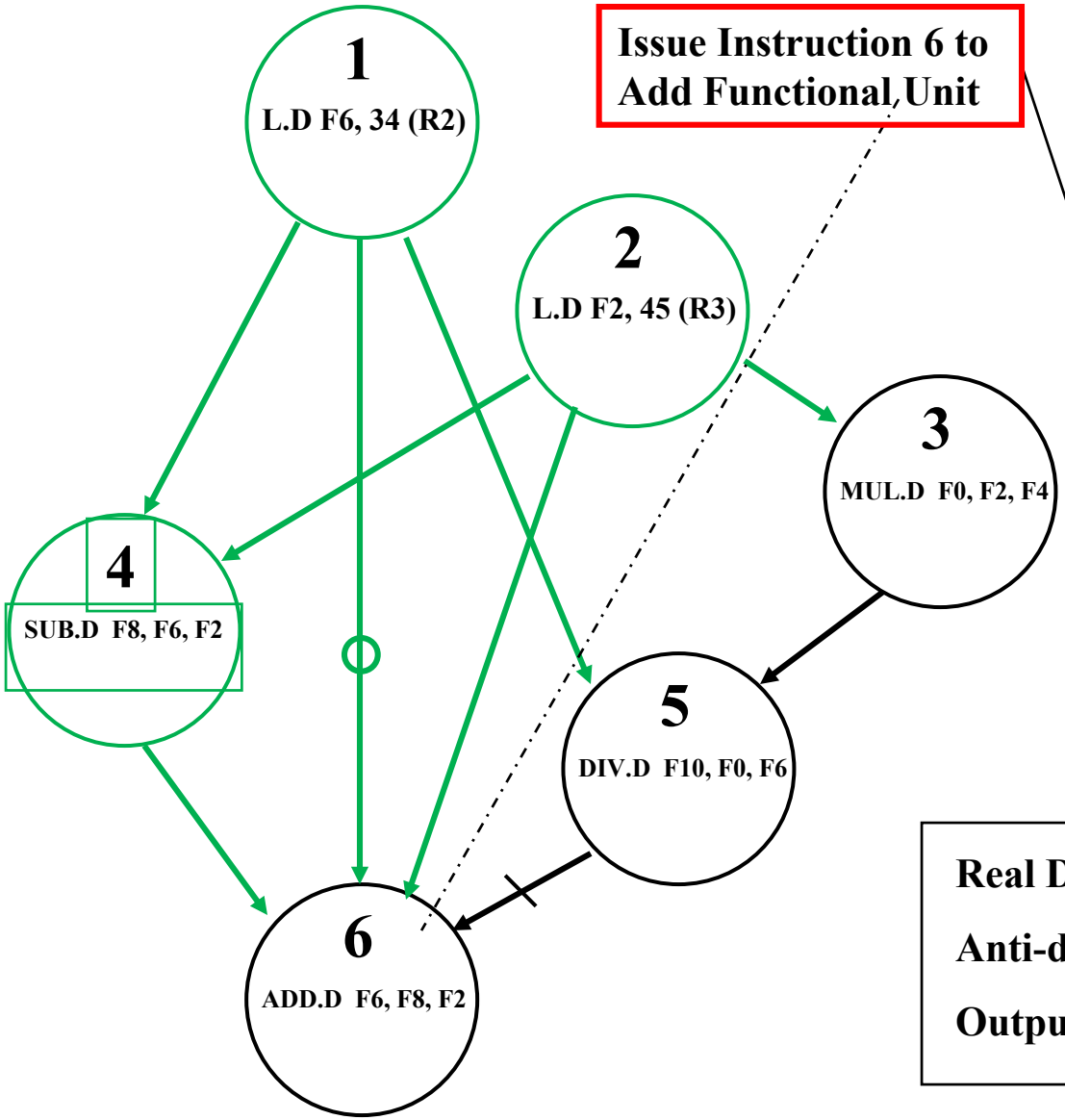
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Divide			

➔ ? • Read operands for DIV.D?

➔ ? • Issue ADD.D? Next Cycle →

Dependency Graph For Example Code

Cycle 13



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)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 13

End of

Instruction status

Instruction		j	k	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	13			

Issue →

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k 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

Execution cycles Remaining

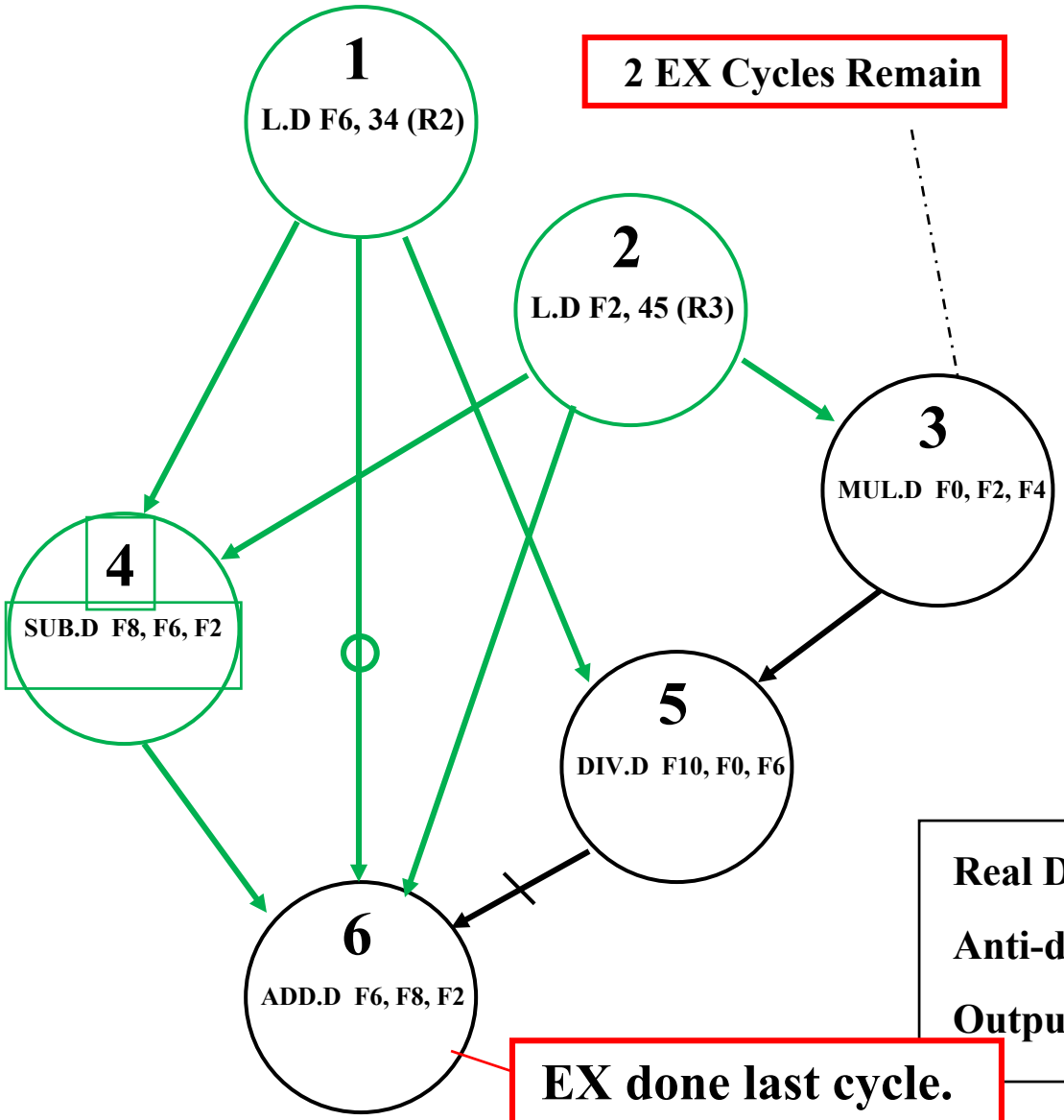
Register result status

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

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

Dependency Graph For Example Code

Cycle 17



2 EX Cycles Remain

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)

EX done last cycle.
WB in this cycle?

Scoreboard Example: Cycle 17

End of

Instruction status

Instruction		j	k	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	13	14	16	

Write result of ADD.D?
No WAR hazard

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k 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

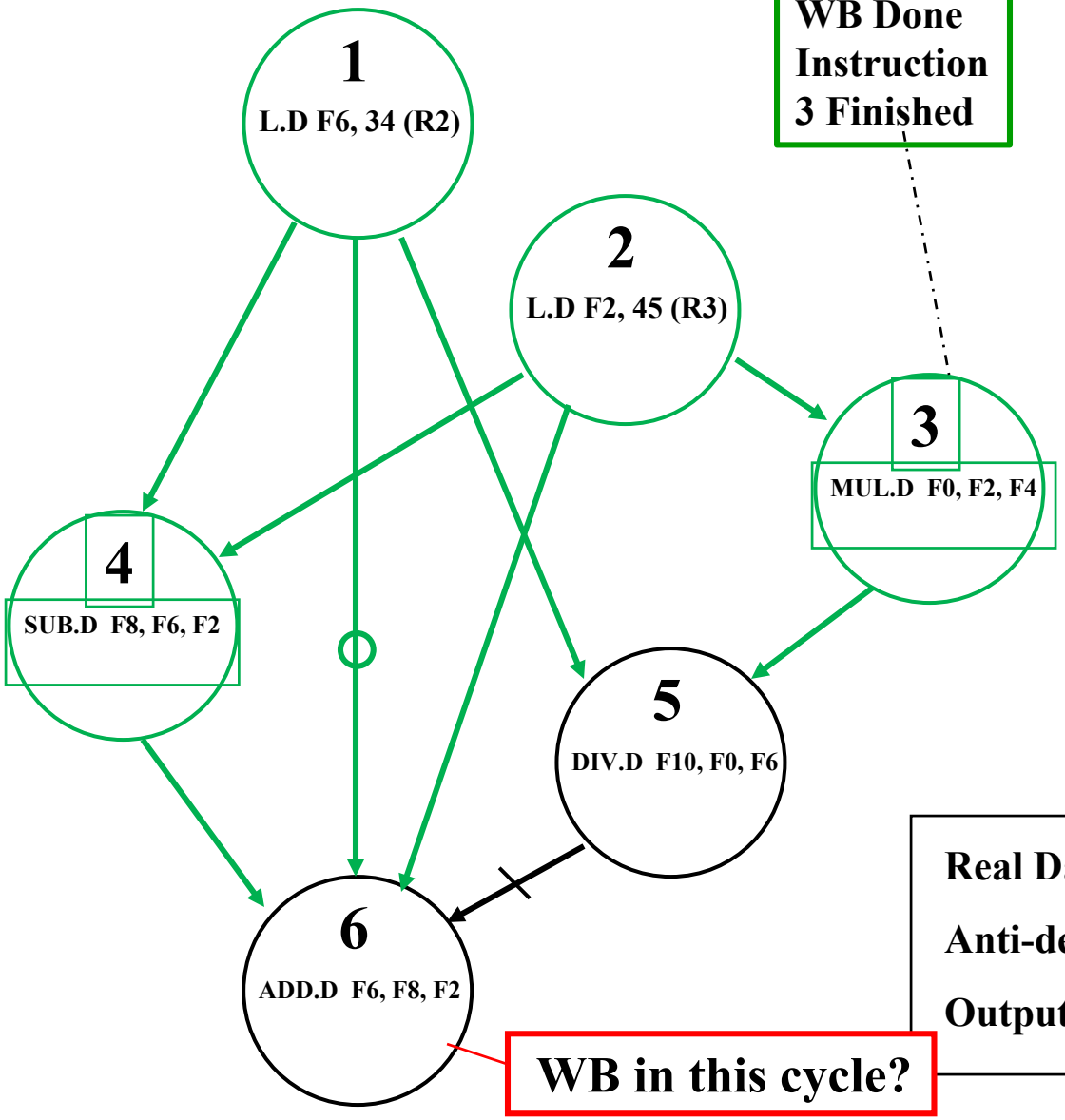
Register result status

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

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

Dependency Graph For Example Code

Cycle 20



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)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 20

End of

Instruction status

Instruction		j	k	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	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

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? 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

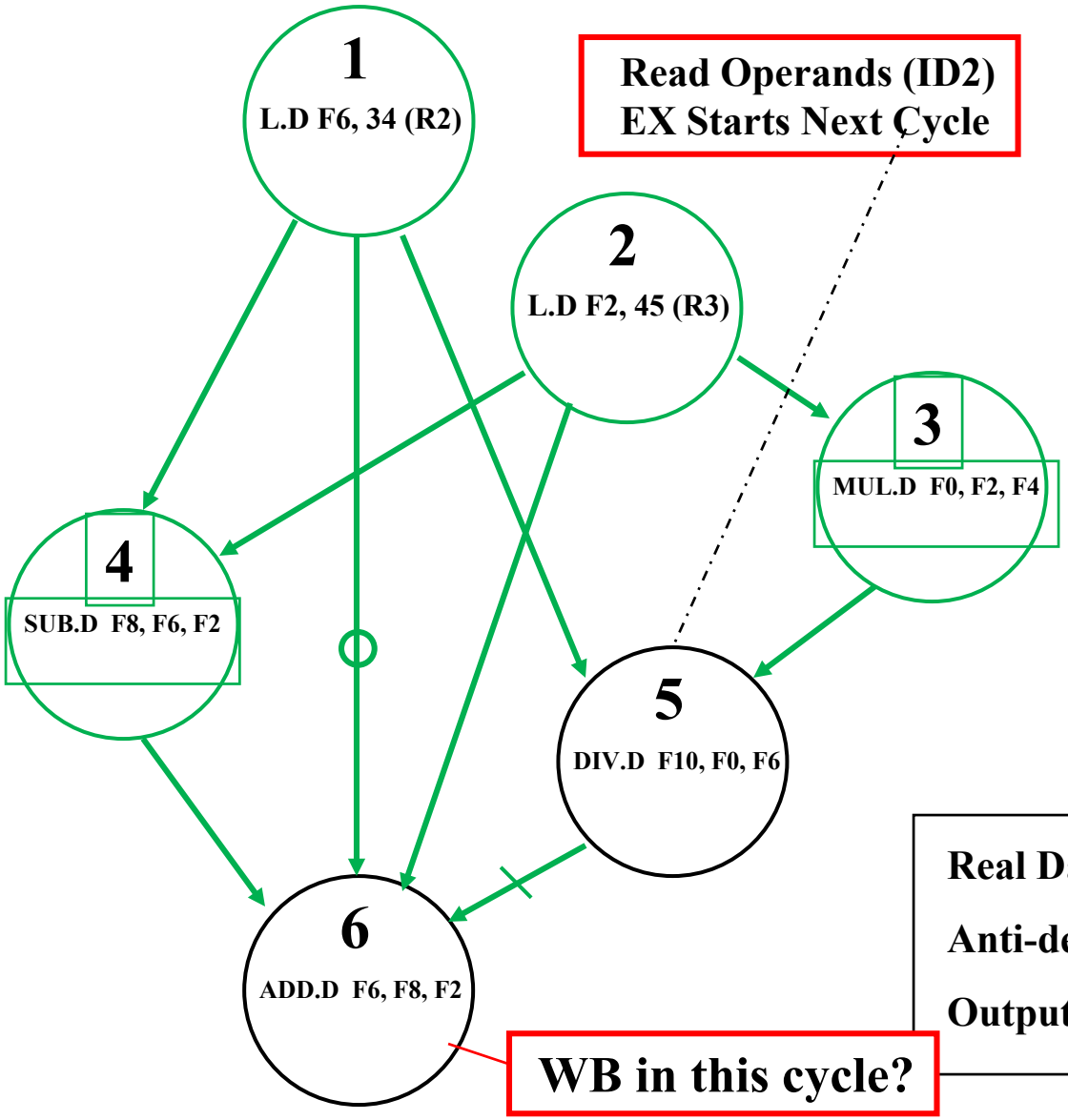
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
20	*FU*				Add		Divide			

→ ? • Read operands for DIV.D? Next Cycle →

→ ? • Write result of ADD.D?

Dependency Graph For Example Code

Cycle 21



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)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 21

End of

Instruction status

Instruction		j	k	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	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

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

Execution cycles Remaining (execution actually starts next cycle)

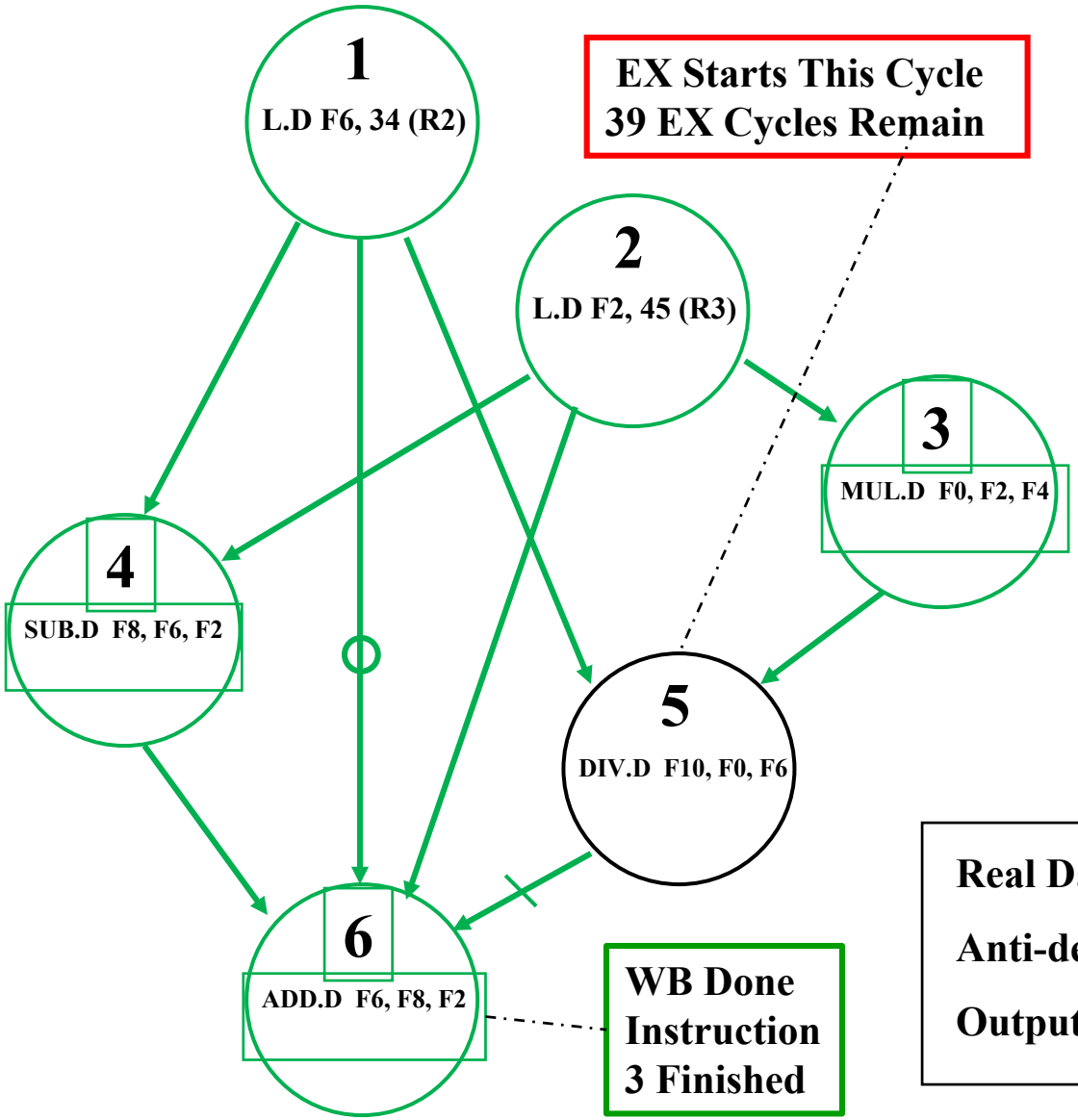
Register result status

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

- **DIV.D reads operands, starts execution next cycle**
- **Write result of ADD.D? Next Cycle →**

Dependency Graph For Example Code

Cycle 22



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) →

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 22

End of

Instruction status

Instruction		j	k	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	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**

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? 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			

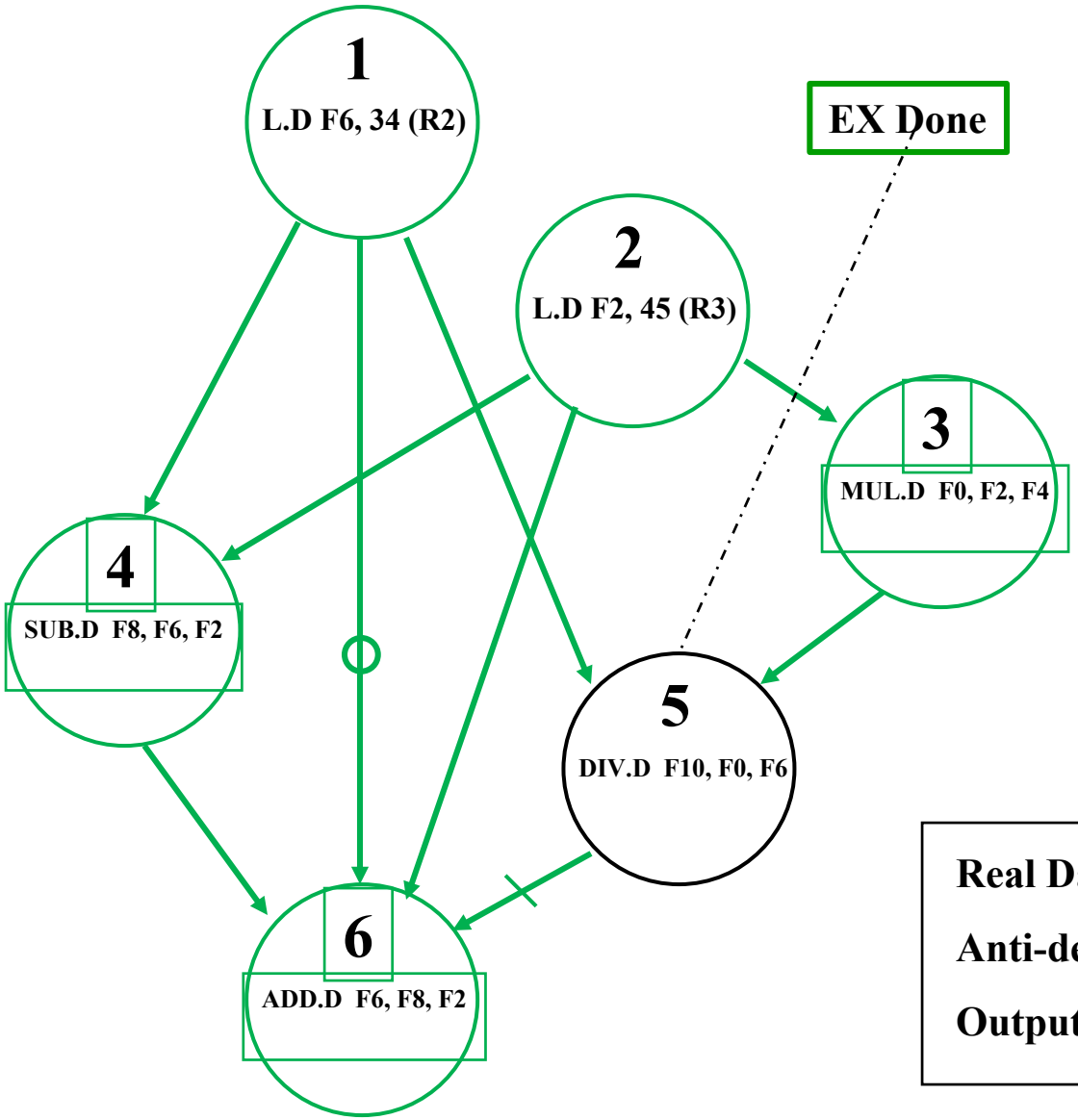
➔ First cycle DIV.D execution (39 more ex cycles)

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

(Possible)

Dependency Graph For Example Code

Cycle 61



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)

ID1 → ID2 → EX → WB

Scoreboard Example: Cycle 61

End of

Instruction status

Instruction		j	k	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	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

