Computer Architecture

Pipeline for Multi-cycle Operations

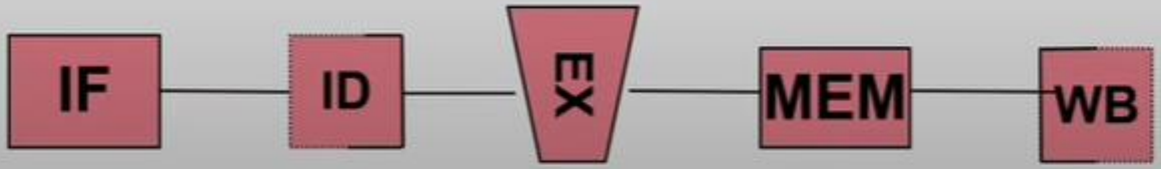
Dr. Mohammad Reza Selim

Lecture Outline

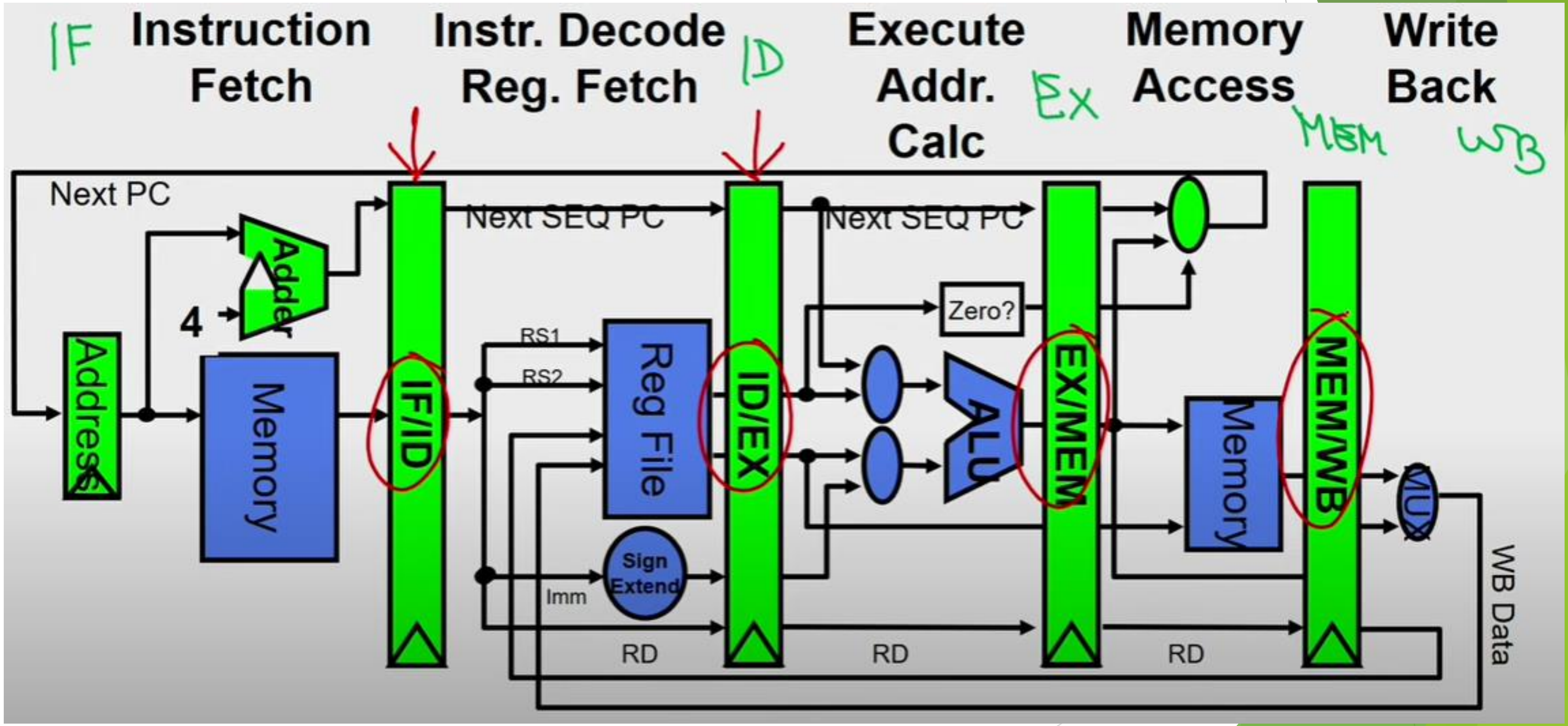
- Pipeline Review
- Multi-cycle Operations
- Issues in Longer Latency Pipelines
- How to Handle Issues in Longer Latency Pipelines