Functional unit status

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

Done executing

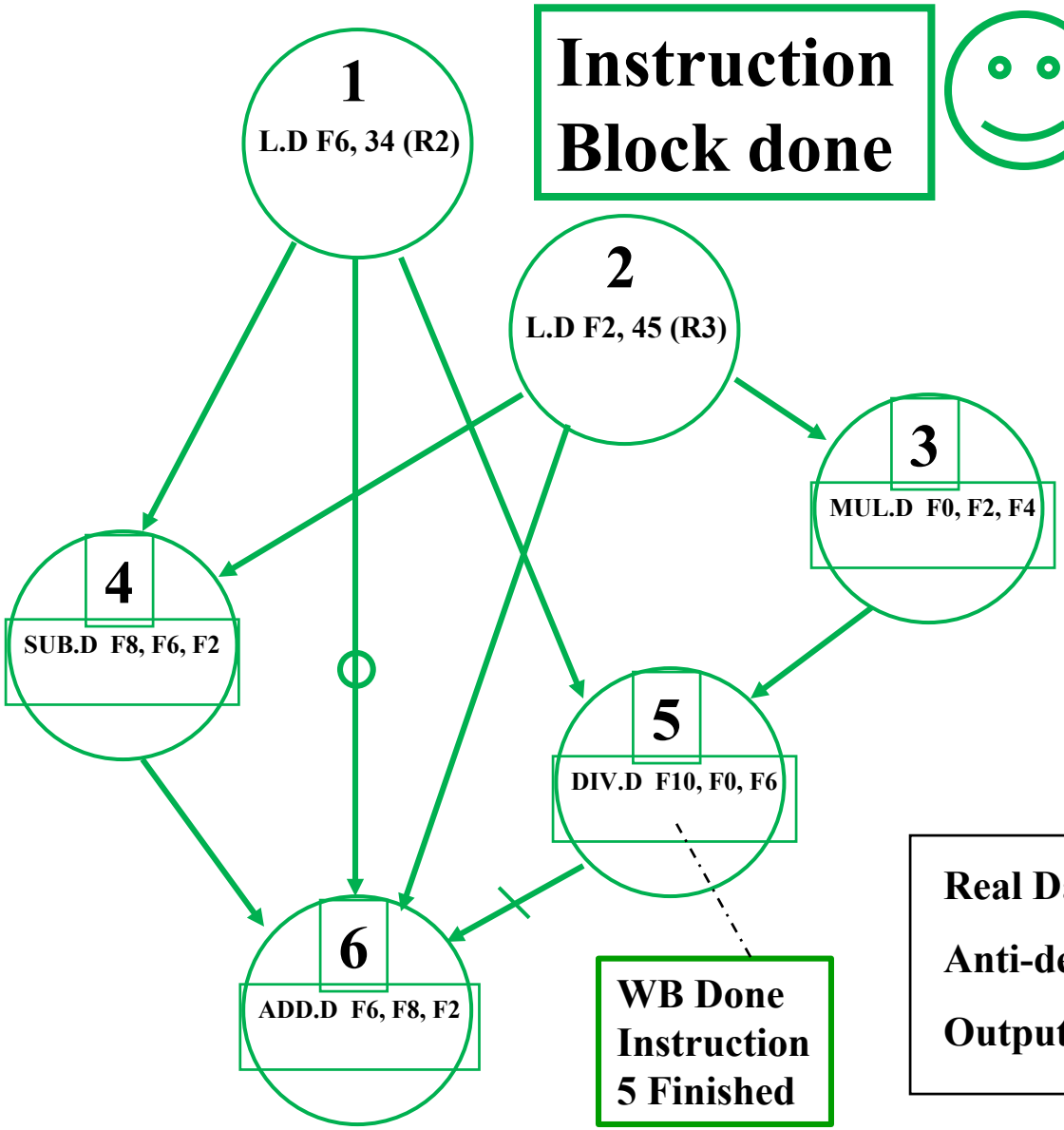
Register result status

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

➔ • DIV.D done executing

Dependency Graph For Example Code

Cycle 62



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)

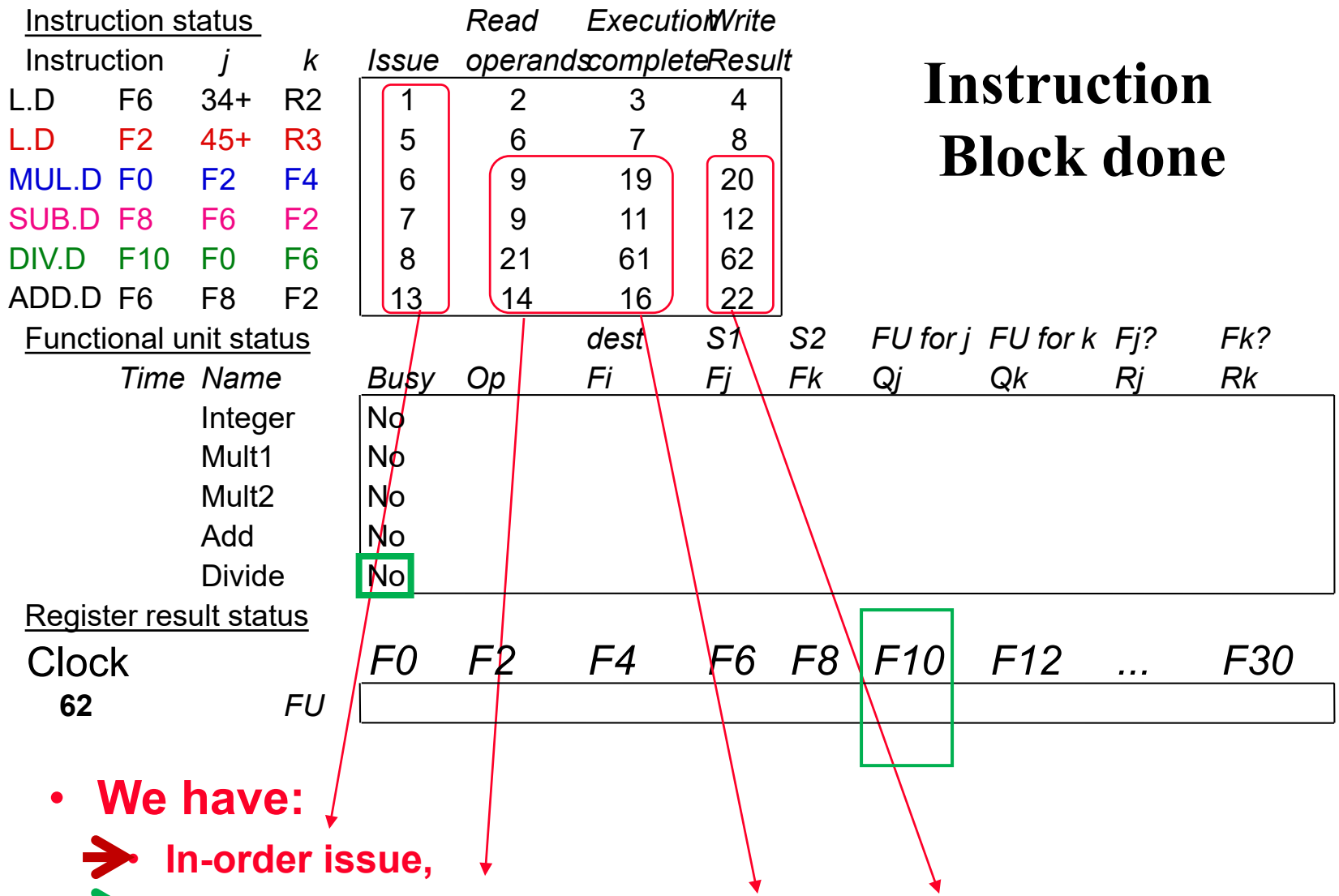
Anti-dependence:
(5,6)

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

Scoreboard Example: Cycle 62



Instruction Block done

- **We have:**
  - **In-order issue,**
  - **Out-of-order read operands, 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).
- Some 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)

A Tomasulo simulator: http://www.ecs.umass.edu/ece/koren/architecture/Tomasulo1/tomasulo.htm

Tomasulo Algorithm Vs. Scoreboard

- Control & buffers *distributed* with Functional Units (FUs) Vs. centralized in Scoreboard:
  - FU buffers are called "*reservation stations*" (RS) 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 ISA registers specified by the ISA.
- ISA Registers in instructions are replaced by either data values (if available) or pointers (renamed) to reservation stations (RS) that will supply the value later (i.e. Operands):

1 Register Renaming

  - 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.

2 Data Forwarding

- Instruction results go (forwarded) from RSs to RSs , *not through registers*, over a broadcast bus, *Common Data Bus (CDB)* that broadcasts results to all waiting RSs (dependant instructions).
- Loads and Stores are treated as FUs with RSs as well.

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

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 Reservation Stations (RSs) have a single **V** field indicating result to be stored.

Used/Needed To Track Data Dependencies

- **Qj, Qk** Reservation stations (if any) producing source registers. (value to be broadcast). (i.e. operand values needed by instruction)
  - No ready flags as in Scoreboard; Qj,Qk=0 => ready.
  - Store buffers only have **Qi** for RS producing result. — to be stored
    - i.e. Store Reservation Stations
    - One
- **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.

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

Register bank behaves like a reservation station (listen to CDB for data)

Three Stages of Tomasulo Algorithm

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

1 Always done in program order

- Instruction issued to[2] 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

Using Q Fields

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

- Write result on Common Data Bus (CDB) to all awaiting units (RSs)

And also to destination register

- 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

Can be done out of program order

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.

Including destination register

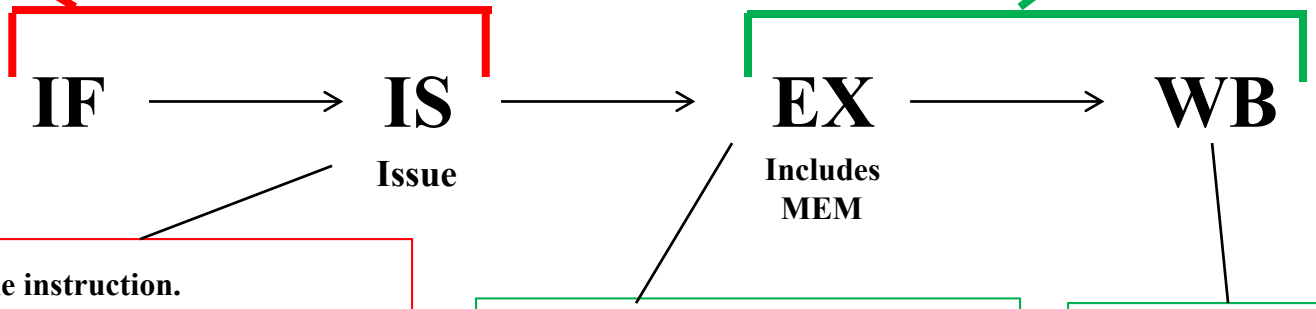
In Fourth Edition: Chapter 2.4
(In Third Edition: Chapter 3.2)

Stages of Tomasulo Algorithm

Including Instruction Fetch (IF): No changes, still in-order

In program order

Can be reached out of program order
Including Start of Execution

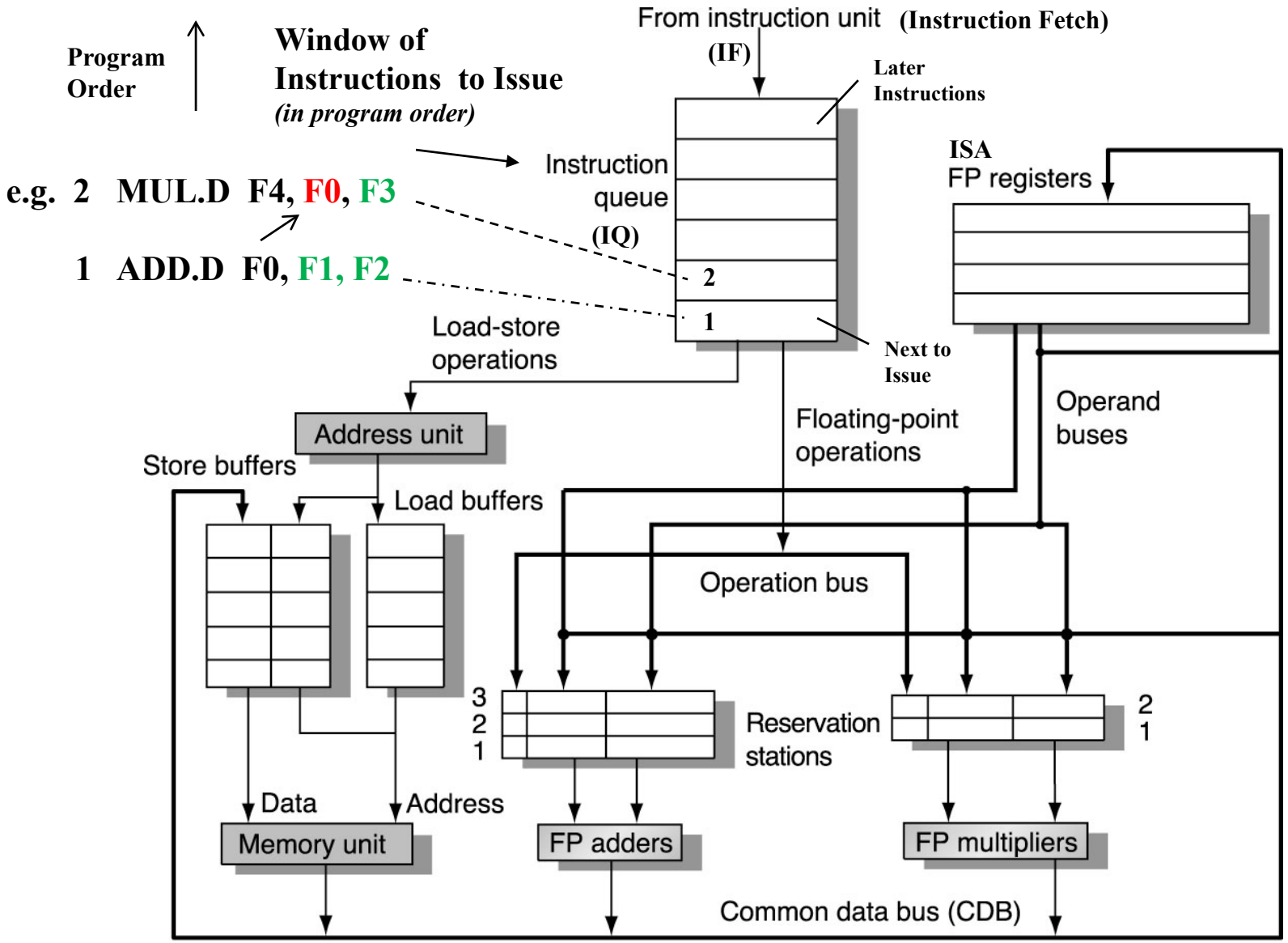


- Decode instruction.
- Instruction issued in-order to a free reservation station(RS) .
- No explicit "Read Operands" step:
  - Available instruction operands values supplied (from ISA registers) to assigned RS.
  - Operands not available yet are renamed to RSs that will produce the operand (register renaming).
  - (Dynamic construction of data dependency graph).

- Operate on operands.
- Also Includes MEM stage.
- When all 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)

- Write result on Common Data Bus (CDB) to all awaiting units (RSs) and destination register. i.e. broadcast result on CDB (data forwarding).
- No WB for stores or branches

Dynamic Scheduling: The Tomasulo Approach

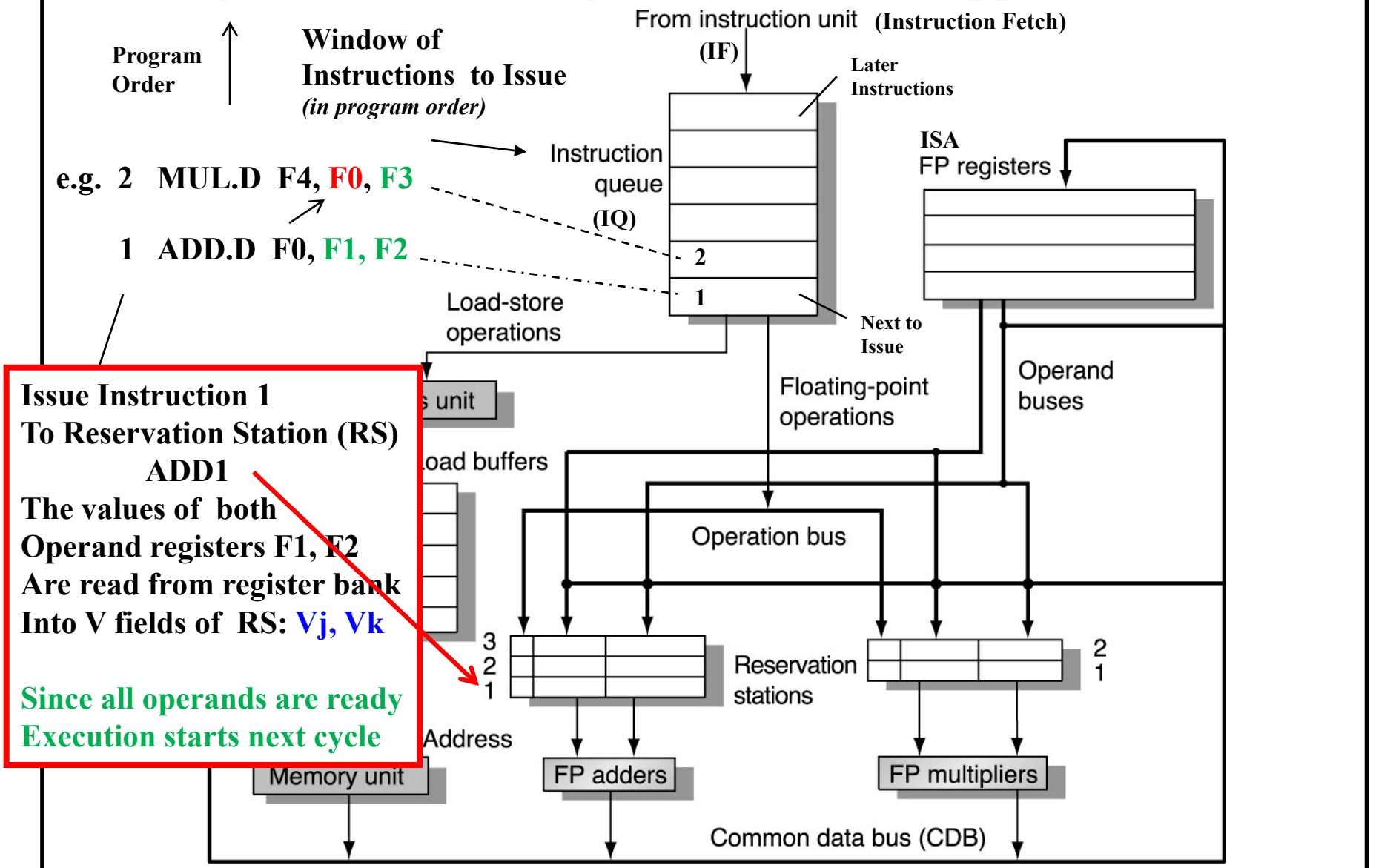


The basic structure of a MIPS floating-point unit using Tomasulo's algorithm

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

Pipelined FP units are used here

Dynamic Scheduling: The Tomasulo Approach

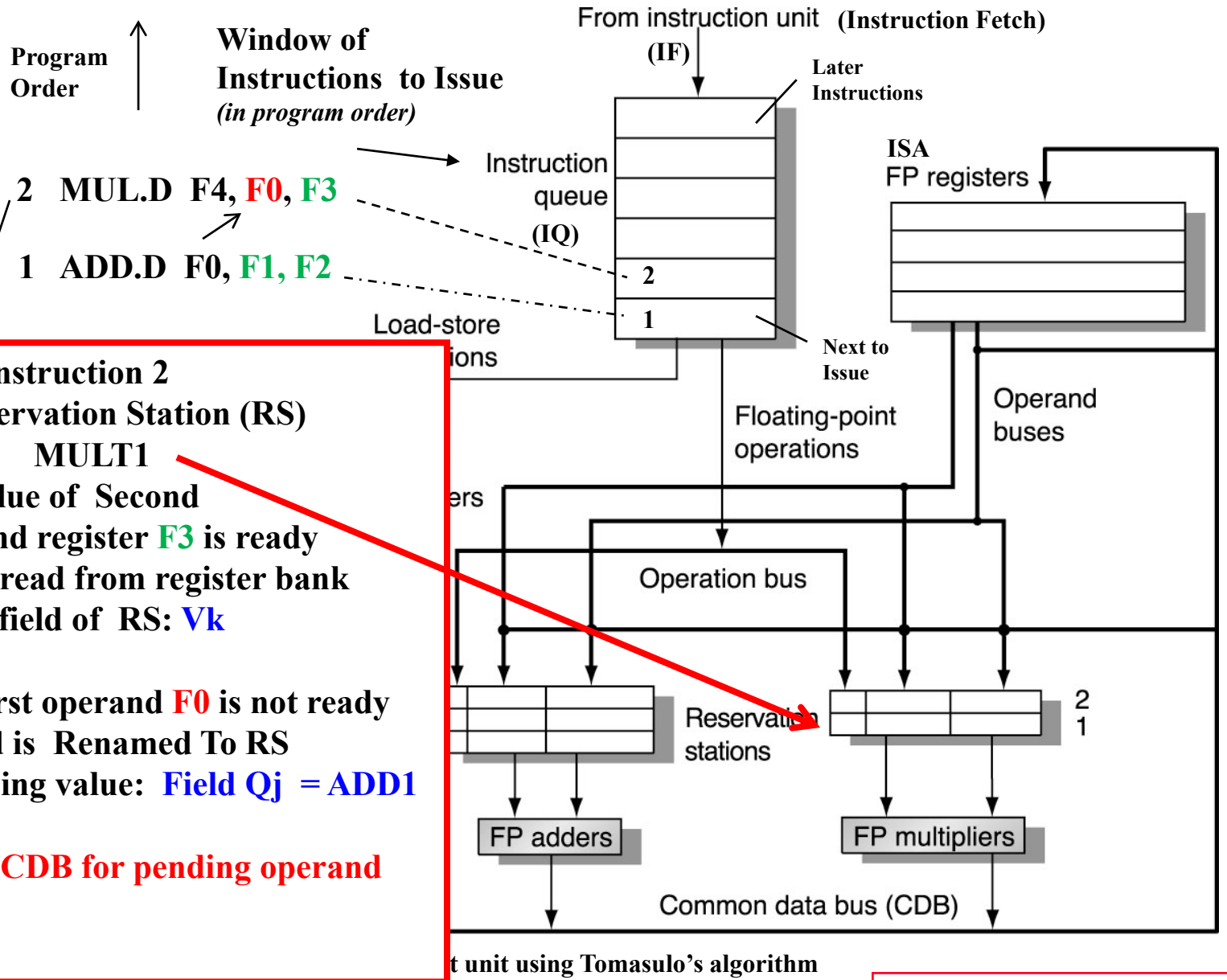


The basic structure of a MIPS floating-point unit using Tomasulo's algorithm

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

Pipelined FP units are used here

Dynamic Scheduling: The Tomasulo Approach



Issue Instruction 2
To Reservation Station (RS)
MULT1
The value of Second Operand register F3 is ready And is read from register bank Into V field of RS: Vk

The First operand F0 is not ready yet and is Renamed To RS producing value: Field Qj = ADD1