MIPS Instruction Execution Cycle

- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
- **Instruction decode/register fetch cycle (ID)**
- **Execution/Effective address cycle (EX)**
- **Memory access cycle (MEM)**
- **Write-back cycle (WB)**

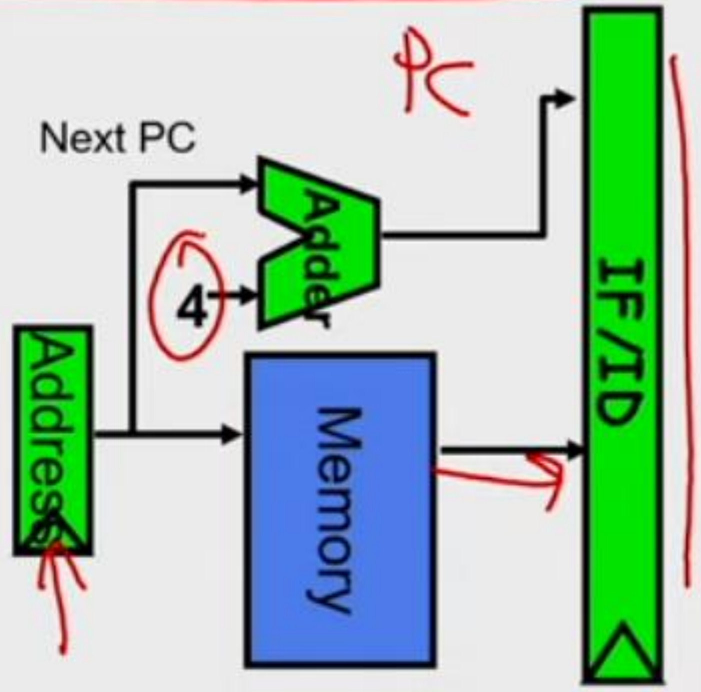


MIPS Instruction Pipeline (1)

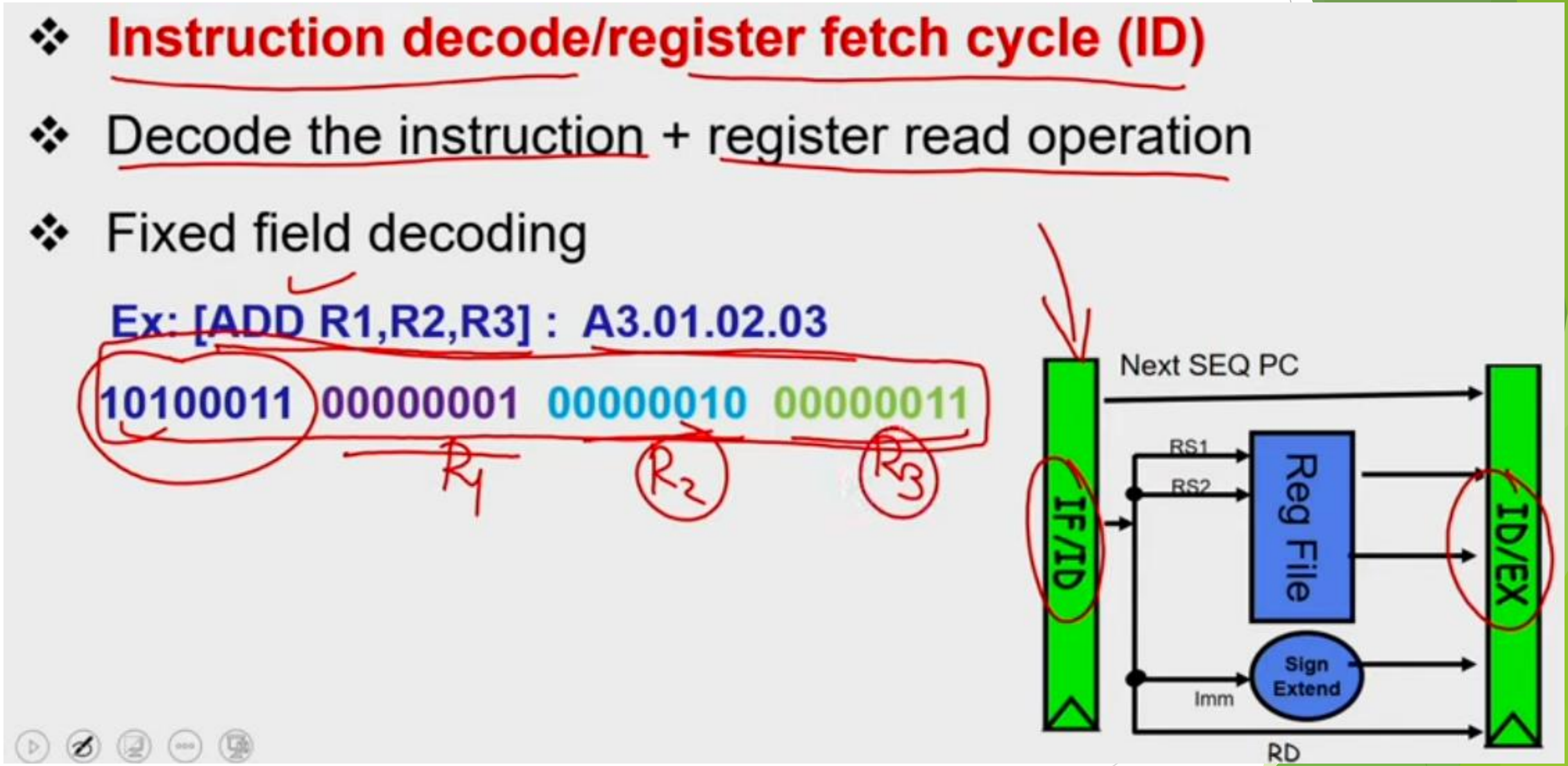


MIPS Instruction Pipeline (2)

- Each instruction can take at most 5 clock cycles
- **Instruction fetch cycle (IF)**
  - Based on PC, fetch the instruction from memory
  - Increment PC



MIPS Instruction Pipeline (3)



- **Instruction decode/register fetch cycle (ID)**
- Decode the instruction + register read operation
- Fixed field decoding

Ex: [ADD R1,R2,R3] : A3.01.02.03

10100011 00000001 00000010 00000011

R1 R2 R3

MIPS Instruction Pipeline (4)

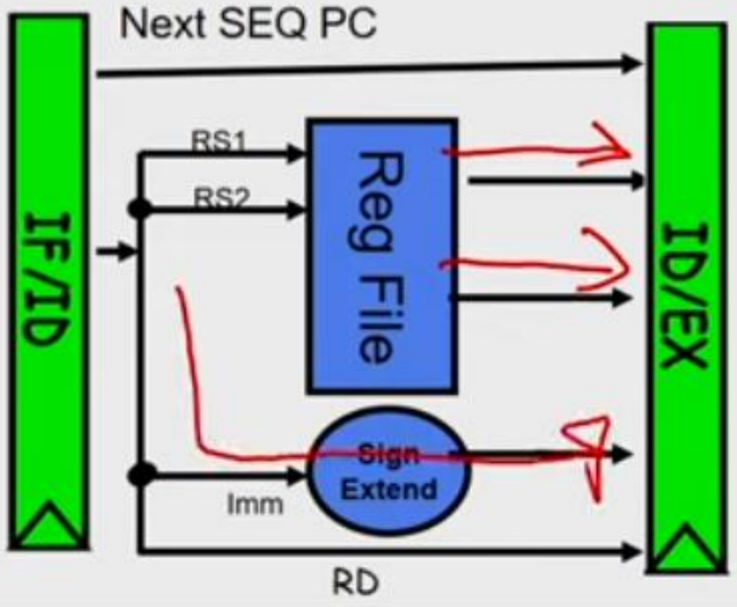
- **Instruction decode/register fetch cycle (ID)**
- Decode the instruction + register read operation
- Fixed field decoding