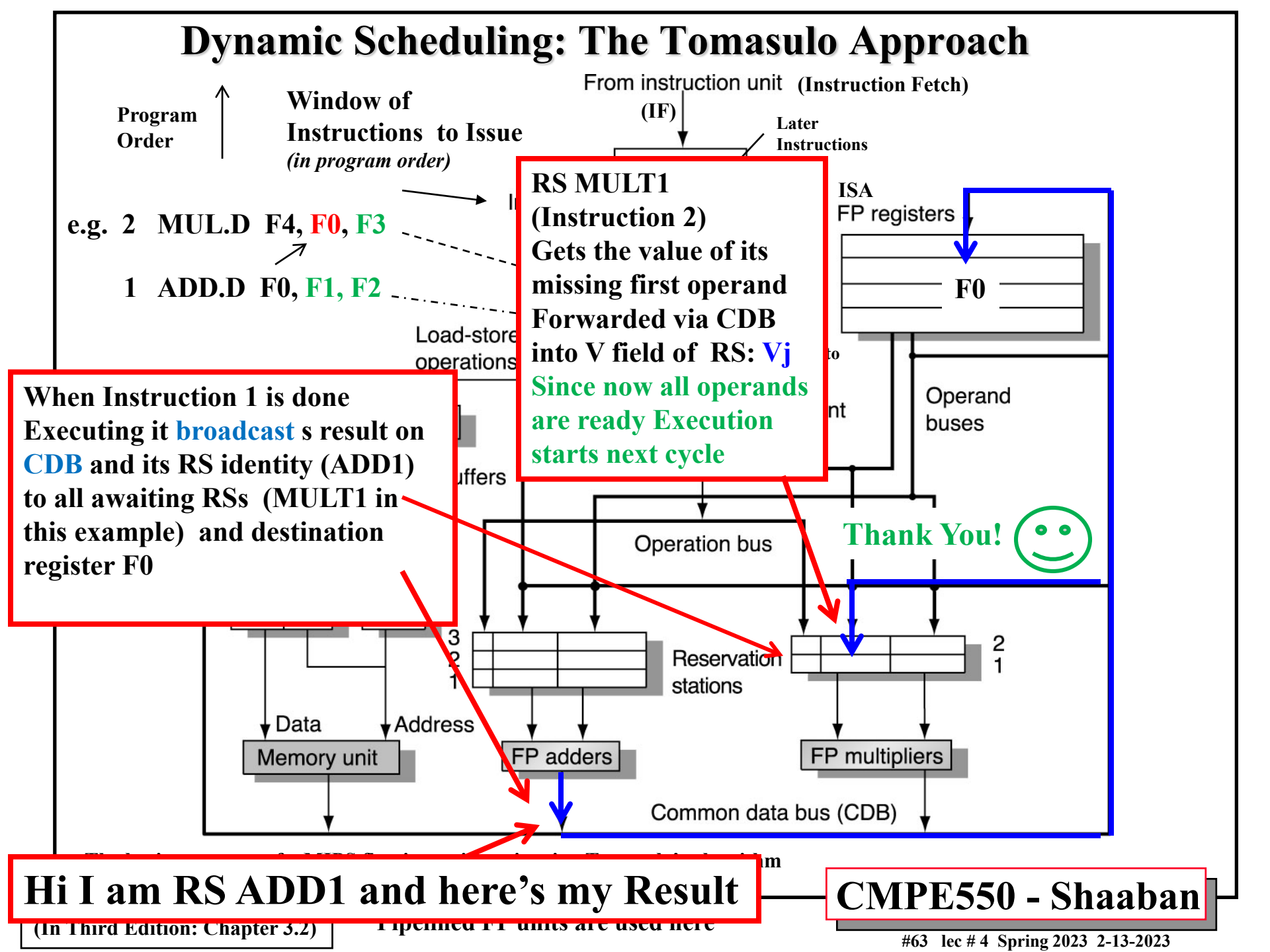
Watch CDB for pending operand

t unit using Tomasulo's algorithm

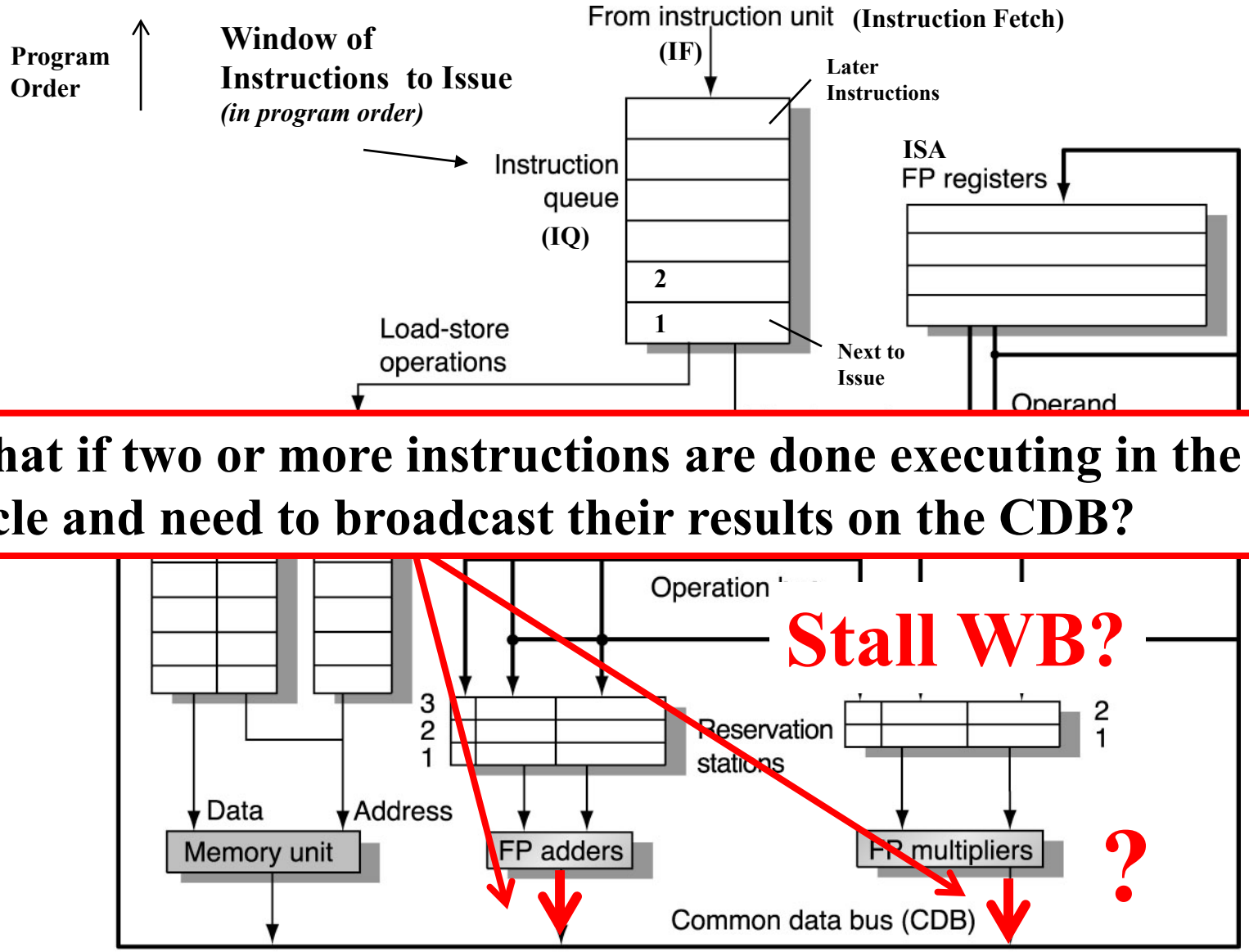
In Fourth Edition: Chapter 2.4
(In Third Edition: Chapter 3.2)

Pipelined FP units are used here

Dynamic Scheduling: The Tomasulo Approach



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

IBM 360/91 Vs. CDC 6600

Tomasulo-based (1966)

Control Data Corp.

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: ≤ 14 instructions	≤ 5 instructions
No issue on structural hazard	same
WAW: renaming avoids it	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

Eliminated By register renaming

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

Steps in The Tomsulo Approach and The Requirements of Each Step

Instruction status	Wait until	Action or bookkeeping

Issue	Station or buffer empty	`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;`

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	`∀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`

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

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 ?)

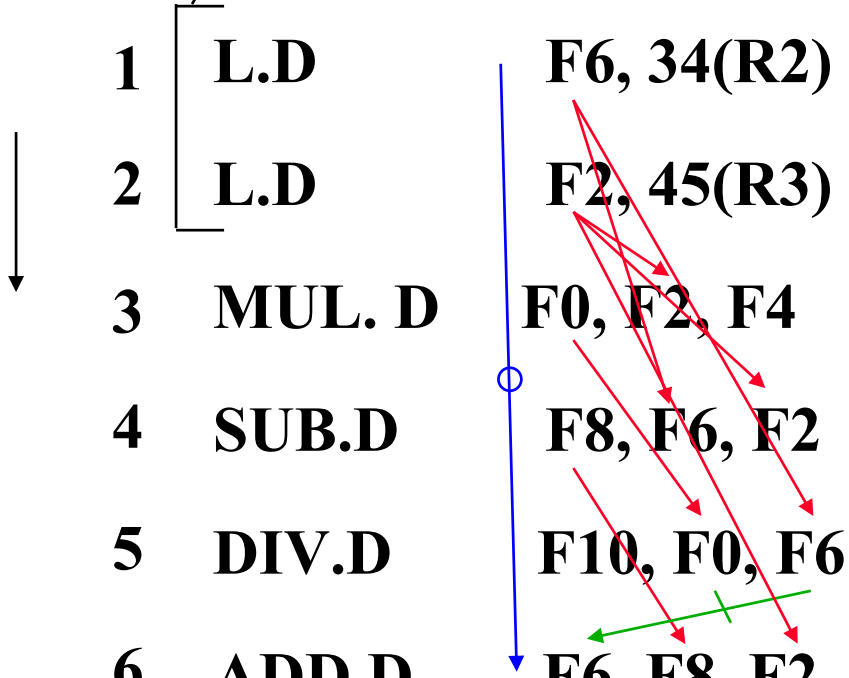
Tomasulo Approach Example

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

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



Pipelined Functional Units

Real Data Dependence (RAW)

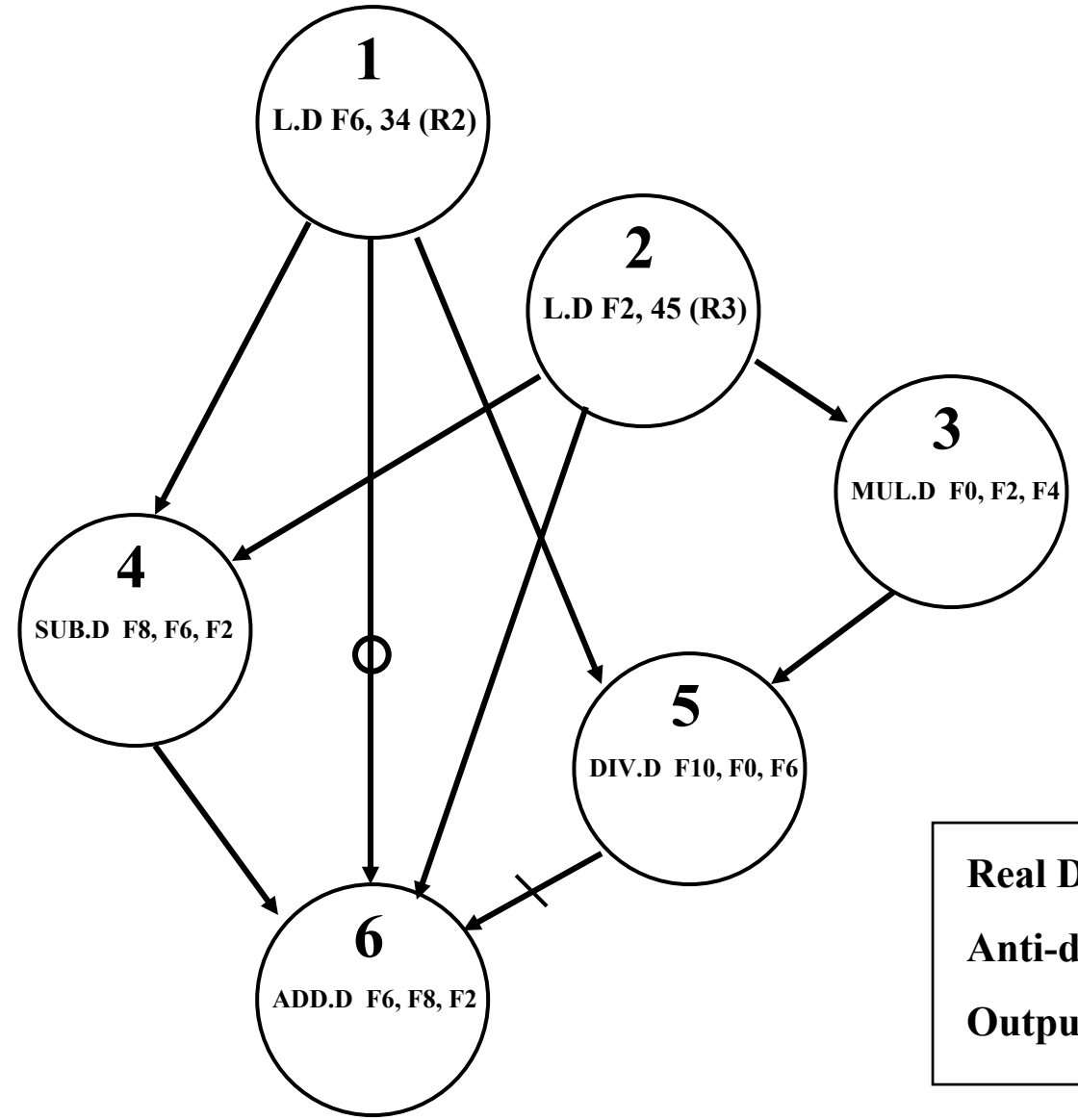
Anti-dependence (WAR)

Output Dependence (WAW)

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

In Fourth Edition: Chapter 2.5 (In Third Edition: Chapter 3.3)

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)

The same code used is the scoreboard example

Tomasulo Example: Cycle 0

(i.e at end of cycle 0)

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

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
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			

Load Reservation Stations (RSs)

/	Busy	Address
Load1	No	
Load2	No	
Load3	No	

RSs

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					
0	Mult1	No					
0	Mult2	No					

Operand Values (Data)

Producer of Result Needed

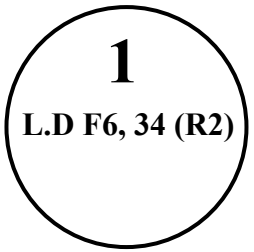
Register result status

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

IS (Issue) → **EX** (Includes MEM) → **WB**

Dependency Graph For Example Code

Cycle 1



Issue Instruction 1 to Load1 RS

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:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example Cycle 1

End of

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

Issue →

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB 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			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations

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

Register result status

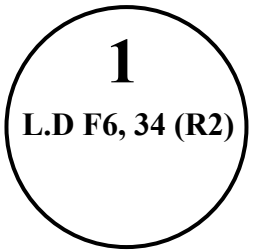
Clock 1

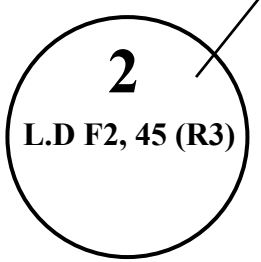
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Load1					

- Issue first load to load1 reservation station (RS)

Dependency Graph For Example Code

Cycle 2





Issue Instruction 2 to Load2 RS

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:

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 2

End of

Issue

Instruction status

Instruction	j	k	IS Issue	EX Execution complete	WB Write Result
L.D F6	34+	R2	1		
L.D F2	45+	R3	2		
MUL.D F0	F2	F4			
SUB.D F8	F6	F2			
DIV.D F10	F0	F6			
ADD.D F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations

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

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	Load2		Load1					

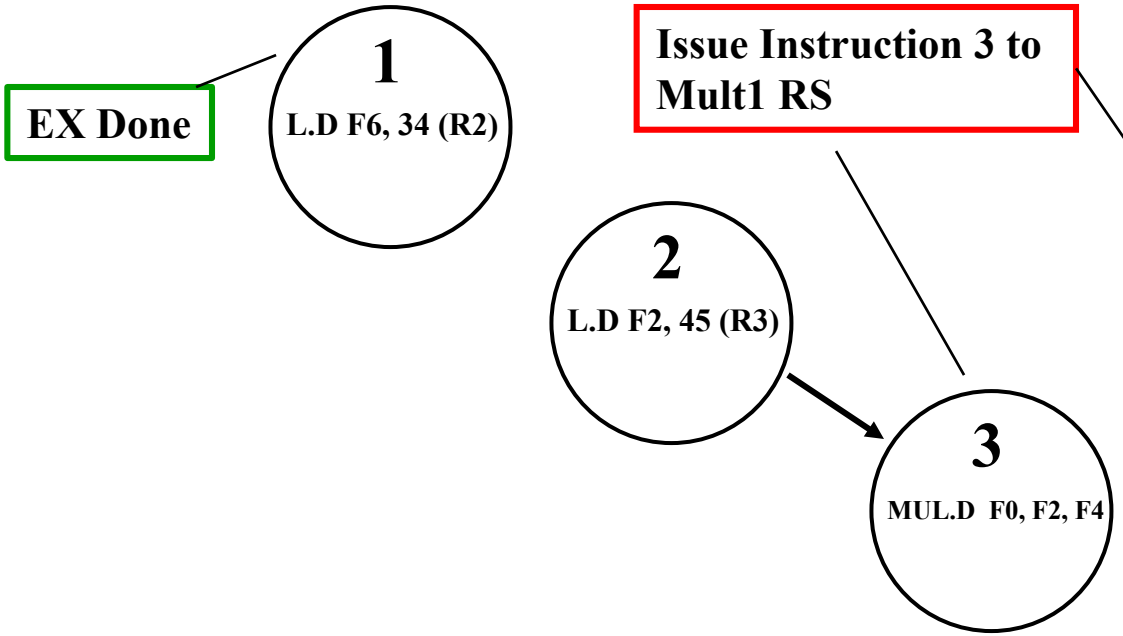
- **Issue second load to load2 reservation station**

Note: Unlike 6600, can have multiple loads outstanding

(CDC6600 only has one integer FU)

Dependency Graph For Example Code

Cycle 3



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:
(2, 3)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 3

End of

Issue →

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result
L.D	F6	34+	R2	1	3	
L.D	F2	45+	R3	2		
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2			
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Load processing takes 2 cycles (EX, Mem)

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	—	R(F4)	Load2	—
0	Mult2	No					

Register result status

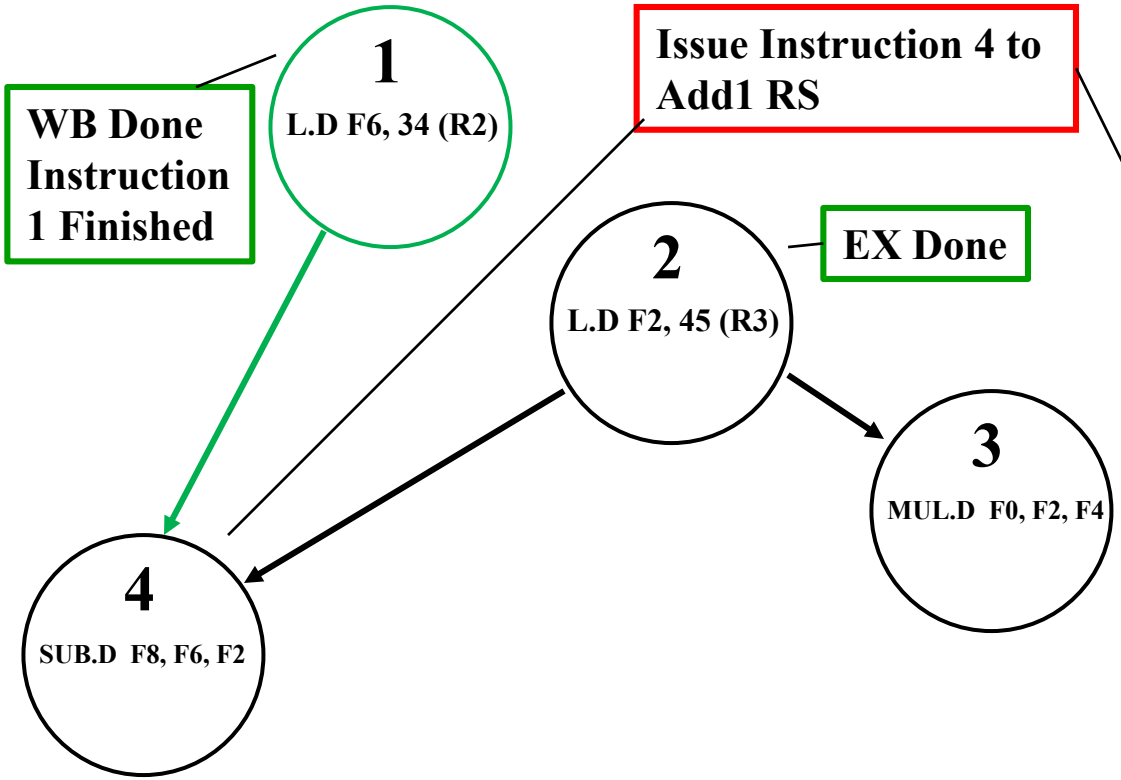
Clock 3

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Load2		Load1					

- Issue MUL.D to reservation station Mult1

Dependency Graph For Example Code

Cycle 4



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) (2, 3) (2, 4)

Output Dependence:

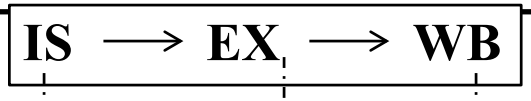
Anti-dependence:

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

The same code used is the scoreboard example



Tomasulo Example: Cycle 4

End of

WB Over CDB

Issue →

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4		
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vi	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	Yes	SUBD	M(34+R2)			Load2
0	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD		R(F4)	Load2	
0	Mult2	No					

Register result status

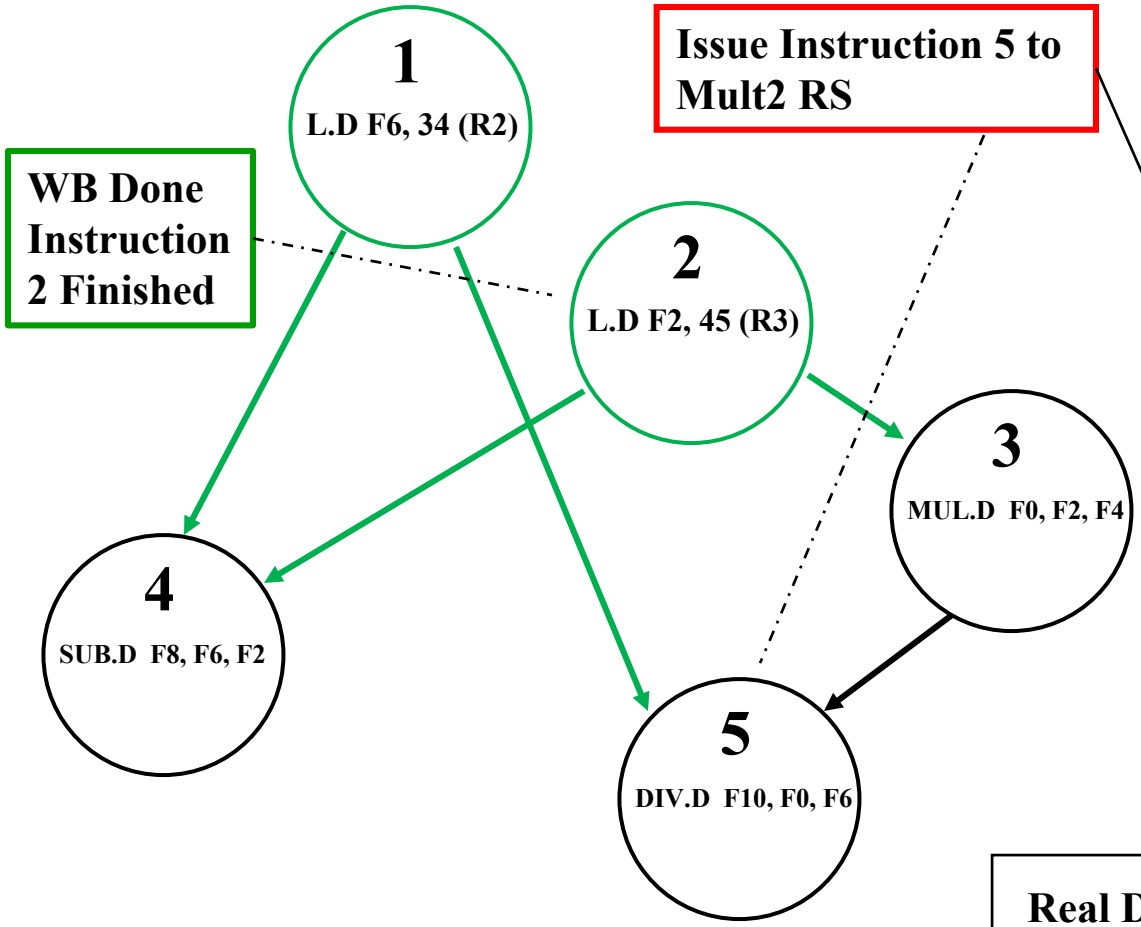
Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
4 FU	Mult1	Load2		M(34+R2)	Add1				

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

- Issue SUB.D
- Load2 completing; what is waiting for it?

Dependency Graph For Example Code

Cycle 5



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)
(3, 5)

Output Dependence:

Anti-dependence:

Real Data Dependence (RAW) →

Anti-dependence (WAR) →

Output Dependence (WAW) →

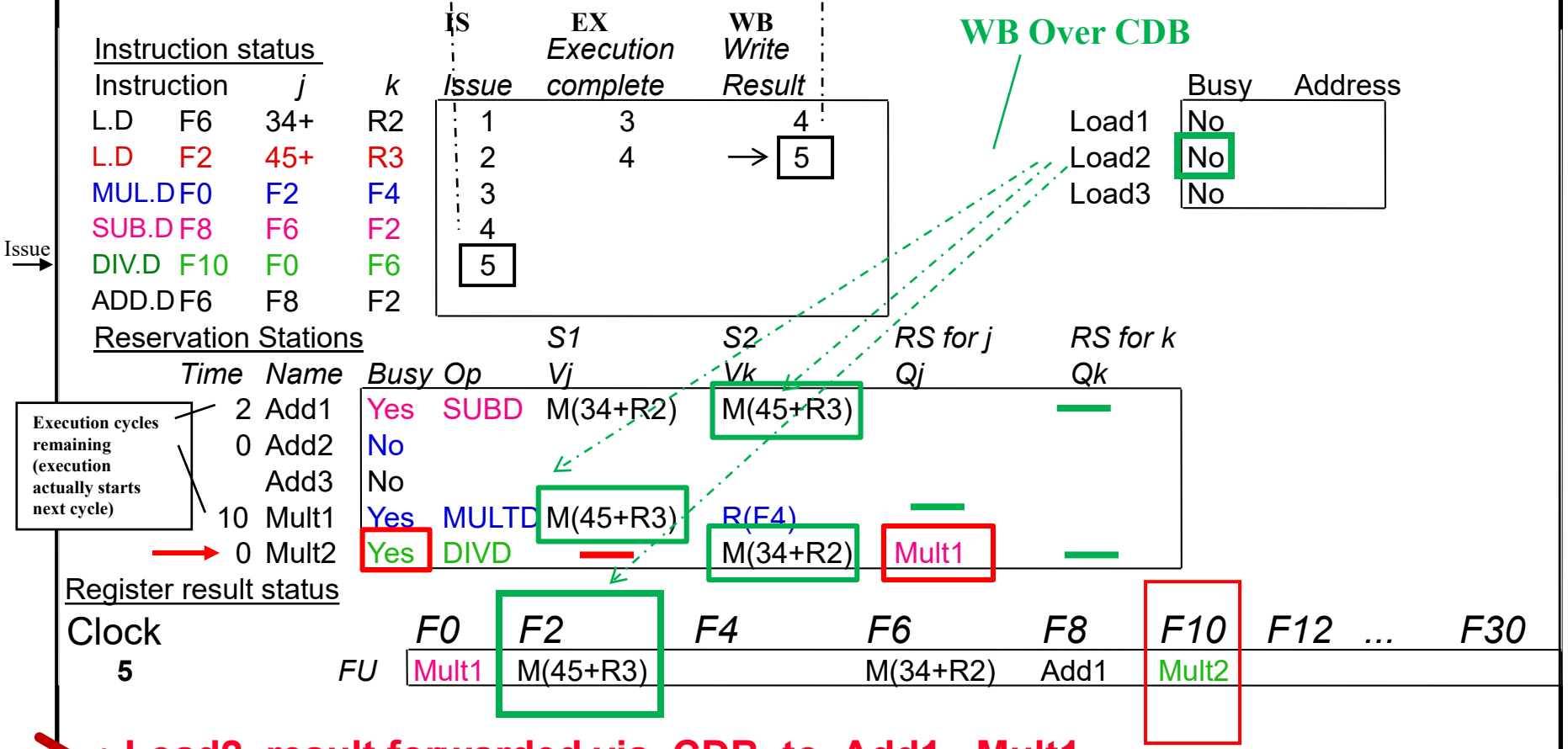
The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 5

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

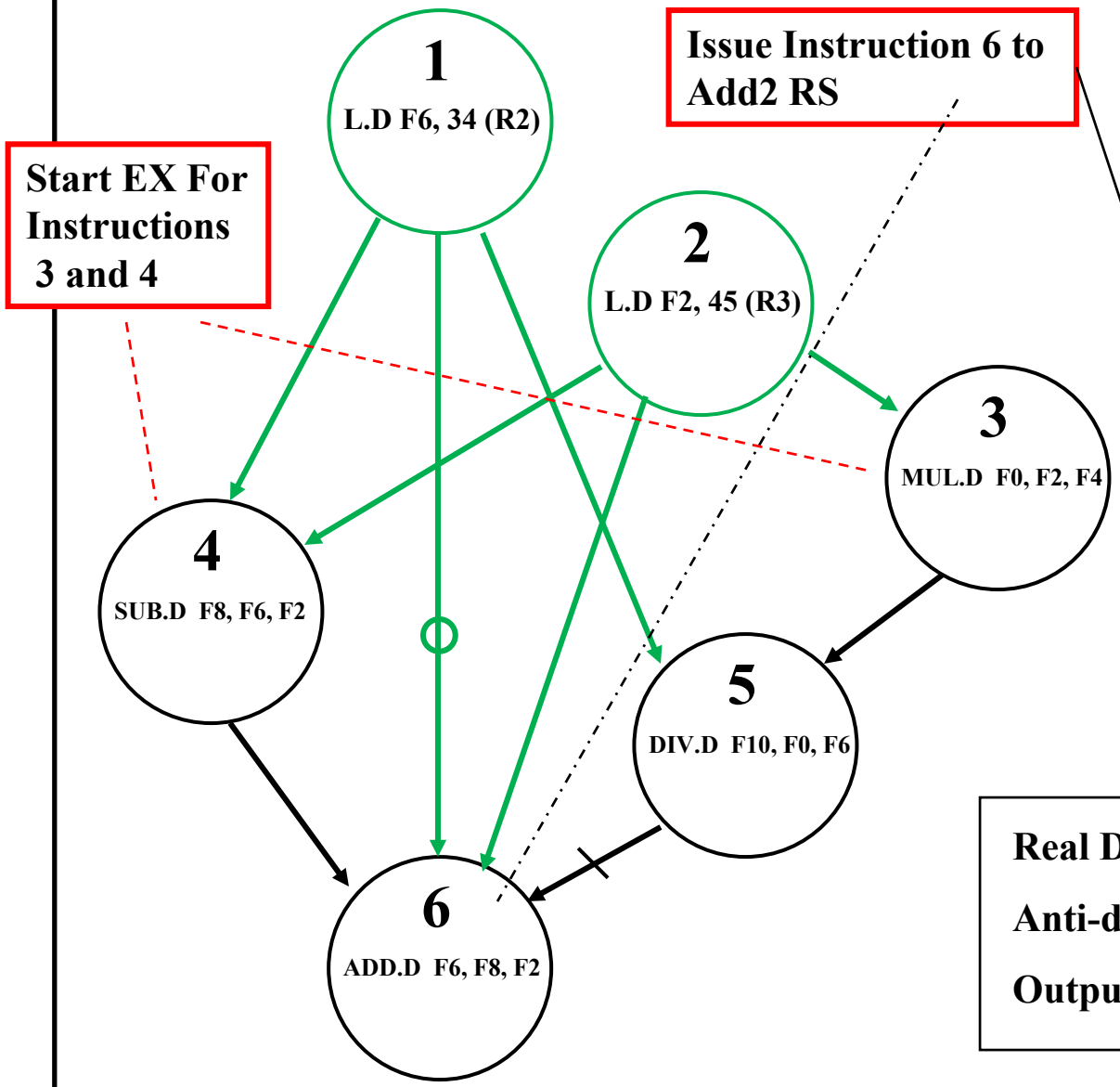
End of



- **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**

Dependency Graph For Example Code

Cycle 6



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)

The same code used is the scoreboard example

IS ⟶ EX ⟶ WB

Tomasulo Example Cycle 6

End of

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

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4		
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6		

Issue →

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

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

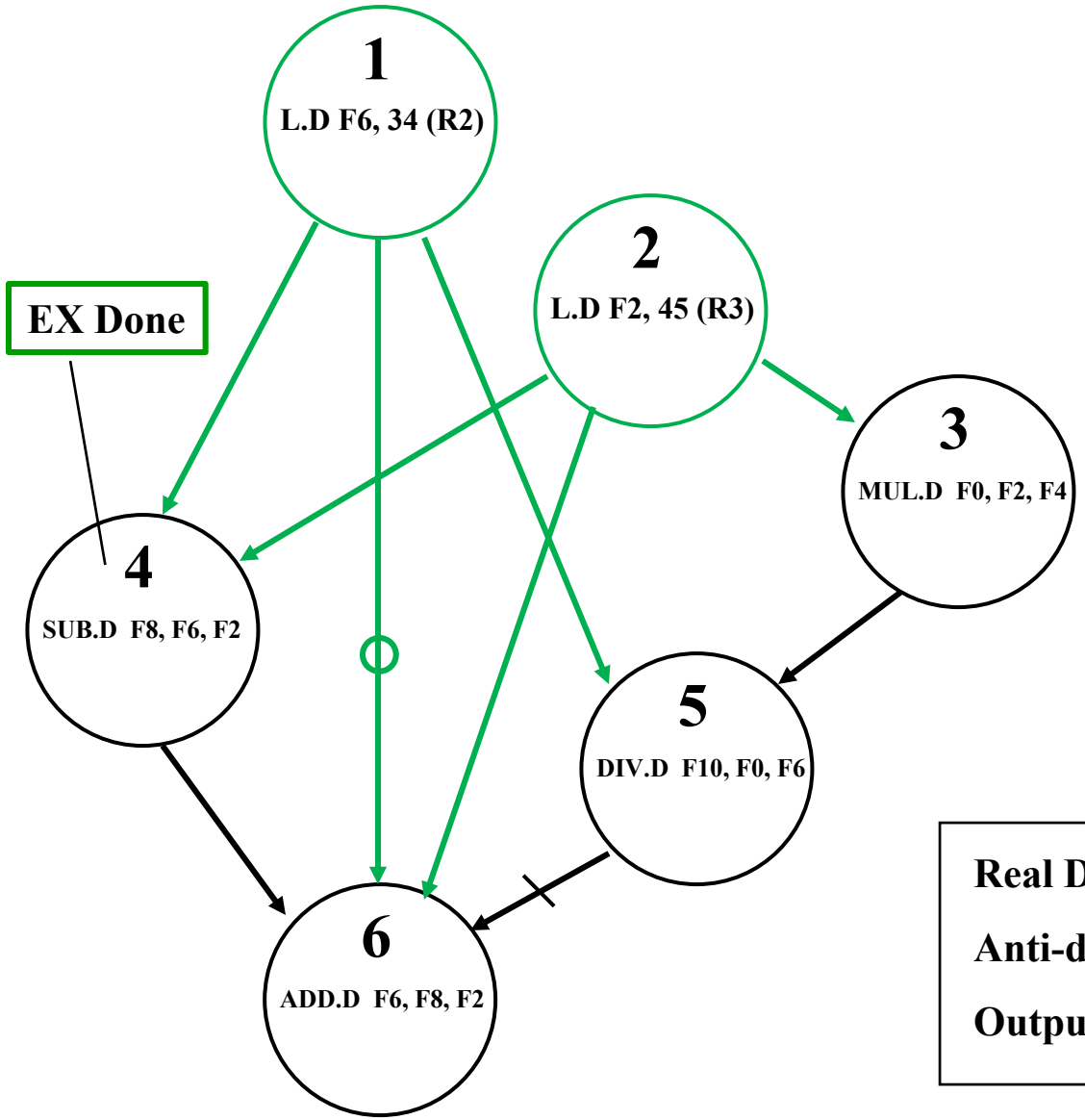
Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
6	Mult1	M(45+R3)		Add2	Add1	Mult2			

FU

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

Dependency Graph For Example Code

Cycle 7



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)

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 7

End of

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

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	→ 7 →	
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

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

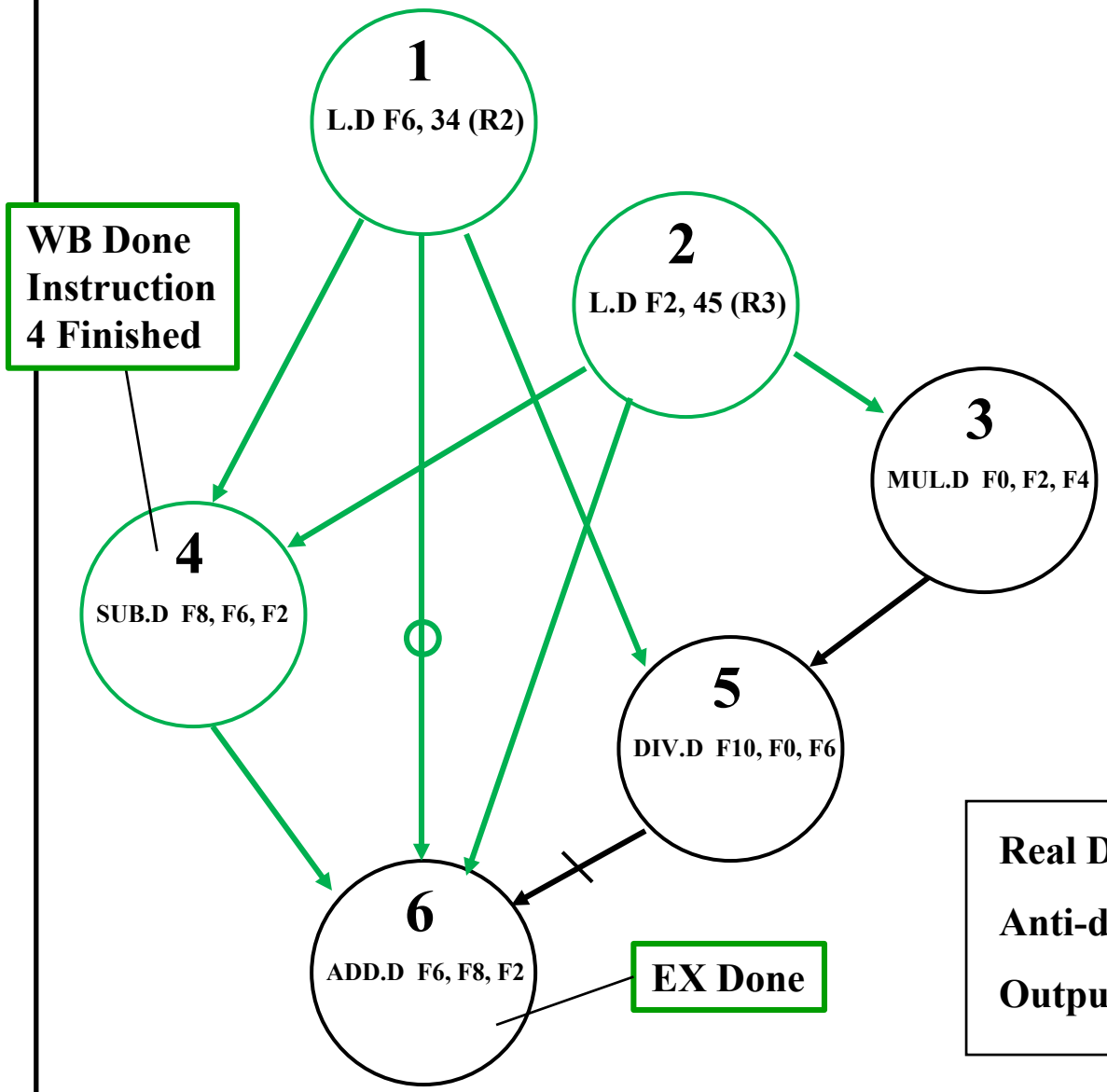
Register result status

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

- **RS Add1 (SUB.D) completing; what is waiting for it?**

Dependency Graph For Example Code

Cycle 10



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)

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 10

End of

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7 →	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6 →	10 →	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
0	Add1	No					
0	Add2	Yes	ADDD	M()–M()	M(45+R3)		
0	Add3	No					
5	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Done executing

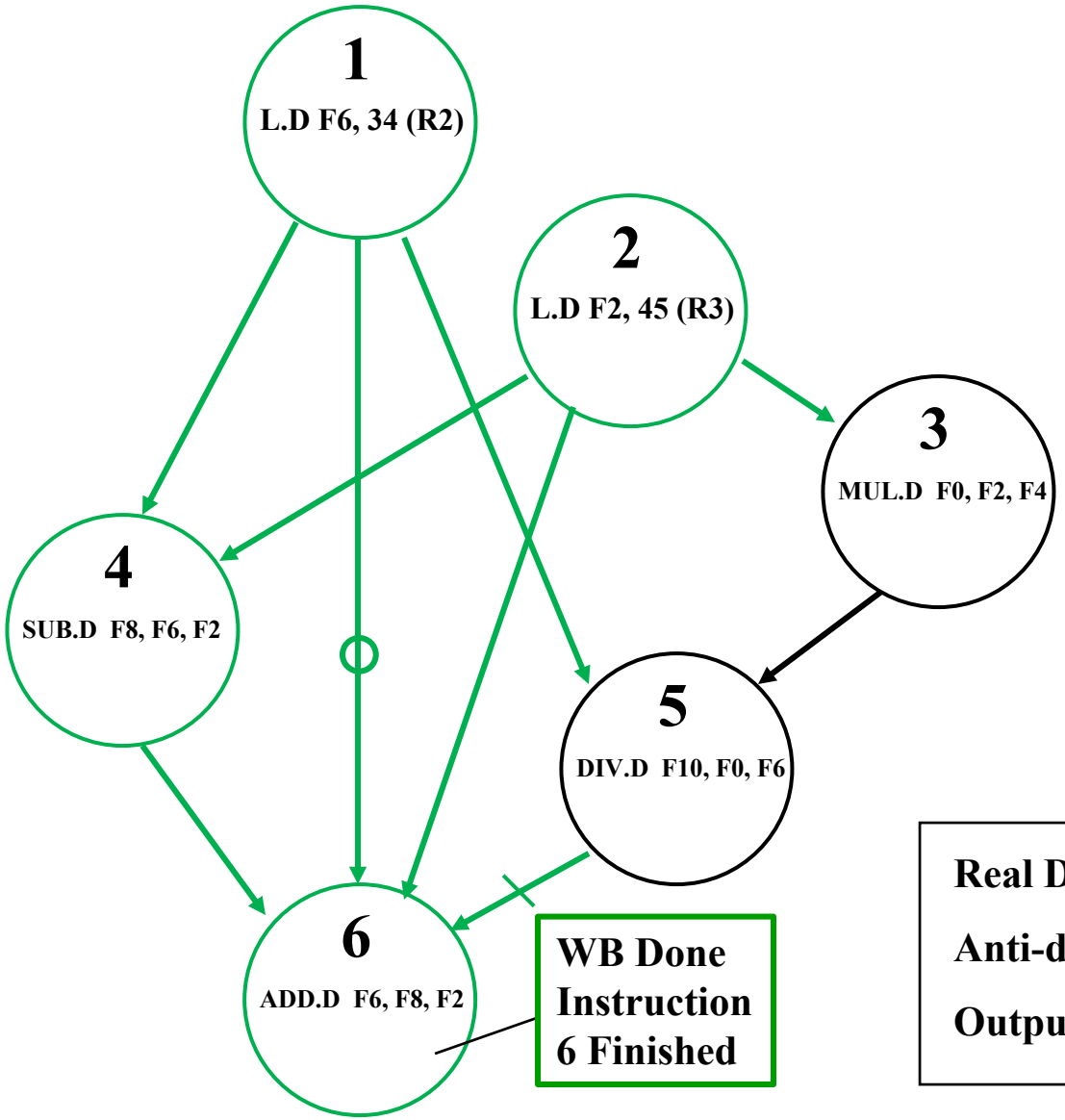
Register result status

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

➔ • RS Add2 (i.e. ADD.D) completed execution

Dependency Graph For Example Code

Cycle 11



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)

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 11

End of

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	→ 11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

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					
4	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

WB Over CDB

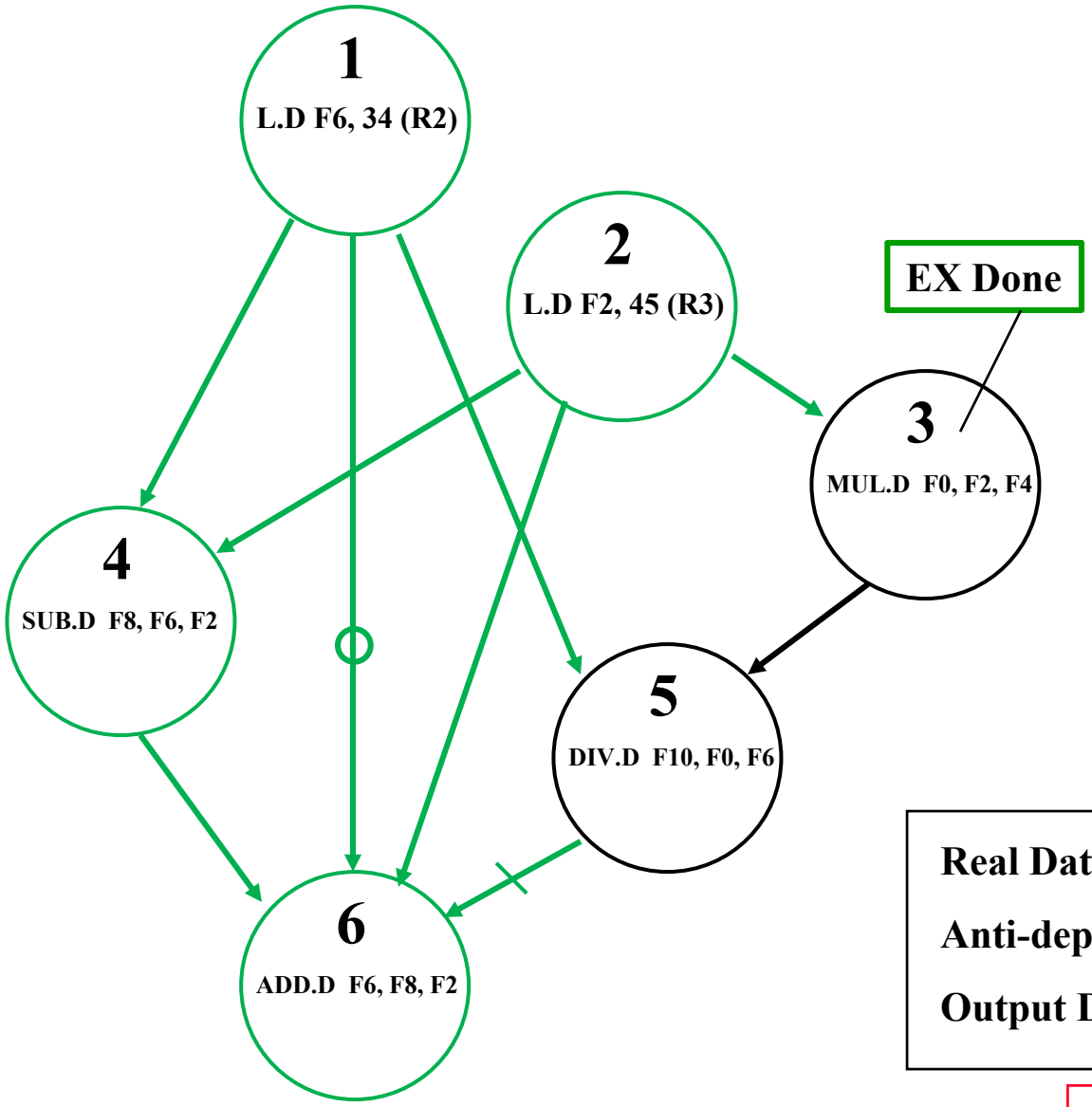
Register result status

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

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

Dependency Graph For Example Code

Cycle 15



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)

The same code used is the scoreboard example

IS → EX → WB

Tomasulo Example: Cycle 15

End of

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3	→ 15 →	
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
		Add3	No					
Done executing	0	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

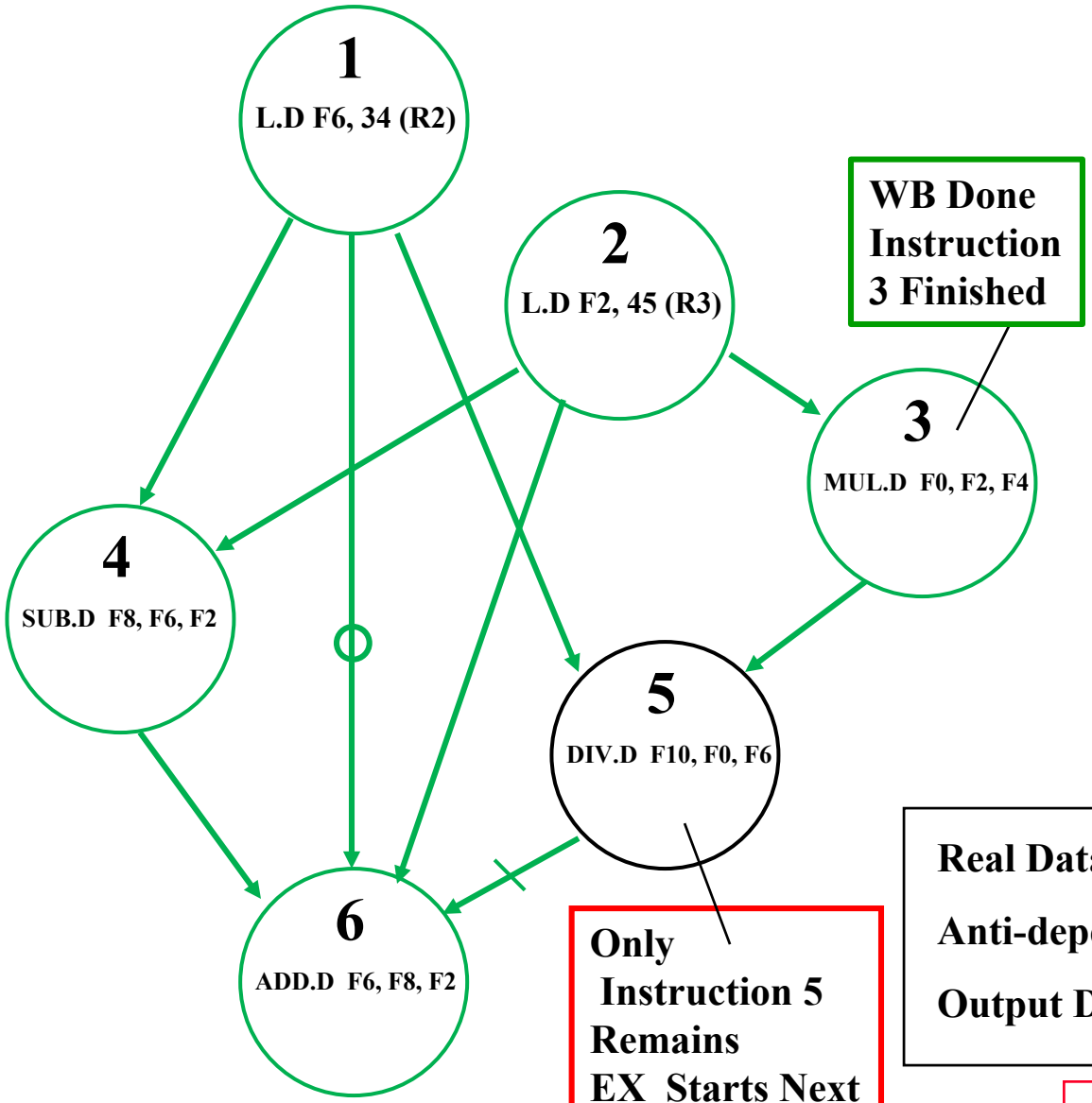
Register result status

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

➔ • Mult1 completed execution; what is waiting for it?