Ex: [ADD R1,R2,R3] : A3.01.02.03

10100011 00000001 00000010 00000011

Ex: [LW R1,8(R2)] : 86.01.02.03

10000110 00000001 00001000 00000010



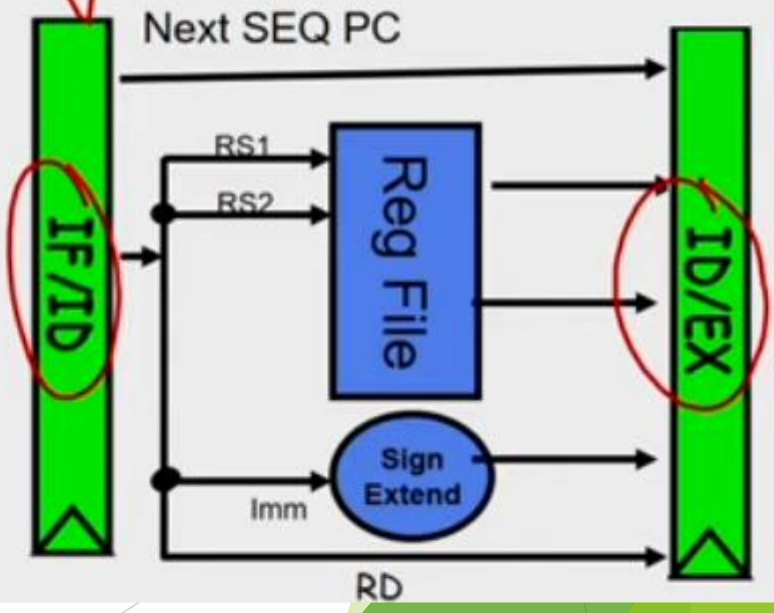
MIPS Instruction Pipeline (5)

- **Instruction decode/register fetch cycle (ID)**
- Decode the instruction + register read operation
- Fixed field decoding