Dependency Graph For Example Code

Cycle 16



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)

The same code used is the scoreboard ex

IS ⟶ EX ⟶ WB

Tomasulo Example: Cycle 16

End of

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

Instruction status

Instruction		j	k	IS Issue	EX Execution complete	WB Write Result
L.D	F6	34+	R2	1	3	4
L.D	F2	45+	R3	2	4	5
MUL.D	F0	F2	F4	3	15	16
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
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)

WB Over CDB

Register result status

Clock 16

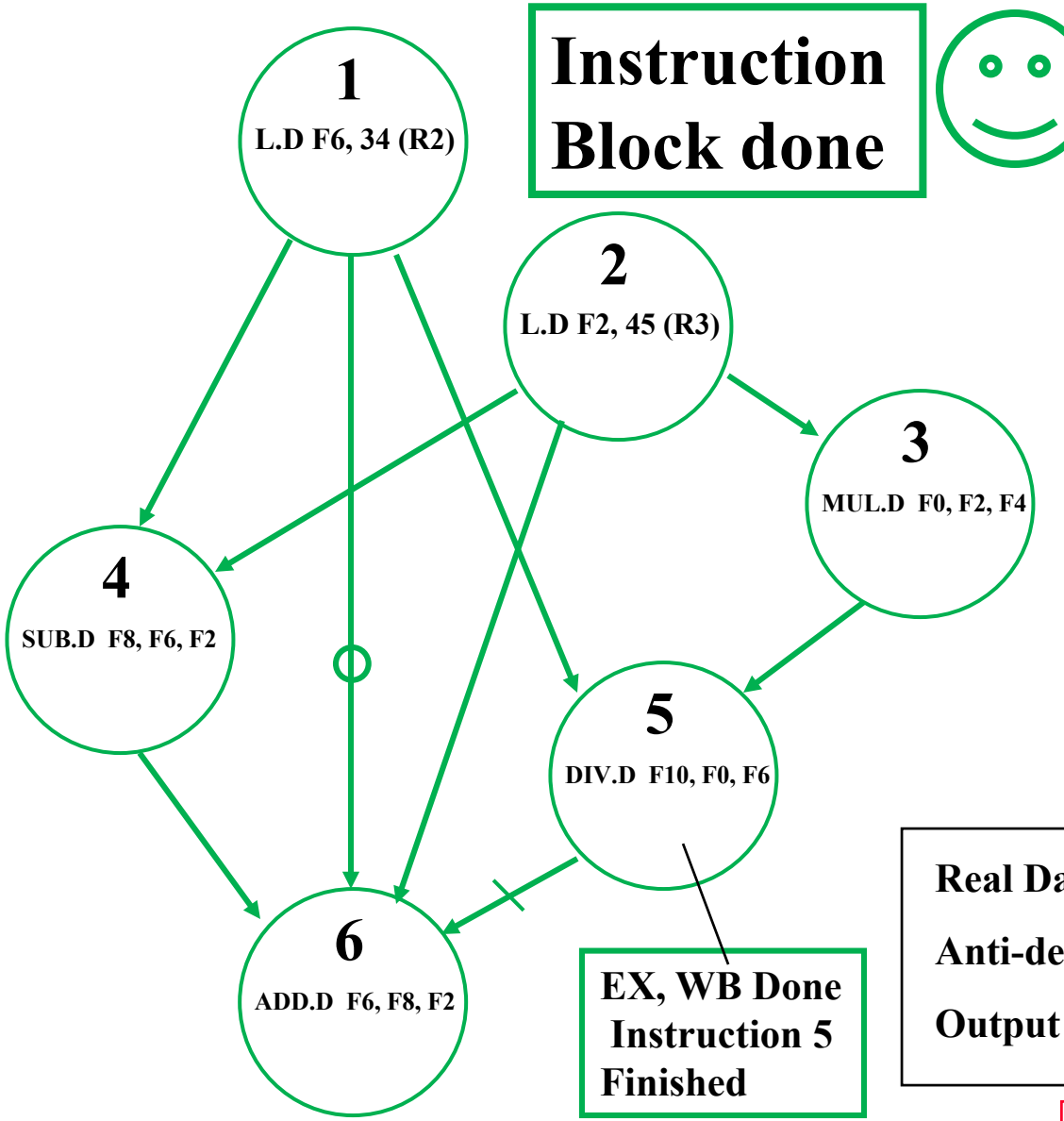
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	M*F4	M(45+R3)		(M–M)+M()	M()–M()	Mult2			

➔ **Only Divide instruction remains**

DIV.D execution will start next cycle (17)

Dependency Graph For Example Code

Cycle 57



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)

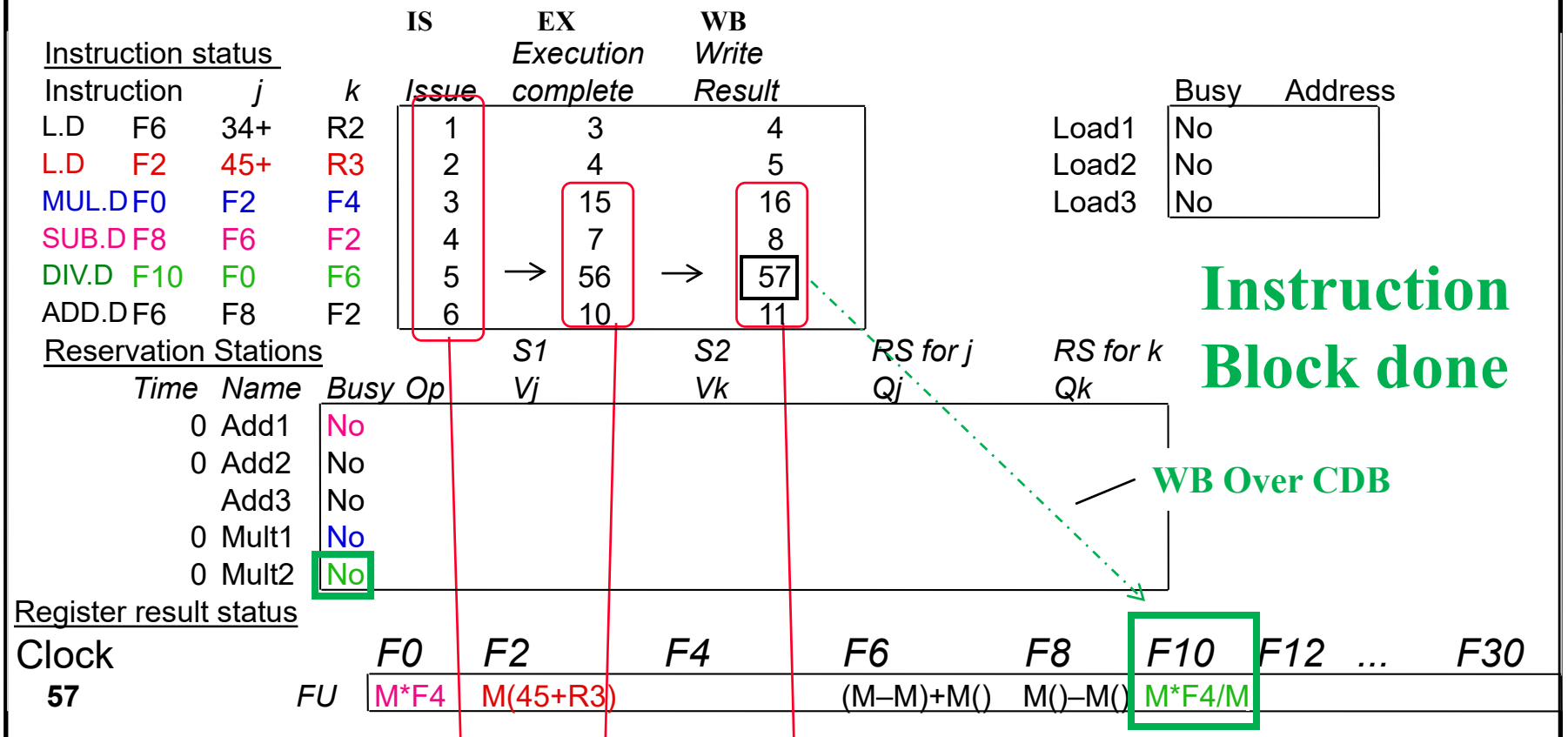
Anti-dependence (WAR)

Output Dependence (WAW)

The same code used is the scoreboard example

Tomasulo Example: Cycle 57

(vs. 62 cycles for scoreboard)



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

Tomasulo Loop Example

(Hardware/CPU-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 (parallel) 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).
- Assume R1 = 80 initially.
- Assume DADDUI only takes one cycle (issue)
- Assume branch resolved in issue stage (no EX or CDB write)
- Assume branch is predicted taken and no branch misprediction.
- 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.

3rd ….

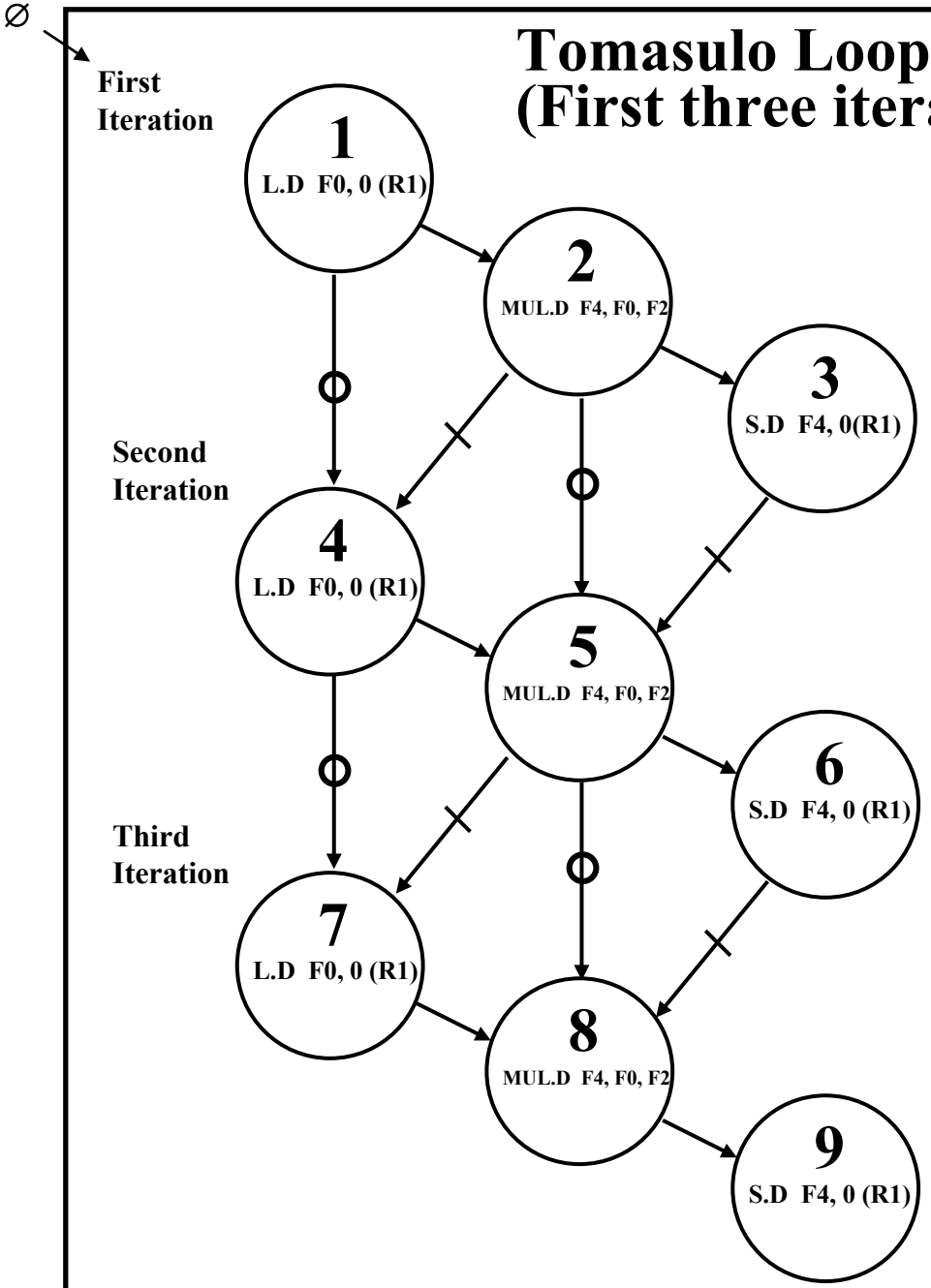
i.e. Perfect branch prediction. How?

Target?

What if prediction Is wrong?

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

Tomasulo Loop Example Dependency Graph (First three iterations shown)



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)

IS EX WB

Instruction status

Instruction	j	k	iteration	Issue	Execution complete	Write Result
L.D F0	0	R1	1			
MUL.D F4	F0	F2	1			
S.D F4	0	R1	1			
L.D F0	0	R1	2			
MUL.D F4	F0	F2	2			
S.D F4	0	R1	2			

Store Reservation Stations

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

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					
0	Mult1	No					
0	Mult2	No					

Operand Values (Data)

Producer of Result Needed

Code:

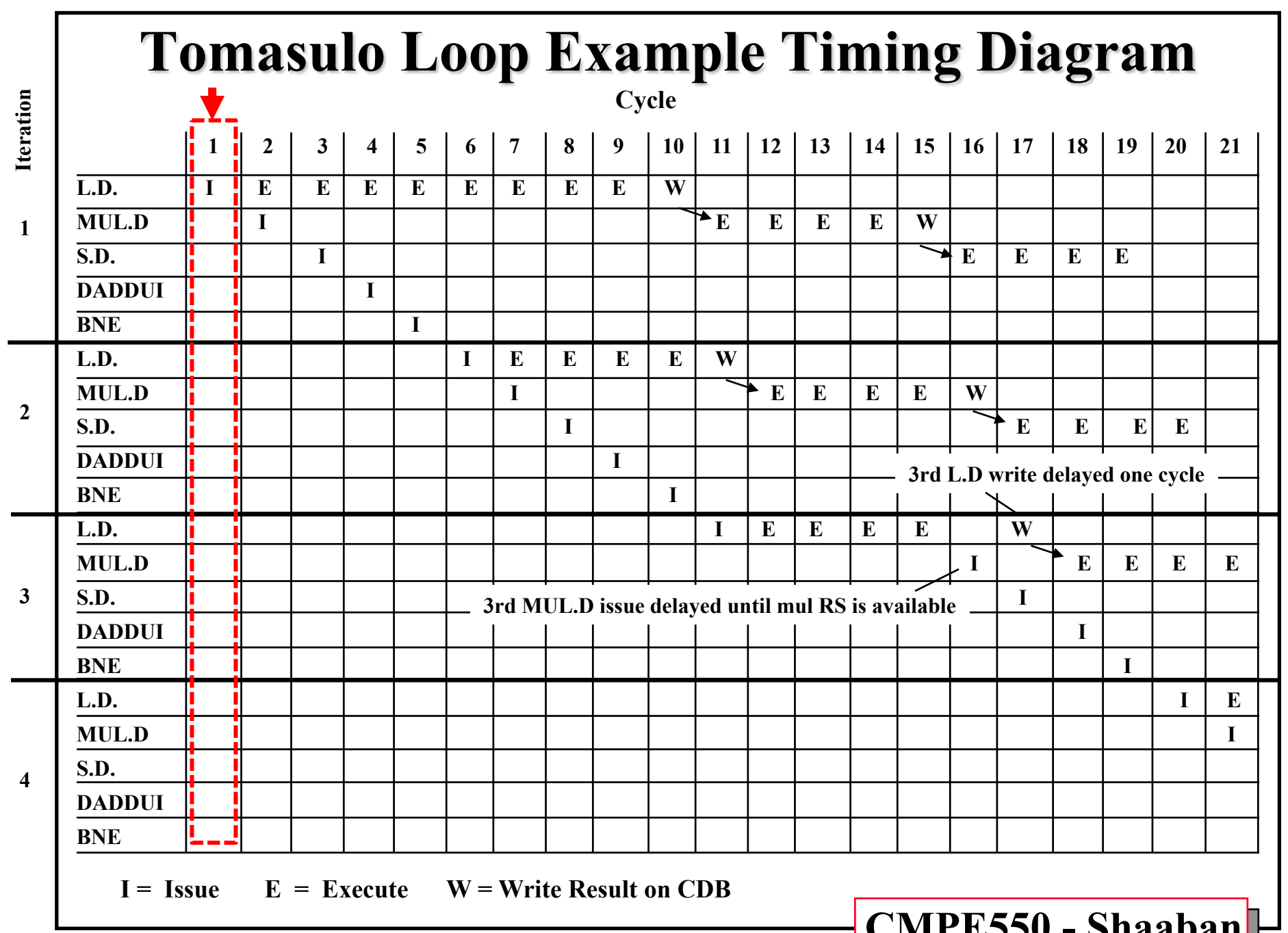
```
L.D     F0, 0(R1)
MUL.D   F4,F0,F2
S.D     F4, 0(R1)
DADDUI  R1, R1, #-8
BNE     R1,R2,loop
```

Register result status

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

IS (Issue) → **EX** (Includes MEM) → **WB**

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 1

End of

IS EX WB

Instruction status

Issue	Instruction		j	k	iteration	Issue	Execution complete	Write Result
→	L.D	F0	0	R1	1	1		
	MUL.D	F4	F0	F2	1			
	S.D	F4	0	R1	1			
	L.D	F0	0	R1	2			
	MUL.D	F4	F0	F2	2			
	S.D	F4	0	R1	2			

EX Cycles remaining

		Busy	Address	
Load1	8	Yes	80	←
Load2		No		
Load3		No		Qi
Store1		No		
Store2		No		
Store3		No		

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					
0	Mult1	No					
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

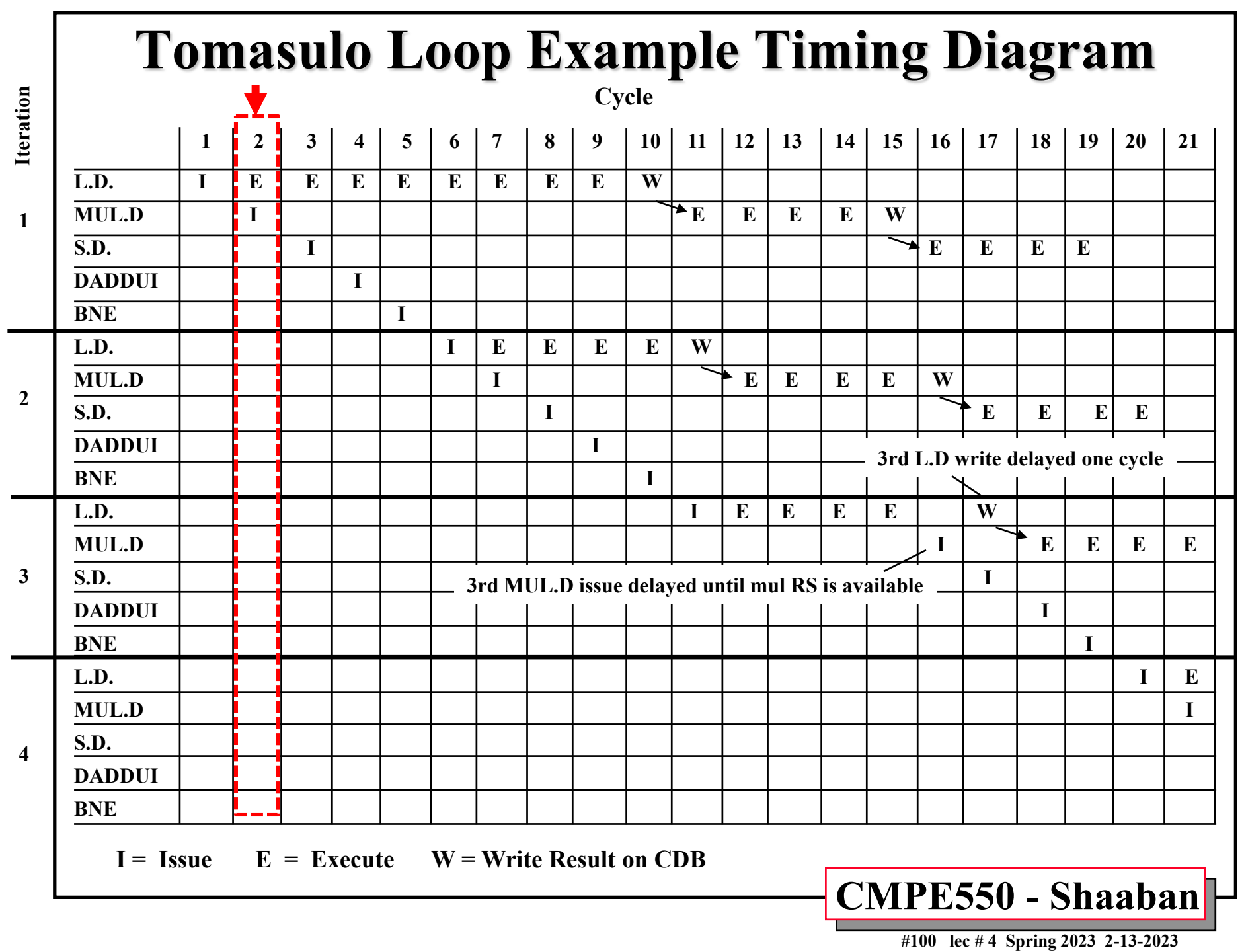
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
1	80	Qi	Load1							

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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 2

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1			
L.D	F0	0	R1	2			
MUL.D	F4	F0	F2	2			
S.D	F4	0	R1	2			

Issue →

	Busy	Address	Qi
Load1 7	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
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
```

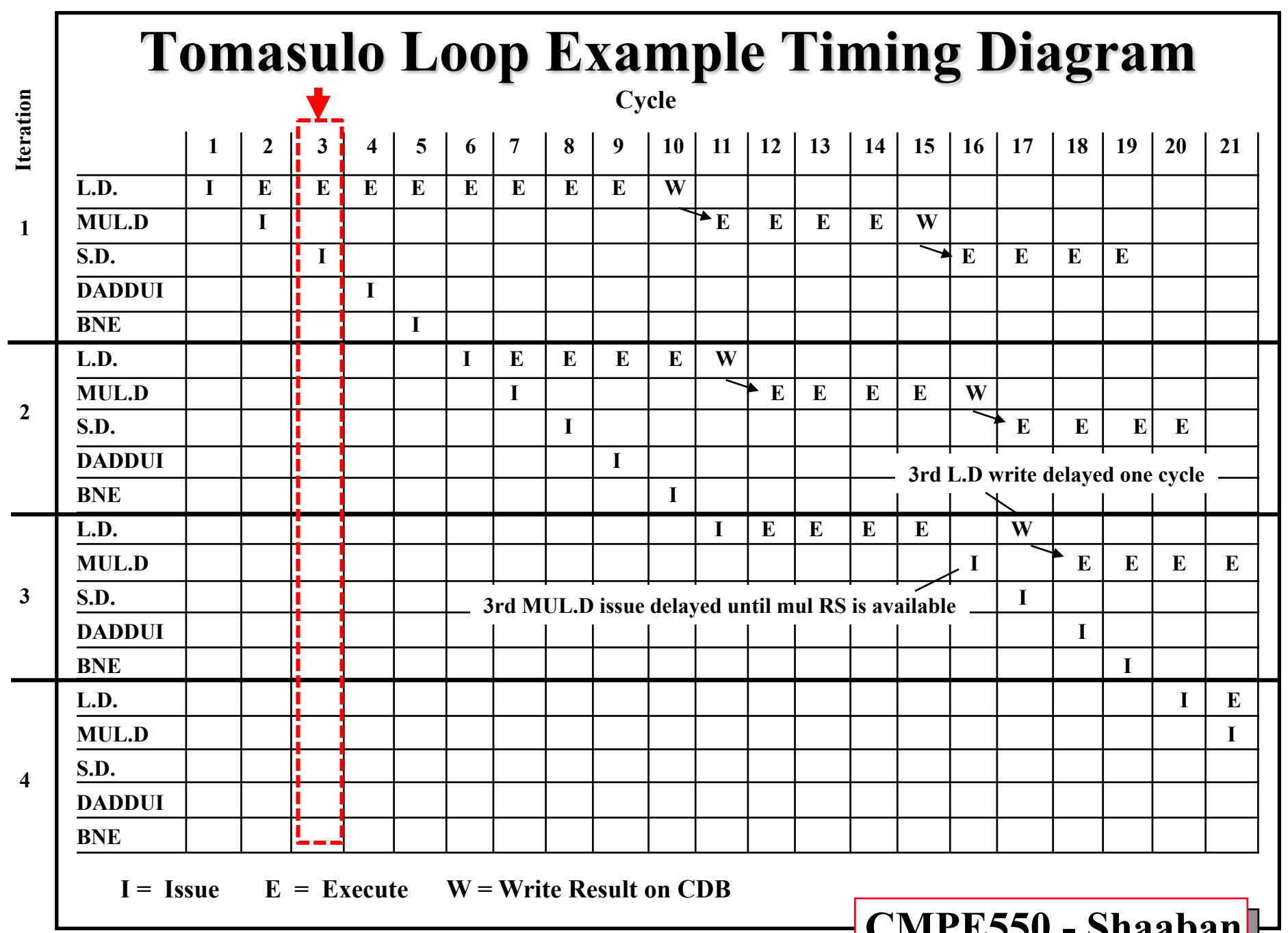
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
2	80	Qi	Load1		Mult1					

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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 3

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2			
MUL.D	F4	F0	F2	2			
S.D	F4	0	R1	2			

Issue

	Busy	Address	Qi
Load1 6	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
0	Mult2	No					

Code:

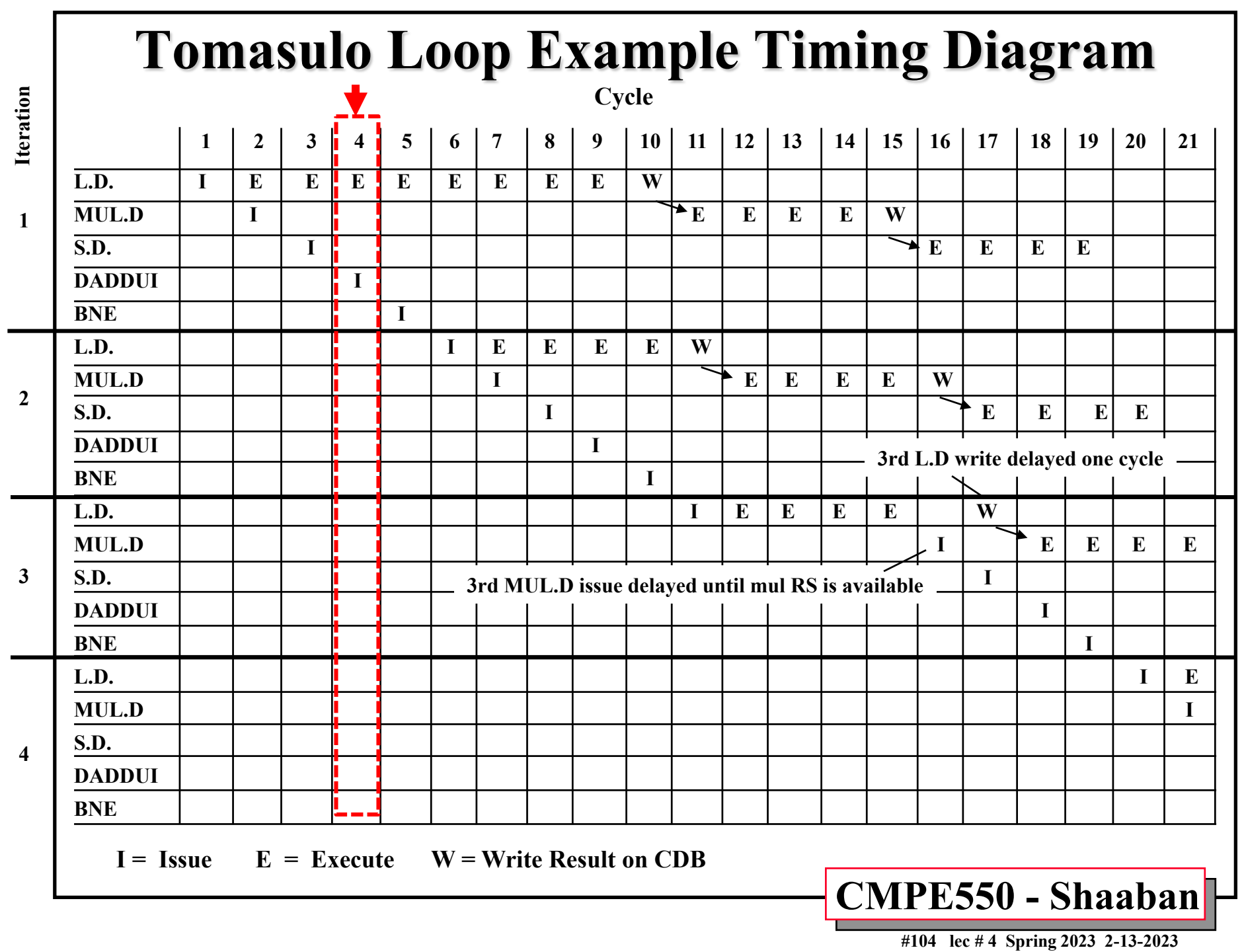
```
L.D     F0, 0(R1)
MUL.D   F4,F0,F2
S.D     F4, 0(R1)      <- Issue
DADDUI  R1, R1, #-8
BNE     R1,R2,loop
```

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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 4

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2			
MUL.D	F4	F0	F2	2			
S.D	F4	0	R1	2			

	Busy	Address	Qi
Load1 5	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
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
```

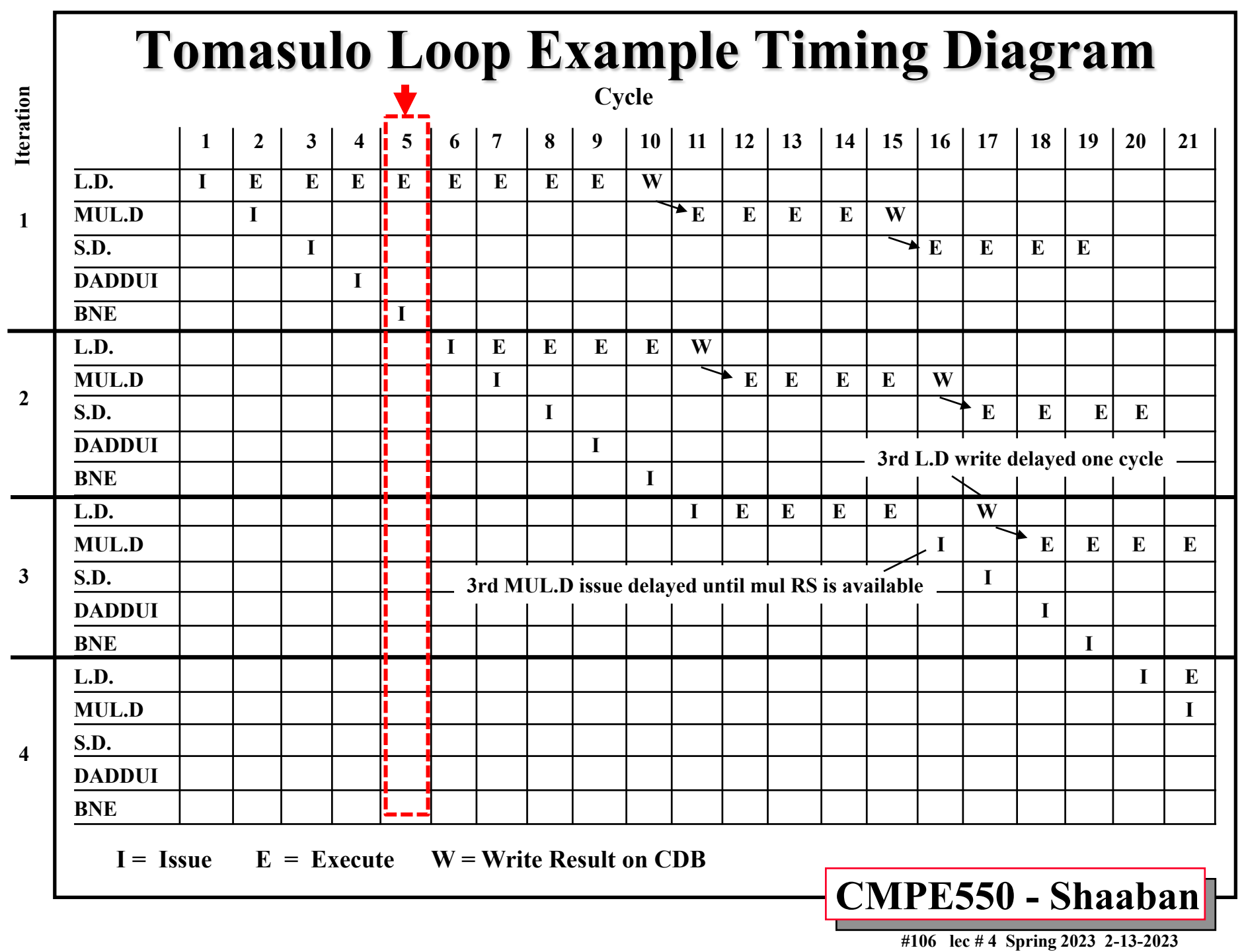
Issue

Register result status

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

First DADDUI issues (not shown)

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 5

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2			
MUL.D	F4	F0	F2	2			
S.D	F4	0	R1	2			

	Busy	Address	Qi
Load1 4	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
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
```

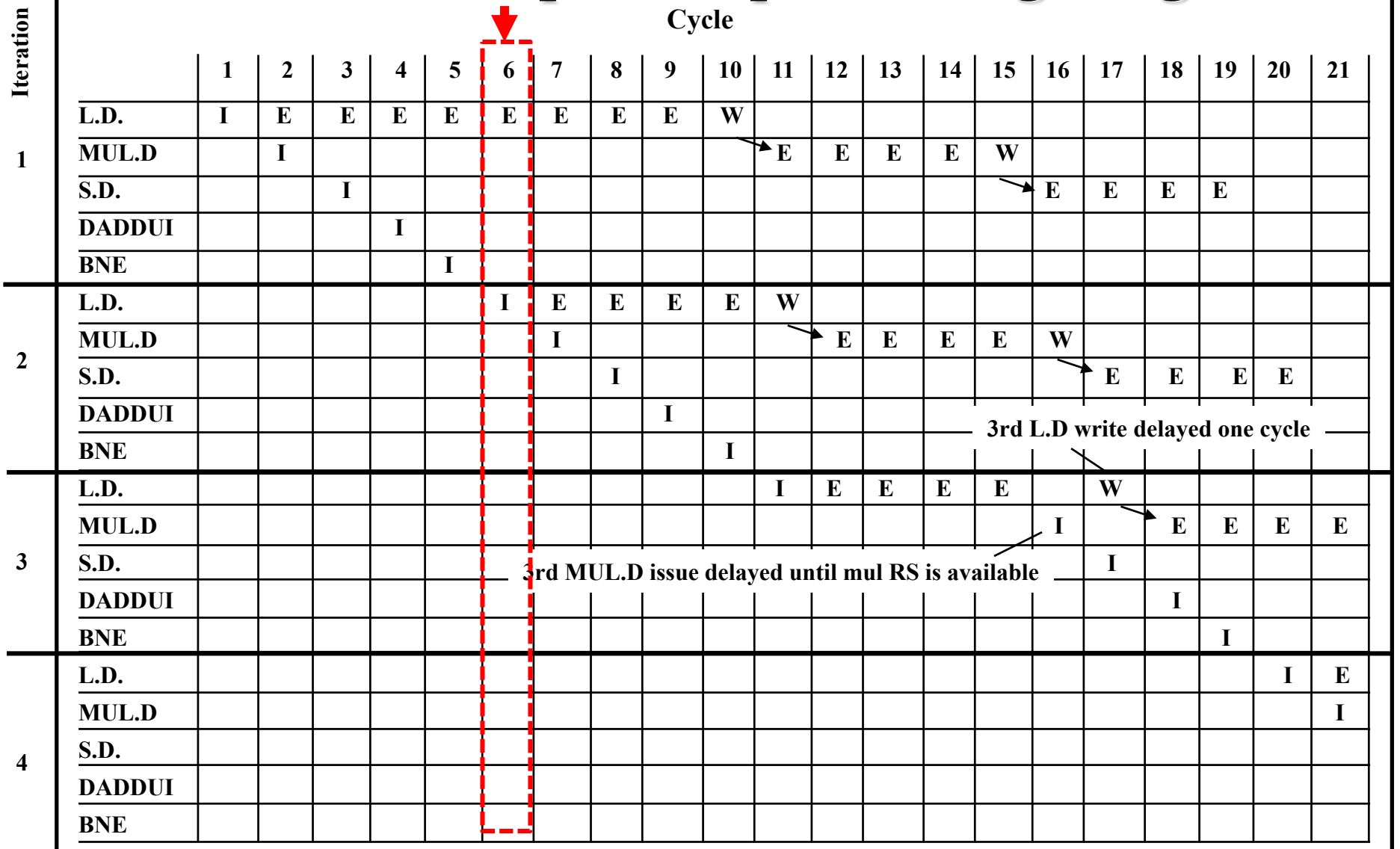
Issue

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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 6

End of

IS EX WB

Instruction status

Instruction	j	k	iteration	Issue	Execution complete	Write Result
L.D F0	0	R1	1	1		
MUL.D F4	F0	F2	1	2		
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6		
MUL.D F4	F0	F2	2			
S.D F4	0	R1	2			

Issue

		Busy	Address	Qi
Load1	3	Yes	80	
Load2	4	Yes	72	
Load3		No		
Store1		Yes	80	Mult1
Store2		No		
Store3		No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
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
```

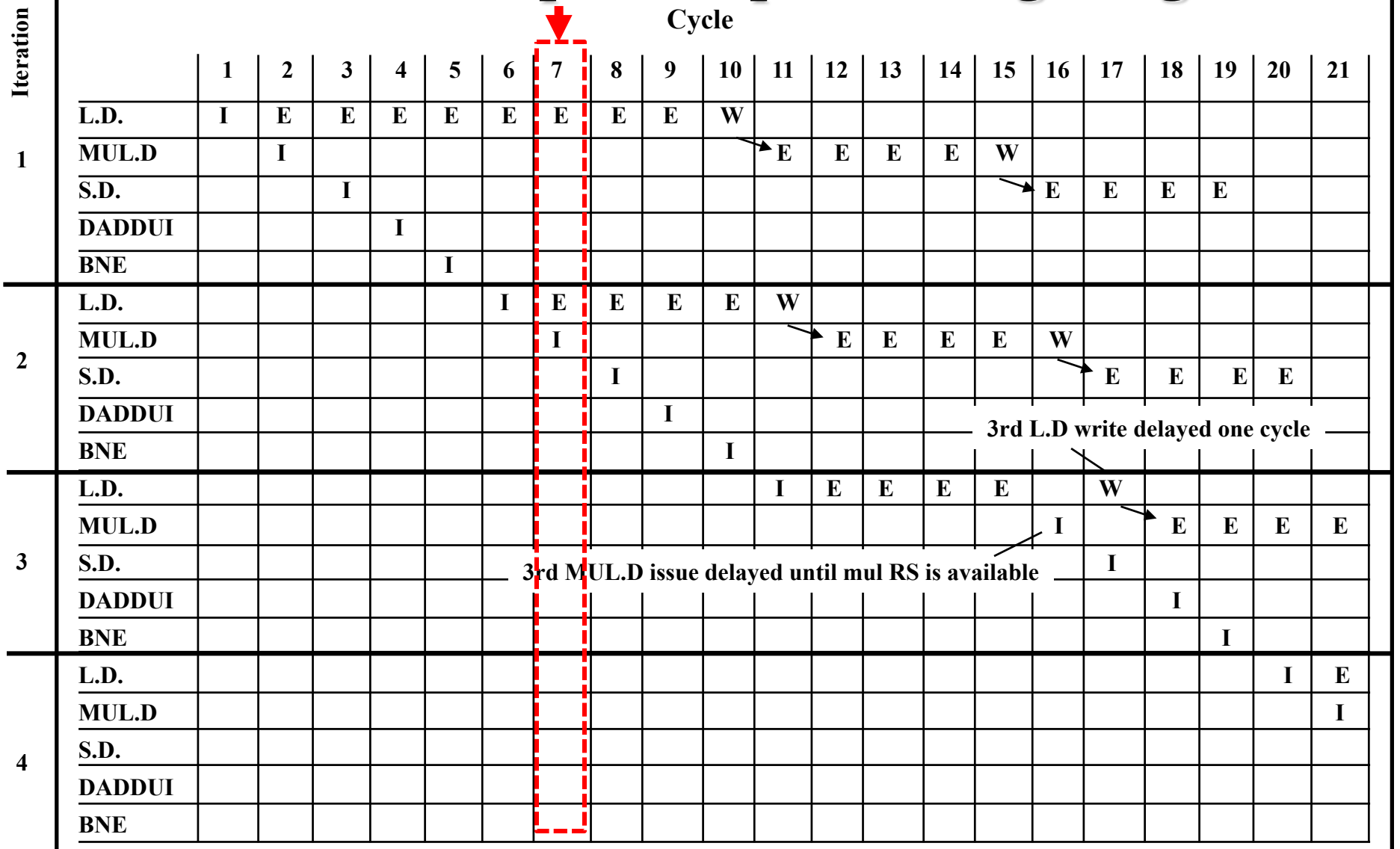
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
6	72	Qi	Load2		Mult1					

- 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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 7

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6		
MUL.D	F4	F0	F2	2	7		
S.D	F4	0	R1	2			

Issue →

Load1	2	Yes	80
Load2	3	Yes	72
Load3		No	
Store1		Yes	80
Store2		No	
Store3		No	

Busy Address Qi

Mult1

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
→ 0	Mult2	Yes	MULTD		R(F2)	Load2	

Code:

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

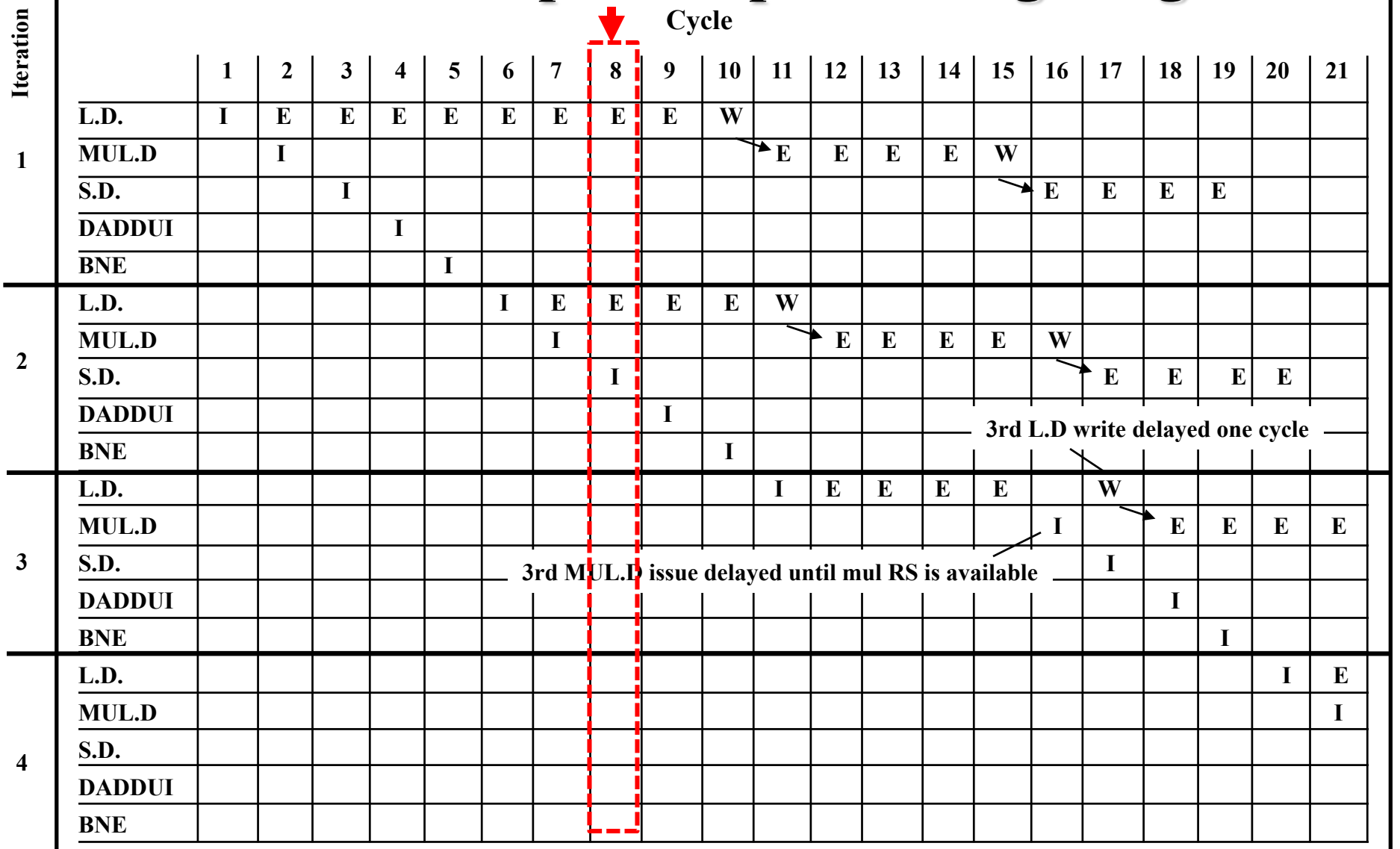
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
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

Tomasulo Loop Example Timing Diagram



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

Loop Example Cycle 8

End of

Issue

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1		
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6		
MUL.D	F4	F0	F2	2	7		
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1 1	Yes	80	
Load2 2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
0	Mult2	Yes	MULTD		R(F2)	Load2	

Code:

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

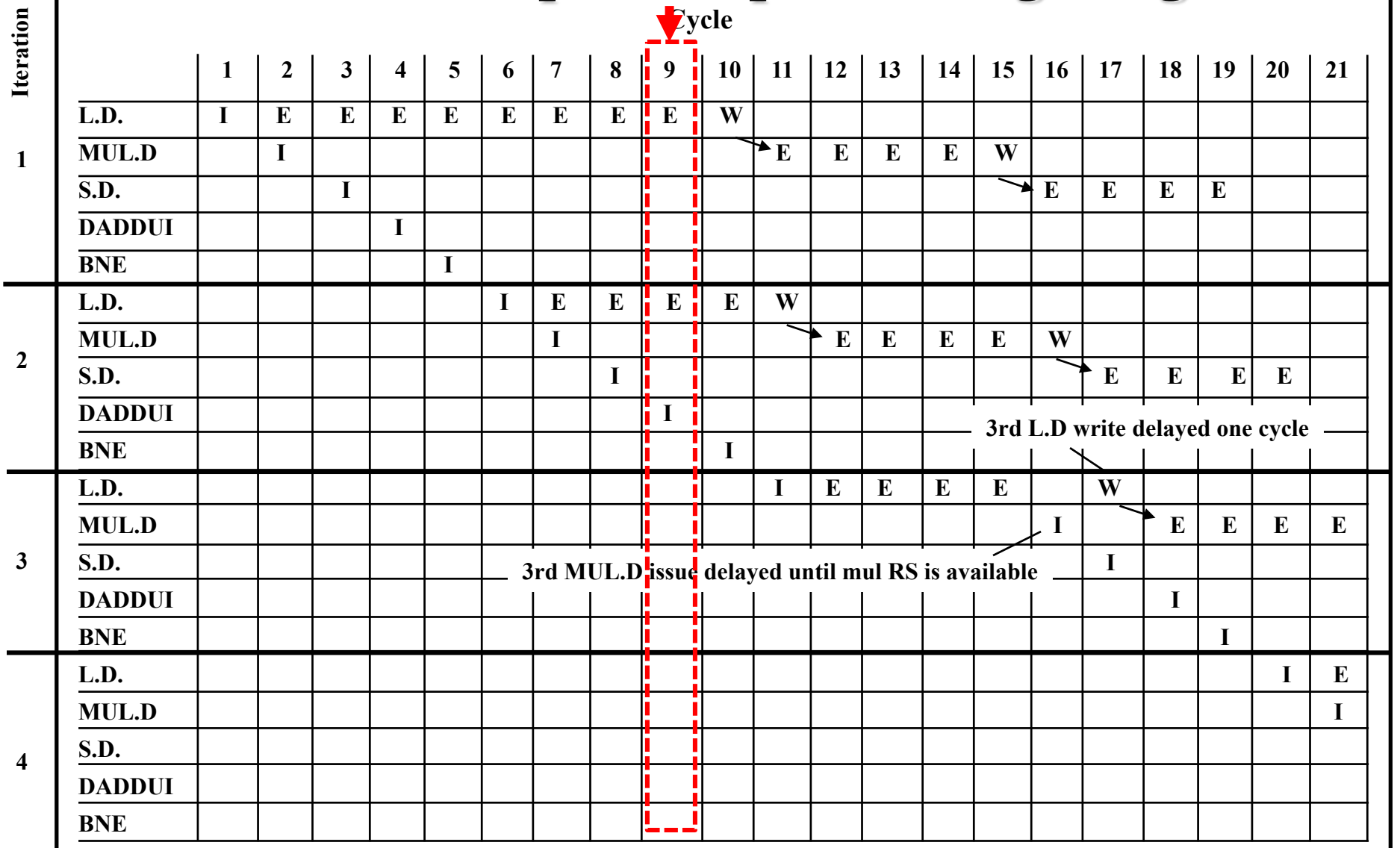
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
8	72	Qi	Load2		Mult2					

- Second S.D issues (to RS Store2)

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 9

End of

First Load EX Done

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1	9	
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6		
MUL.D	F4	F0	F2	2	7		
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1 0	Yes	80	
Load2 1	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
0	Mult1	Yes	MULTD		R(F2)	Load1	
0	Mult2	Yes	MULTD		R(F2)	Load2	

Code:

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

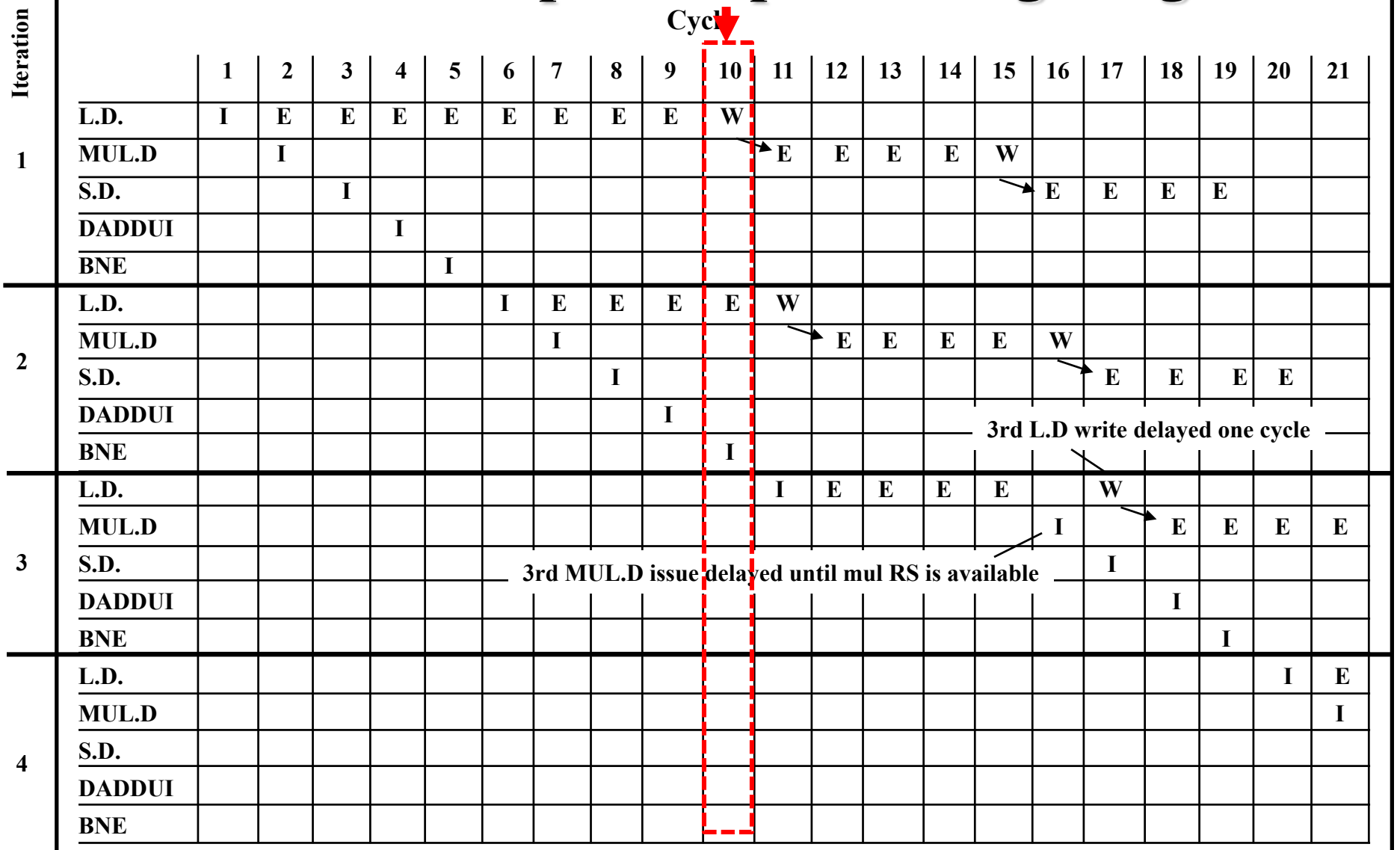
Issue

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?