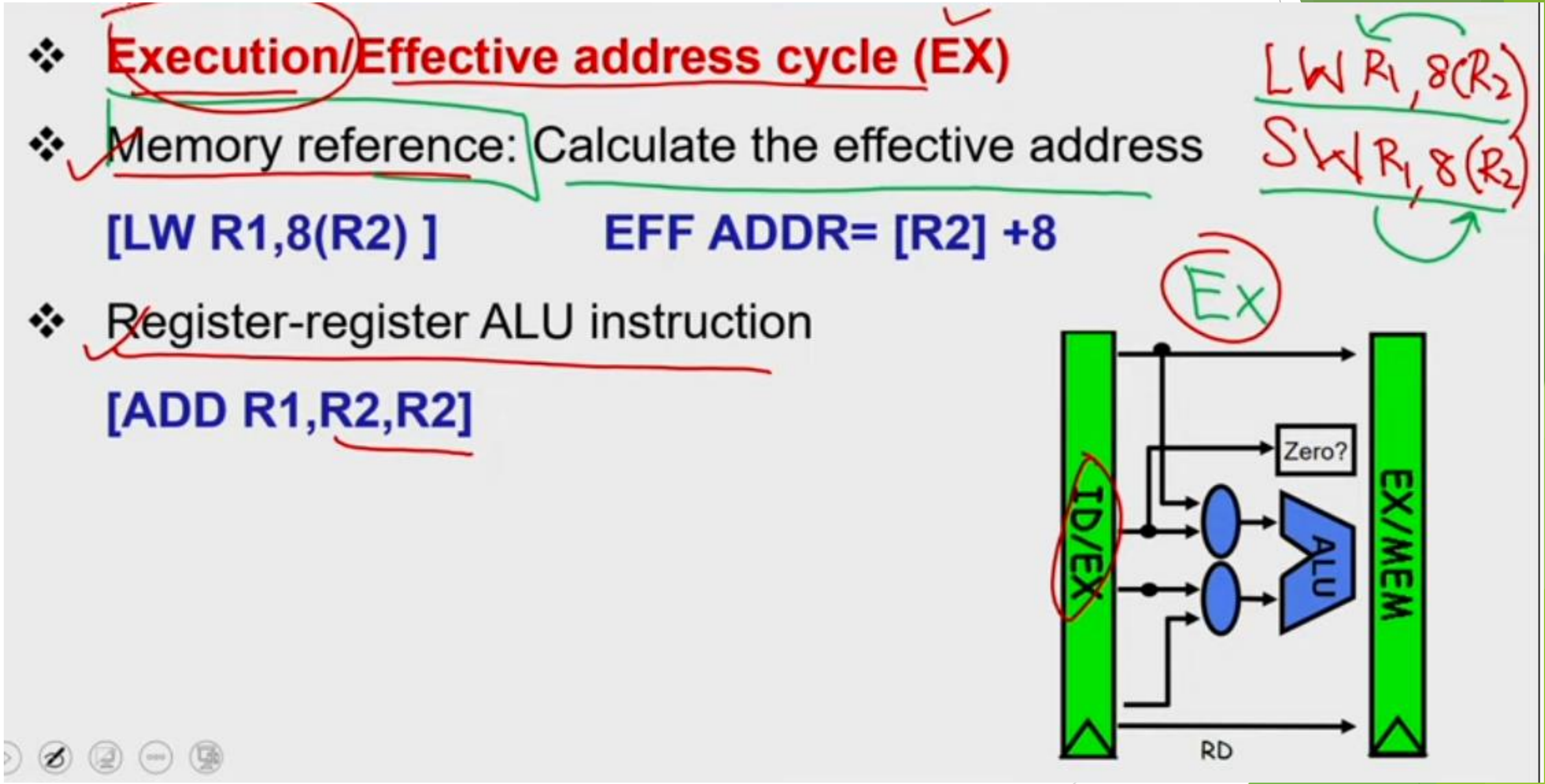
Ex: [ADD R1,R2,R3] : A3.01.02.03

10100011 00000001 00000010 00000011

R1 R2 R3



MIPS Instruction Pipeline (6)



- **Execution/Effective address cycle (EX)**
- Memory reference: Calculate the effective address

  [LW R1,8(R2)] **EFF ADDR= [R2] +8**

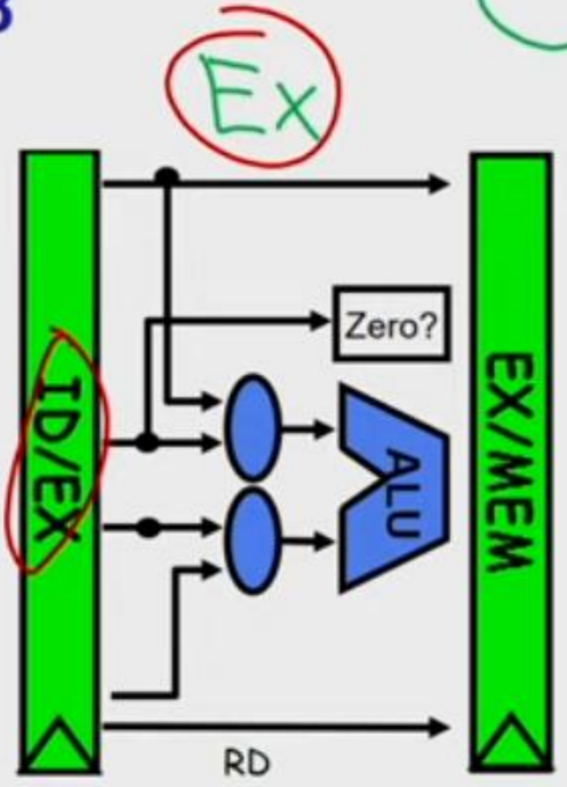
- Register-register ALU instruction

  [ADD R1,R2,R2]