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 10

End of

Second Load EX Done

Instruction status

Instruction	j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2		
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6	10	
MUL.D F4	F0	F2	2	7		
S.D F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
4	Mult1	Yes	MULTD	M(80)	R(F2)		
0	Mult2	Yes	MULTD		R(F2)	Load2	

WB Over CDB

Execution cycles remaining (execution actually starts next cycle)

Code:

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

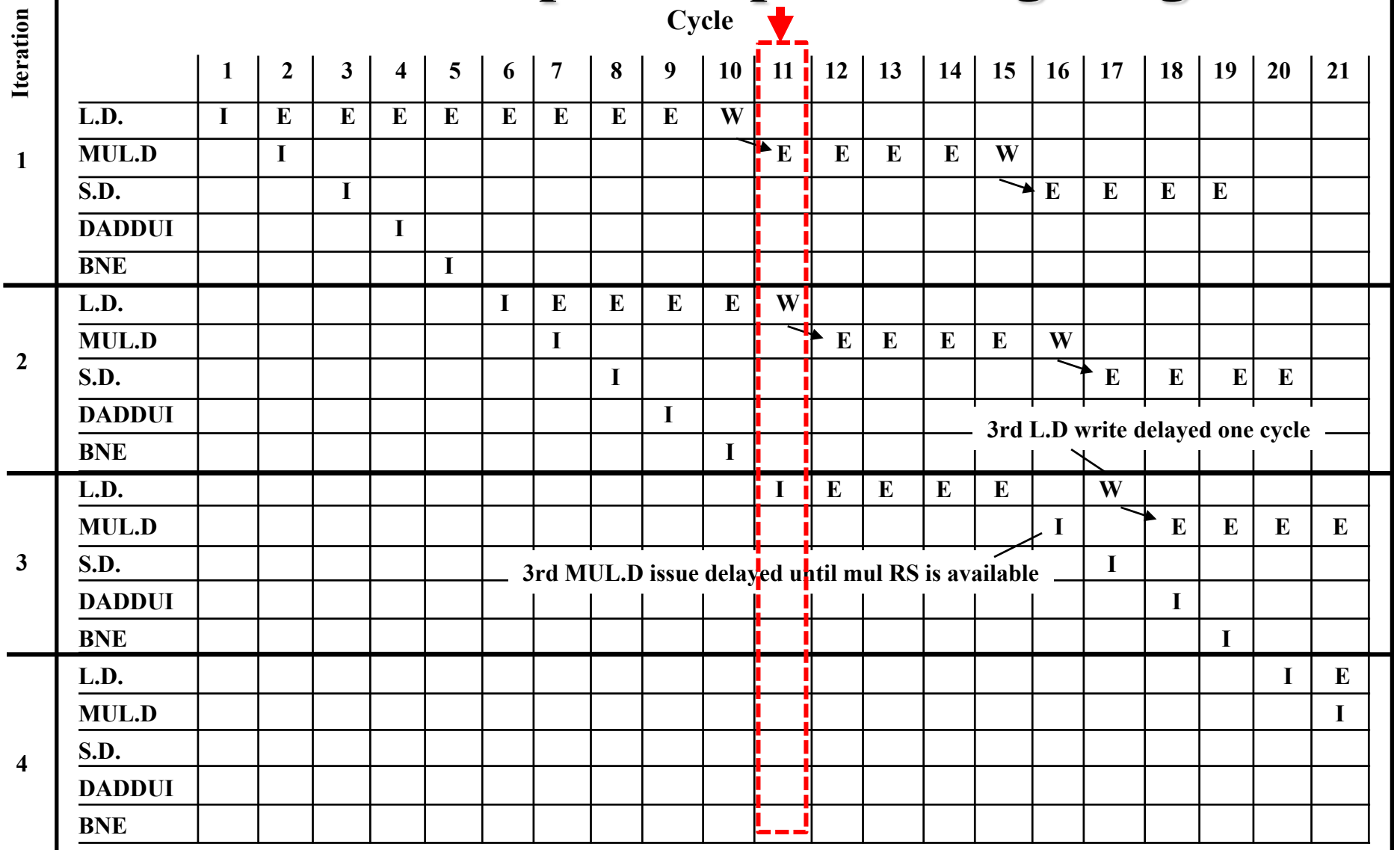
Issue

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
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?

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 11

End of

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2		
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7		
S.D	F4	0	R1	2	8		

IS EX WB

	Busy	Address	
Load1	No		
Load2	No		
Load3 4	Yes	64	Qi
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
3	Mult1	Yes	MULTD	M(80)	R(F2)		
4	Mult2	Yes	MULTD	M(72)	R(F2)		

WB Over CDB

Execution cycles remaining (execution actually starts next cycle)

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

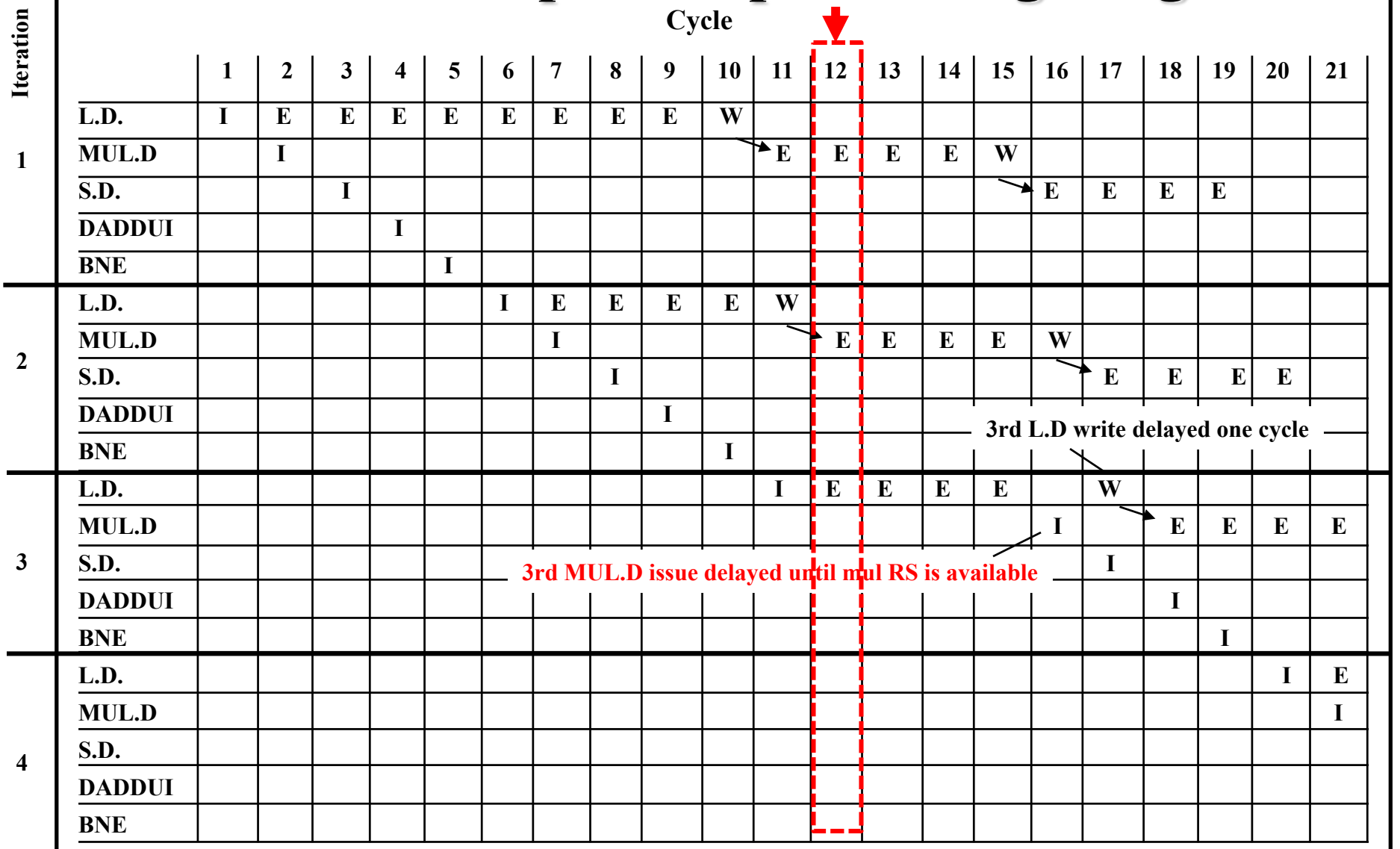
Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
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)

Tomasulo Loop Example Timing Diagram



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

Loop Example Cycle 12

End of

Instruction status

Instruction	j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2		
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6	10	11
MUL.D F4	F0	F2	2	7		
S.D F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3 3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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(80)	R(F2)		
3	Mult2	Yes	MULTD	M(72)	R(F2)		

? — Code:

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

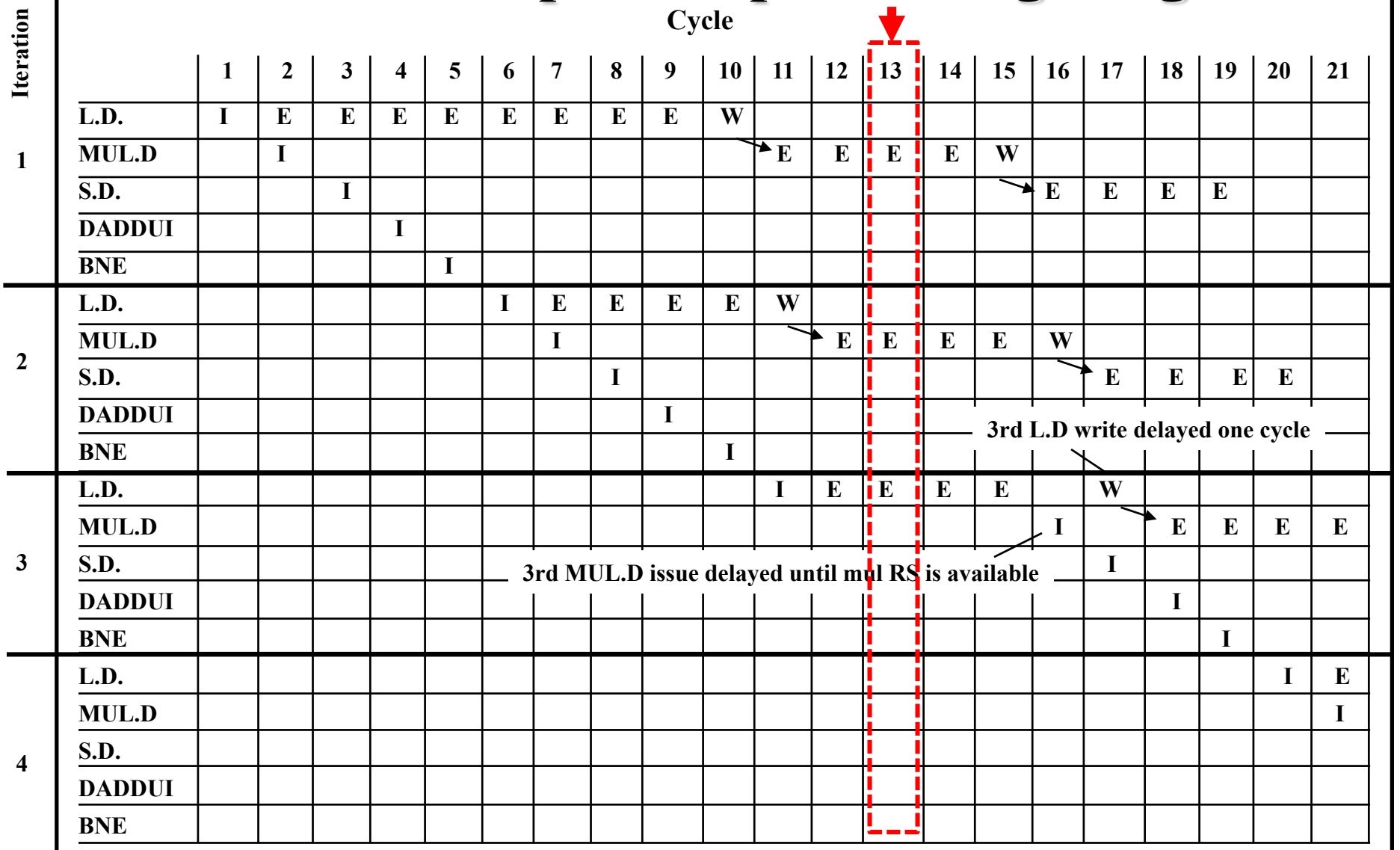
Register result status

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

➔ Issue third iteration MUL.D ?

No both Mult reservation stations busy

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 13

End of

Instruction status

Instruction	j	k	iteration	Issue (IS)	Execution complete (EX)	Write Result (WB)
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2		
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6	10	11
MUL.D F4	F0	F2	2	7		
S.D F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3 2	Yes	64	Qi
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
1	Mult1	Yes	MULTD	M(80)	R(F2)		
2	Mult2	Yes	MULTD	M(72)	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
```

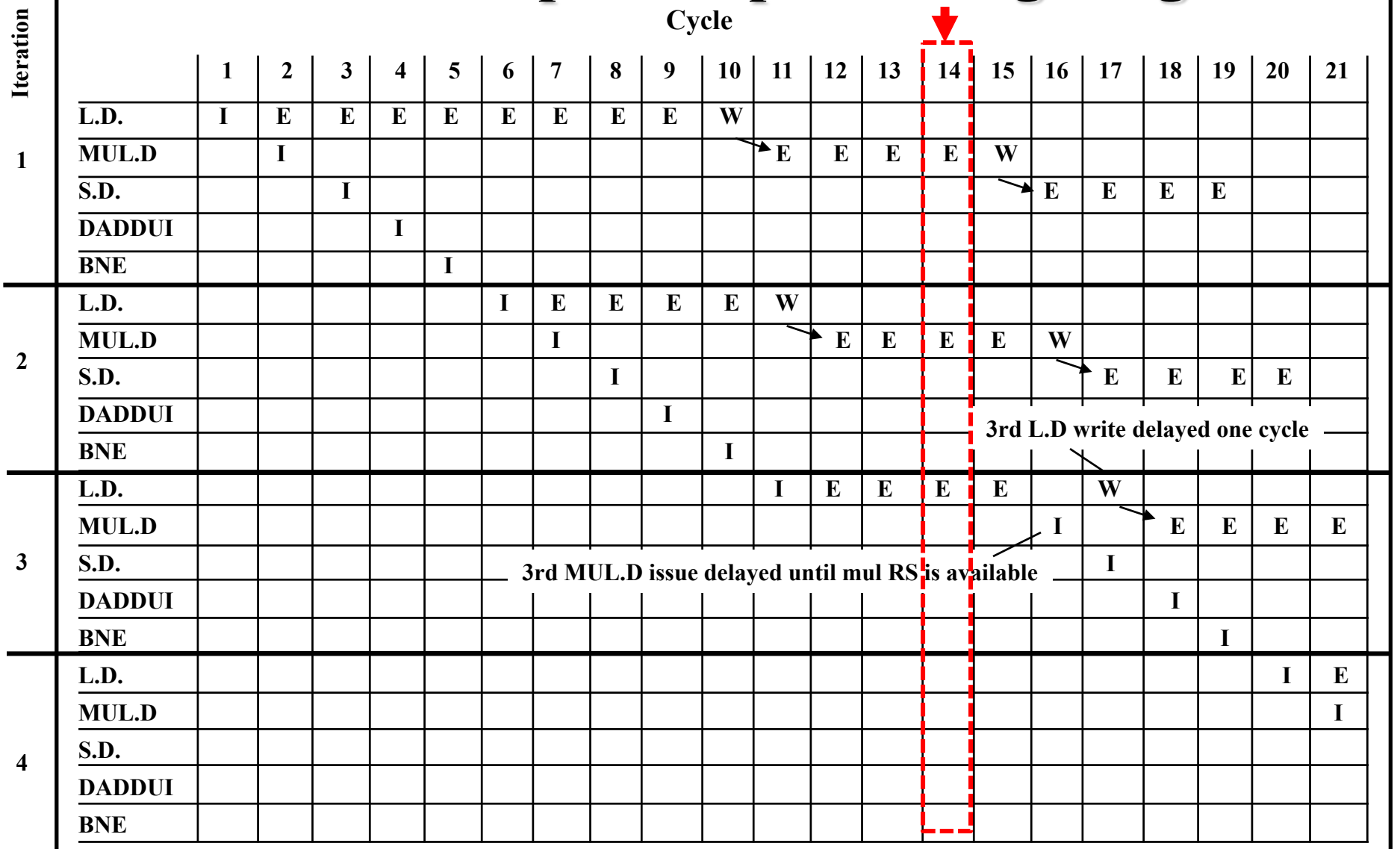
Issue?

Register result status

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

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

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 14

End of

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2	14	
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7		
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3 1	Yes	64	Qi
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

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					
EX Done	0	Mult1	Yes	MULTD	M(80)	R(F2)		
	1	Mult2	Yes	MULTD	M(72)	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
```

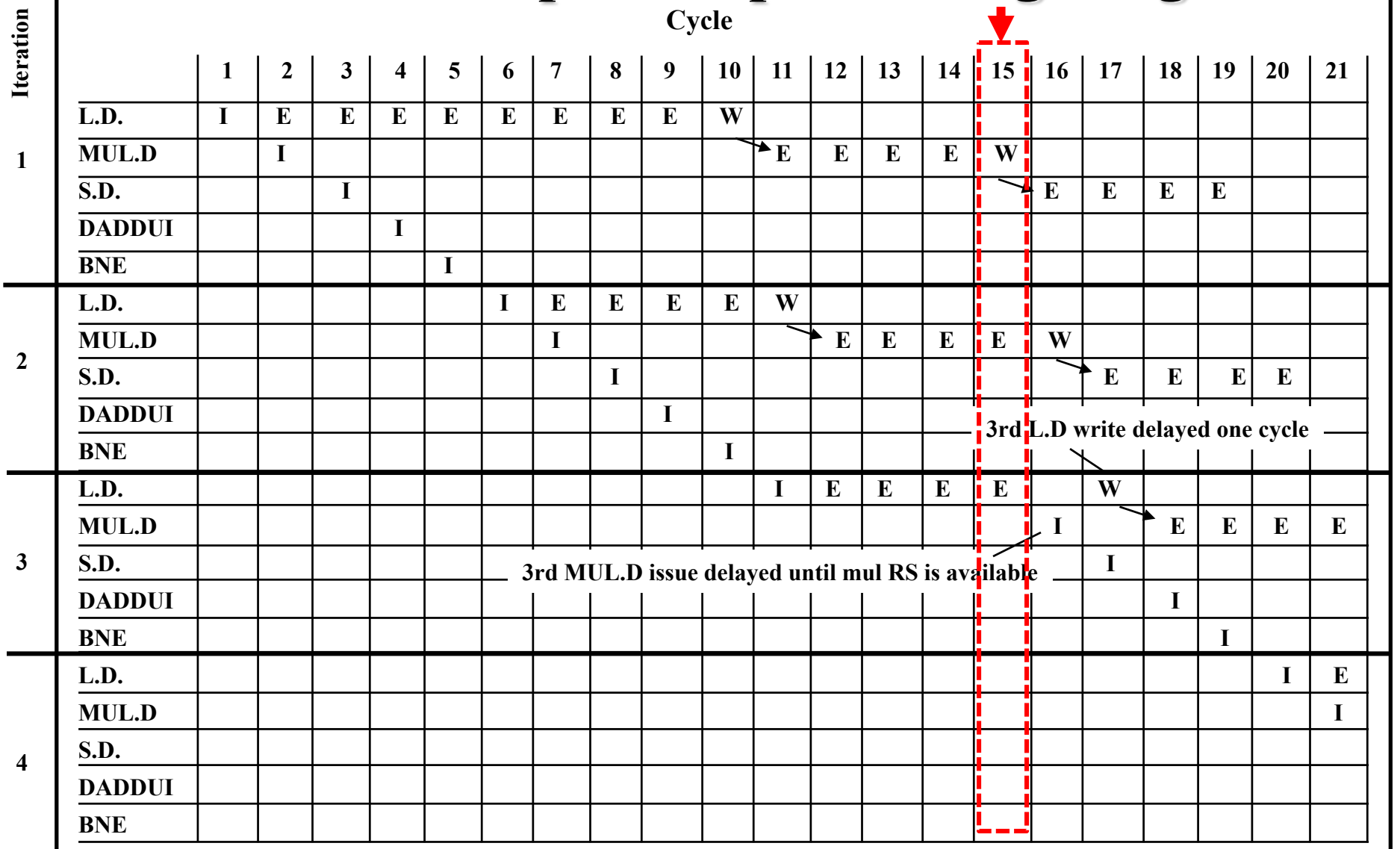
Issue?

Register result status

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

- **Mult1 completing; what is waiting for it?**

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 15

End of

Third Load EX Done

Instruction status

Instruction	j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2	14	15
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6	10	11
MUL.D F4	F0	F2	2	7	15	
S.D F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3 0	Yes	64	
Store1 4	Yes	80	M(80)*R(F2)
Store2	Yes	72	Mult2
Store3	No		

WB Over CDB

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					
0	Mult1	No					
0	Mult2	Yes	MULTD	M(72)	R(F2)		

Available by end of this cycle

EX Done

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?

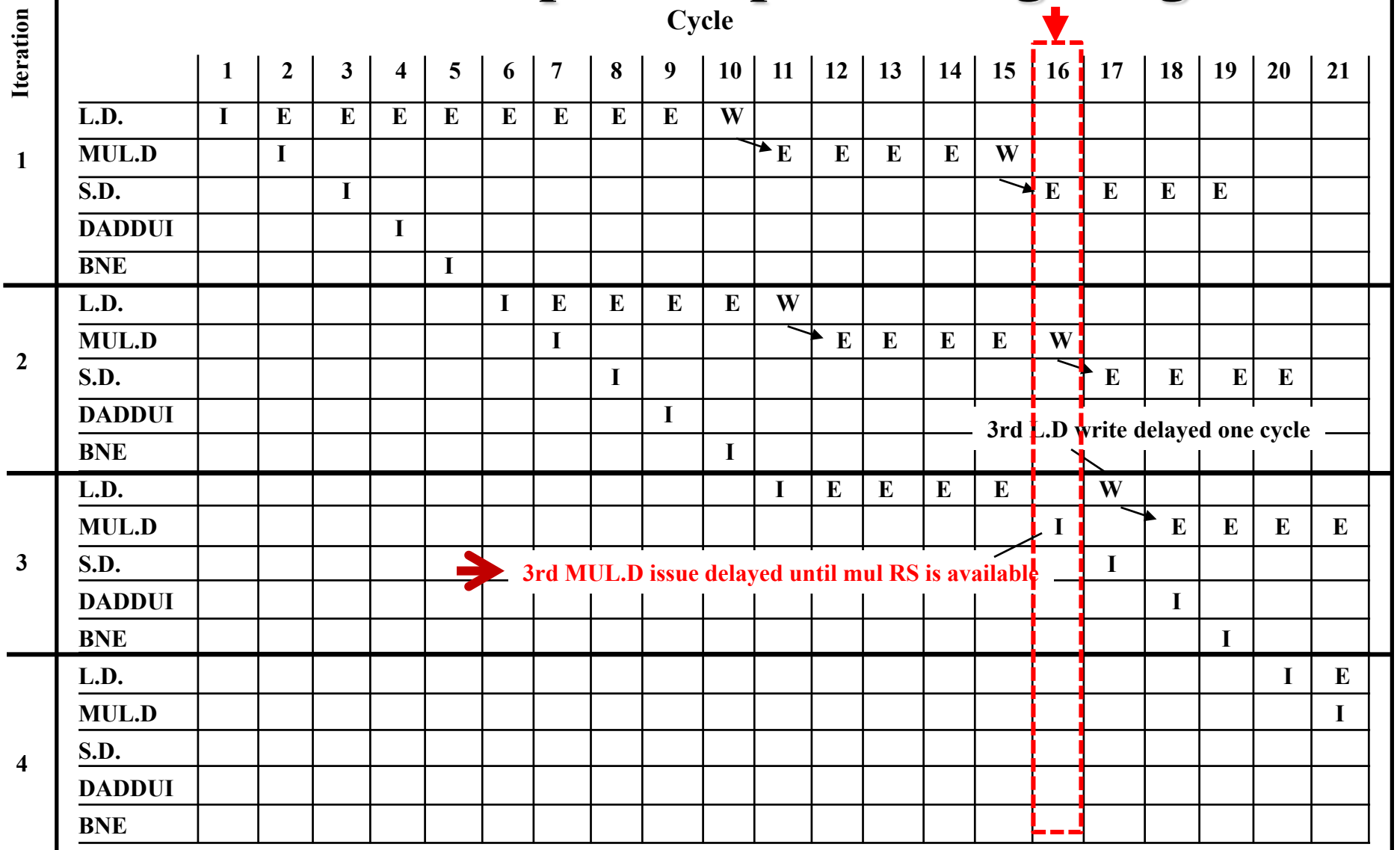
Register result status

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

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

Issue third multiply?

Tomasulo Loop Example Timing Diagram



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

Loop Example Cycle 16

End of

Instruction status

Instruction	j	k	iteration	Issue	Execution complete	Write Result
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2	14	15
S.D F4	0	R1	1	3		
L.D F0	0	R1	2	6	10	11
MUL.D F4	F0	F2	2	7	15	16
S.D F4	0	R1	2	8		

IS EX WB

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3	Yes		
Store1	Yes	80	M(80)*R(F2)
Store2	Yes	72	M(72)*R(72)
Store3	No		

WB Over CDB

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					
0	Mult1	Yes	MULTD		R(F2)	Load3	
0	Mult2	No					

Code:

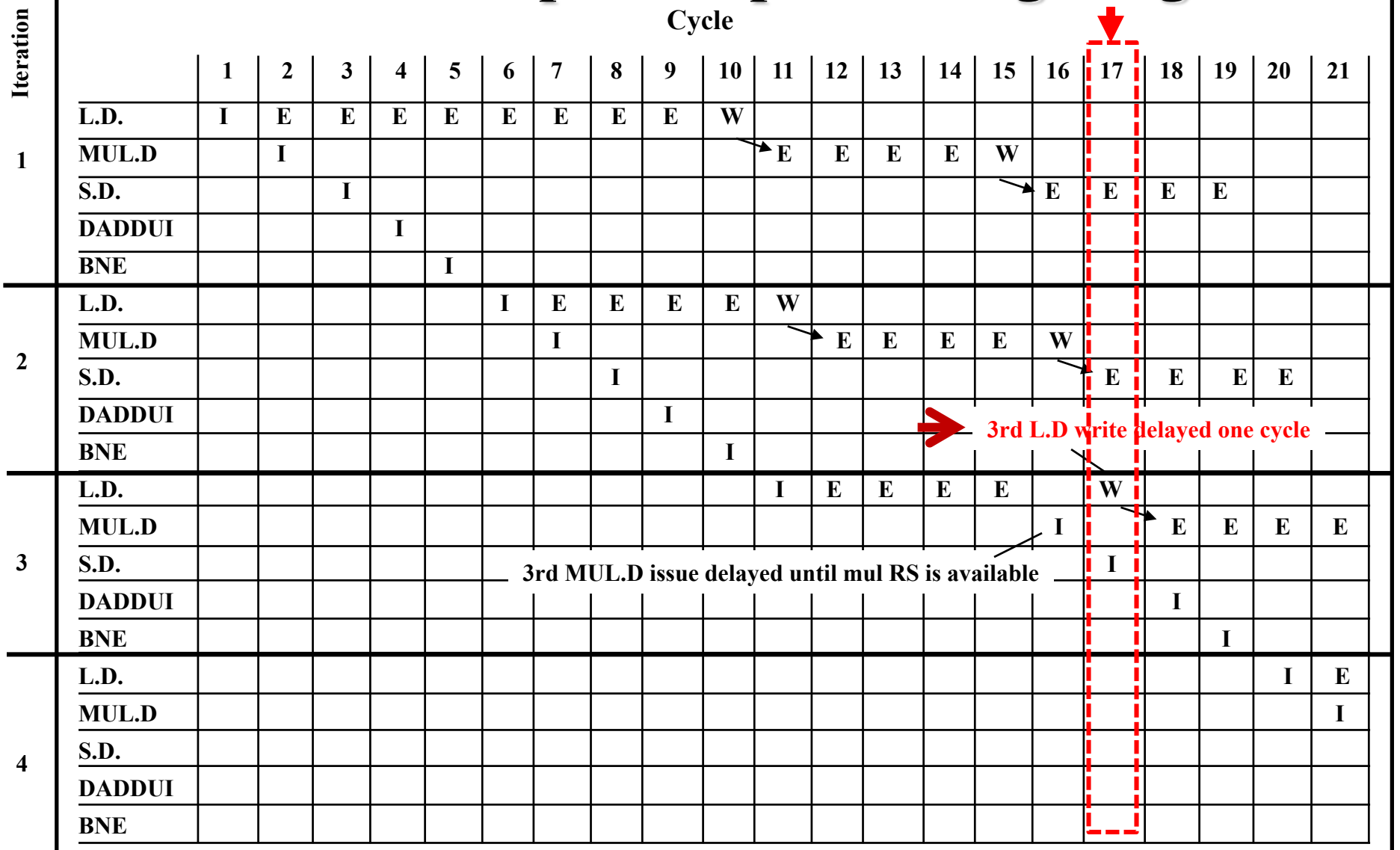
```
L.D     F0, 0(R1)
MUL.D   F4,F0,F2      Issue
S.D     F4, 0(R1)
DADDUI  R1, R1, #-8
BNE     R1,R2,loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
16	64	Qi	Load3		Mult1					

Issue third iteration MUL.D (to RS Mult1)

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 17

End of

IS EX WB

Instruction status

Instruction		j	k	iteration	Issue	Execution complete	Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2	14	15
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7	15	16
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3	No		
Store1 2	Yes	80	M(80)*R(F2)
Store2 3	Yes	72	M(72)*R(72)
Store3	Yes	64	Mult1

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					
	4	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
```

Issue

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

WB Over CDB

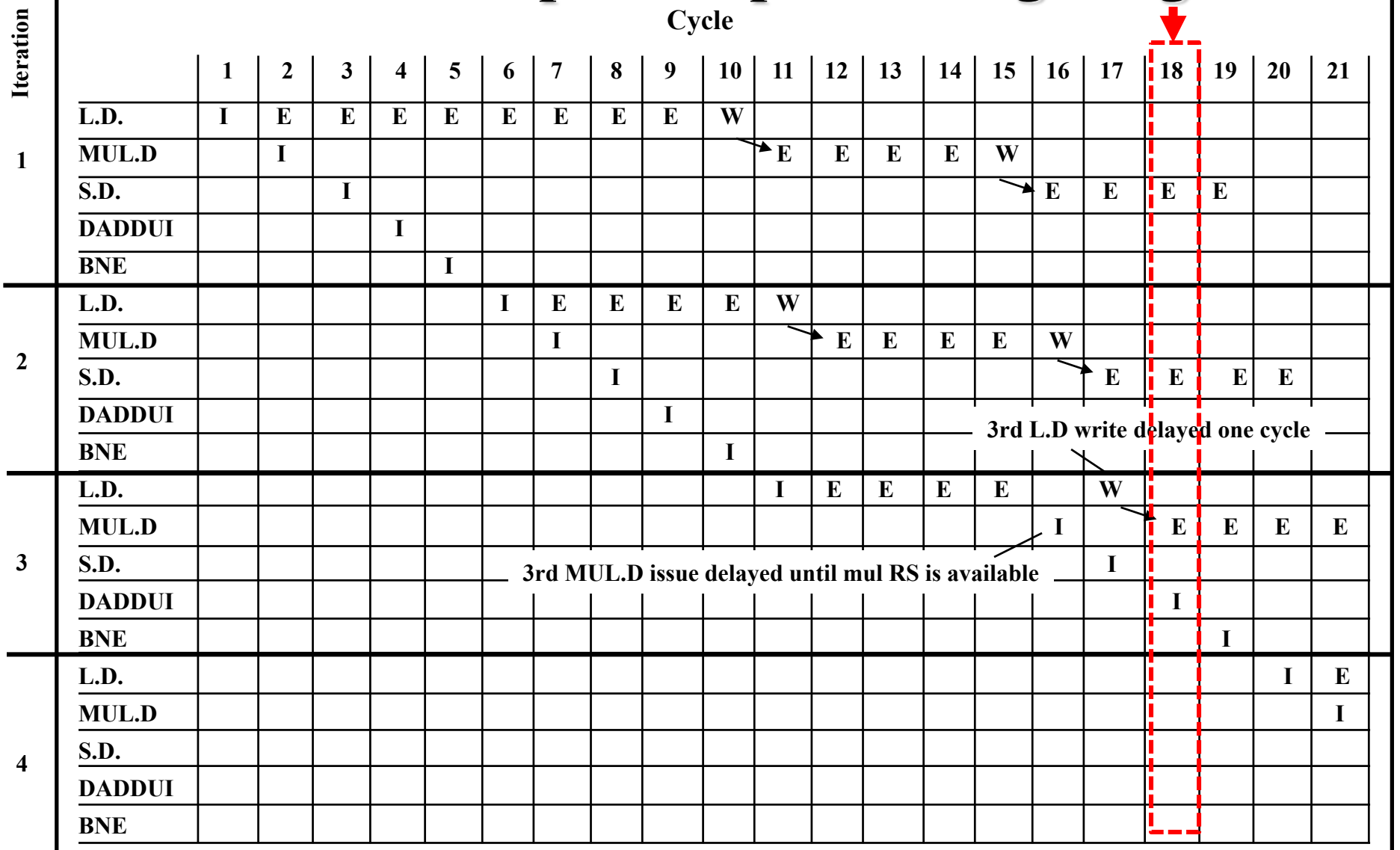
Register result status

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)

Tomasulo Loop Example Timing Diagram



Loop Example Cycle 18

End of

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2	14	15
S.D	F4	0	R1	1	3		
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7	15	16
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3	No		
Store1 1	Yes	80	M(80)*R(F2)
Store2 2	Yes	72	M(72)*R(72)
Store3	Yes	64	Mult1

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					
3	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
```

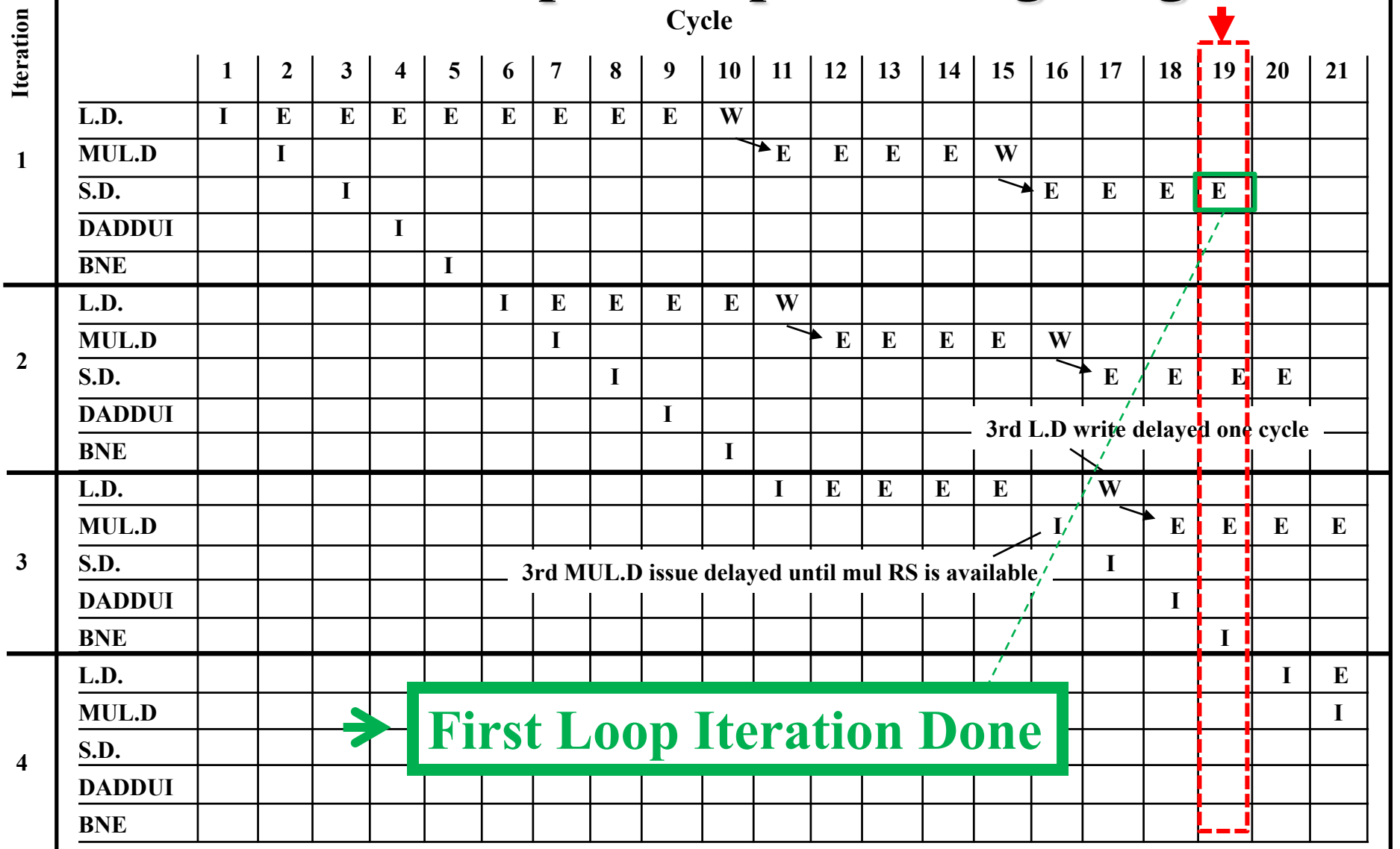
Issue

Register result status

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

➔ **Issue third iteration DADDUI**

Tomasulo Loop Example Timing Diagram



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

(First Loop Iteration Done)

Loop Example Cycle 19

End of

First Store Done

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2	14	15
S.D	F4	0	R1	1	3	19	
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7	15	16
S.D	F4	0	R1	2	8		

	Busy	Address	Qi
Load1	No		
Load2	No		
Load3	No		
Store1 0	No		
Store2 1	Yes	72	M(72)*R(72)
Store3	Yes	64	Mult1

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
```

Issue

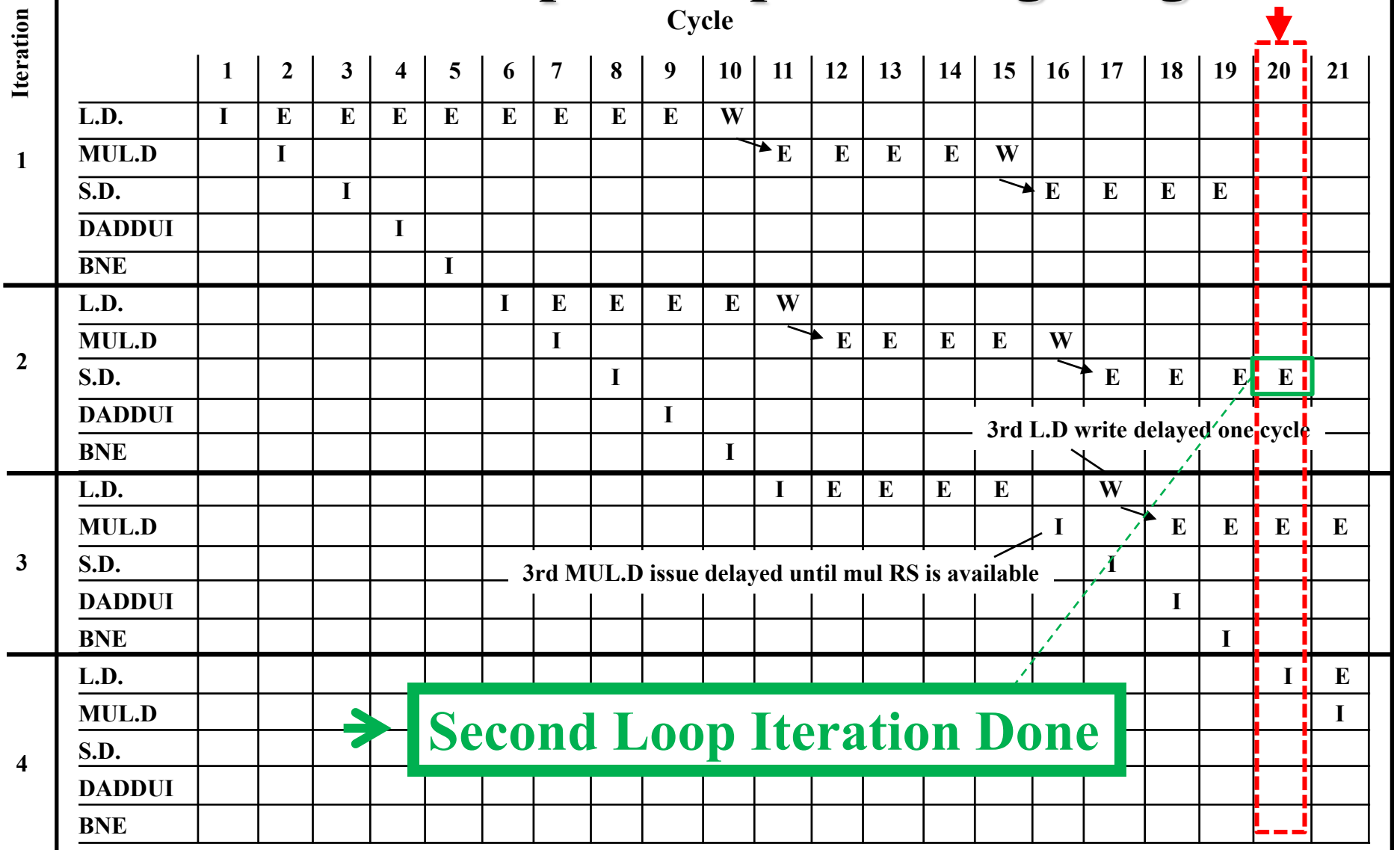
Register result status

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

Tomasulo Loop Example Timing Diagram



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

➔ (First Two Loop Iterations Done)

Loop Example Cycle 20

End of

Instruction status

Instruction		j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D	F0	0	R1	1	1	9	10
MUL.D	F4	F0	F2	1	2	14	15
S.D	F4	0	R1	1	3	19	
L.D	F0	0	R1	2	6	10	11
MUL.D	F4	F0	F2	2	7	15	16
S.D	F4	0	R1	2	8	20	

	Busy	Address	Qi
Load1 4	Yes	54	
Load2	No		
Load3	No		
Store1	No		
Store2 0	No		
Store3	Yes	64	Mult1

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					
1	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
```

Issue

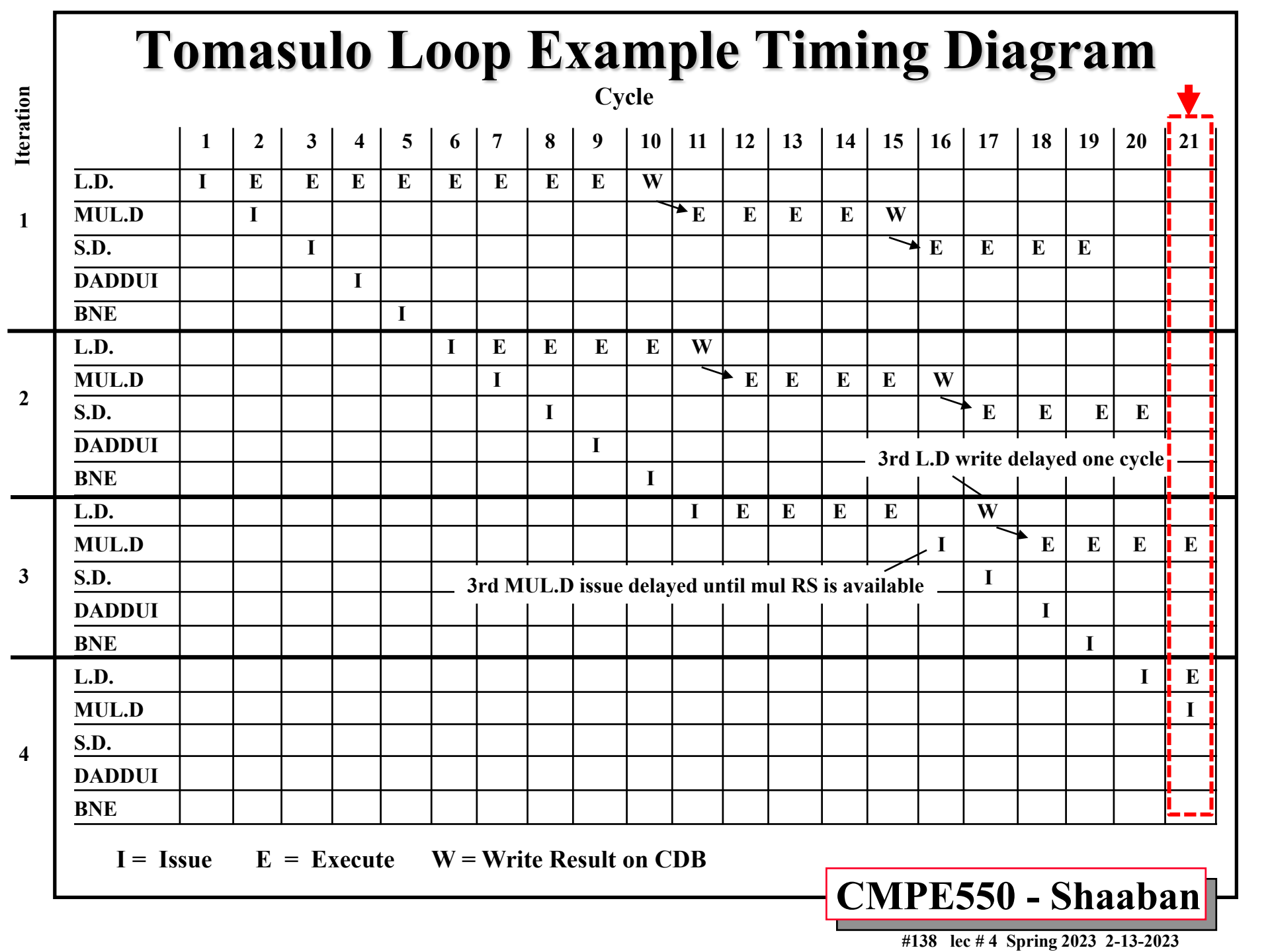
Register result status

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)

Tomasulo Loop Example Timing Diagram



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

Loop Example Cycle 21

End of

Instruction status

Instruction	j	k	iteration	IS Issue	EX Execution complete	WB Write Result
L.D F0	0	R1	1	1	9	10
MUL.D F4	F0	F2	1	2	14	15
S.D F4	0	R1	1	3	19	
L.D F0	0	R1	2	6	10	11
MUL.D F4	F0	F2	2	7	15	16
S.D F4	0	R1	2	8	20	

	Busy	Address	Qi
Load1 3	Yes	54	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

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					
0	Mult1	Yes	MULTD	M(64)	R(F2)		
0	Mult2	Yes	MULTD		R(F2)	Load1	

EX Done

Code:

```
L.D     F0, 0(R1)
MUL.D   F4,F0,F2      Issue
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
21	56	Qi	Load3		Mult2					

➔ Mult1 (third iteration MUL.D) completing; what is waiting for it?

➔ Issue fourth iteration MUL.D (to RS Mult2)

Quiz # 4

On Lecture Notes Set # 4

- **Wednesday, March 1**

- Quiz On Tomasulo Only – Not on Scoreboard (FYI)