LW R1,8(R2)

SW R1,8(R2)

Ex

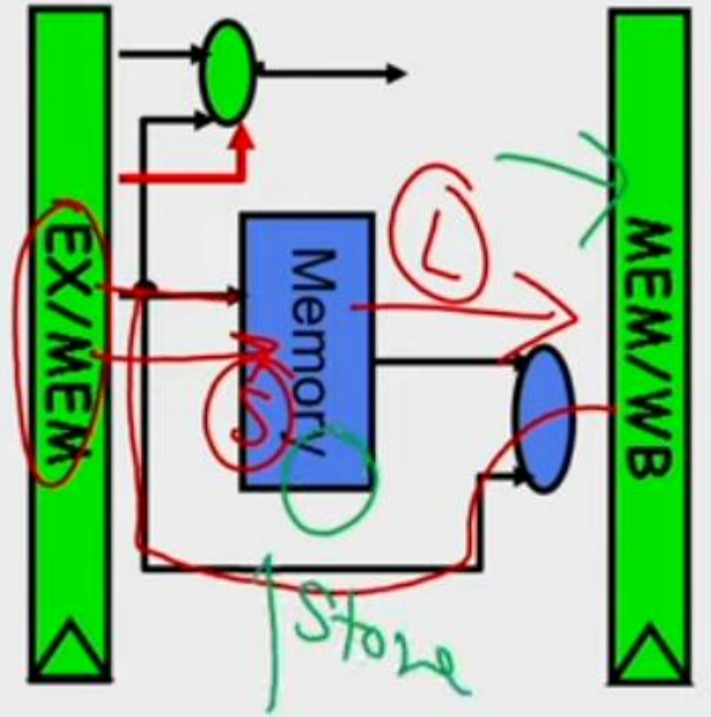


MIPS Instruction Pipeline (7)

❖ **Memory access cycle (MEM)**

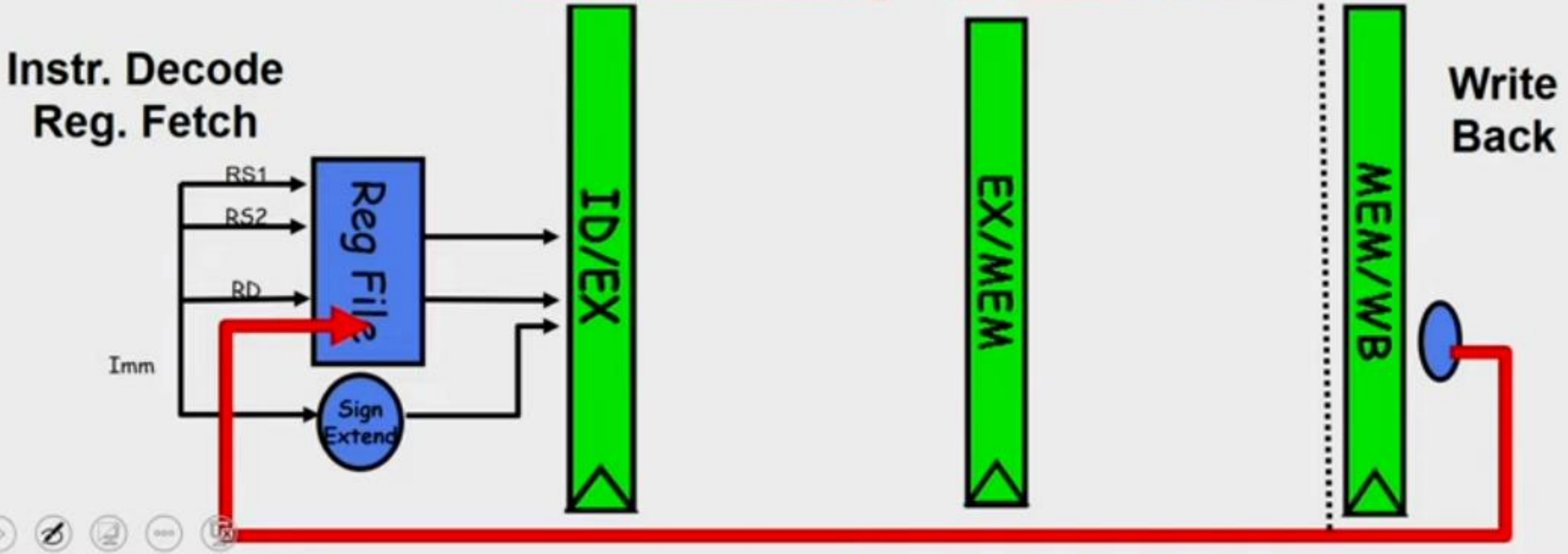
❖ Load from memory and store in register **[LW R1,8(R2)]**

❖ Store the data from the register to memory**[SW R3,16(R4)]**

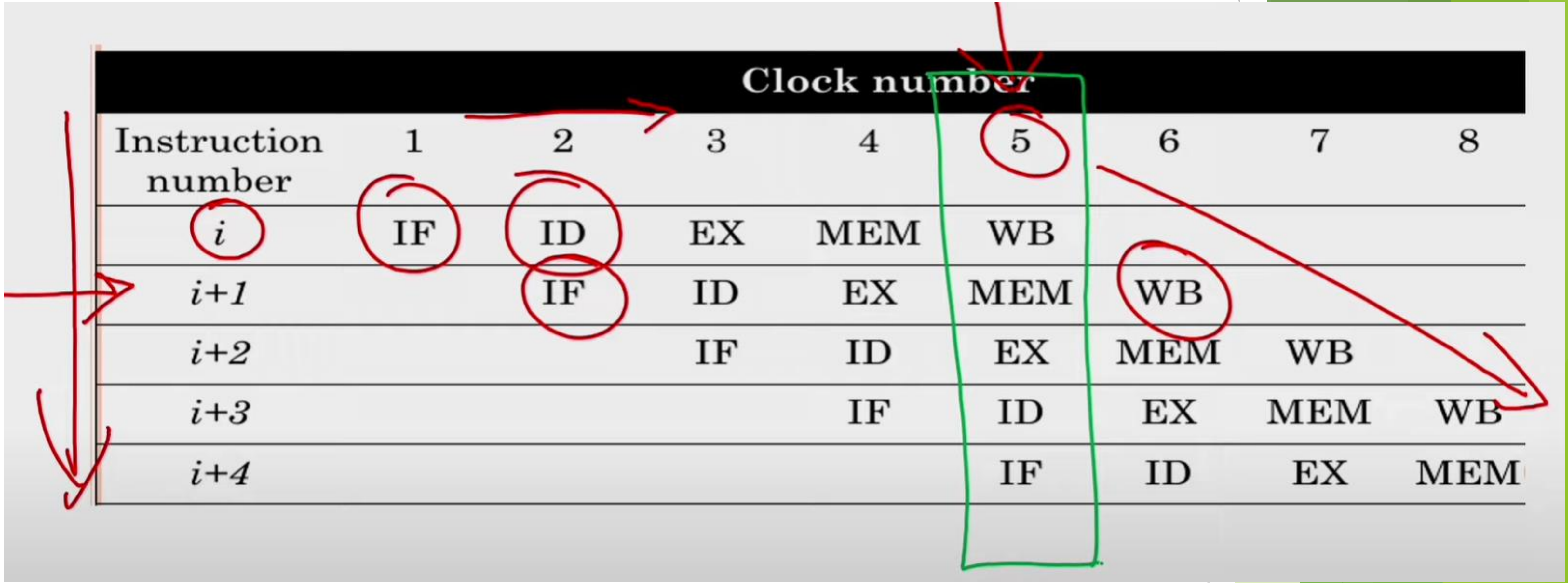


MIPS Instruction Pipeline (8)

- **Write-back cycle (WB)**
  - Register-register ALU instruction or load instruction
  - Write to register file **[LW R1,8(R2)] , [ADD R1,R2,R3]**

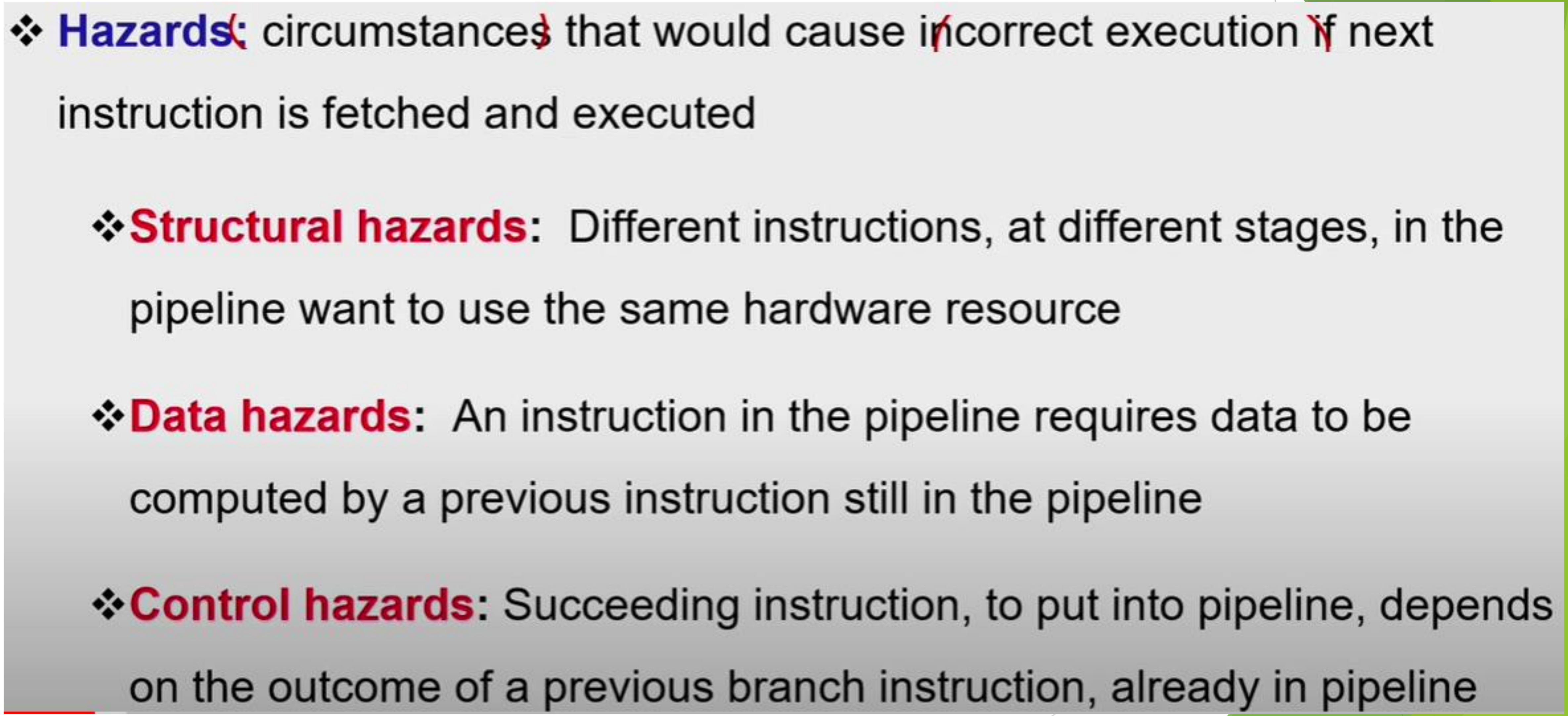


Visualizing the Pipeline



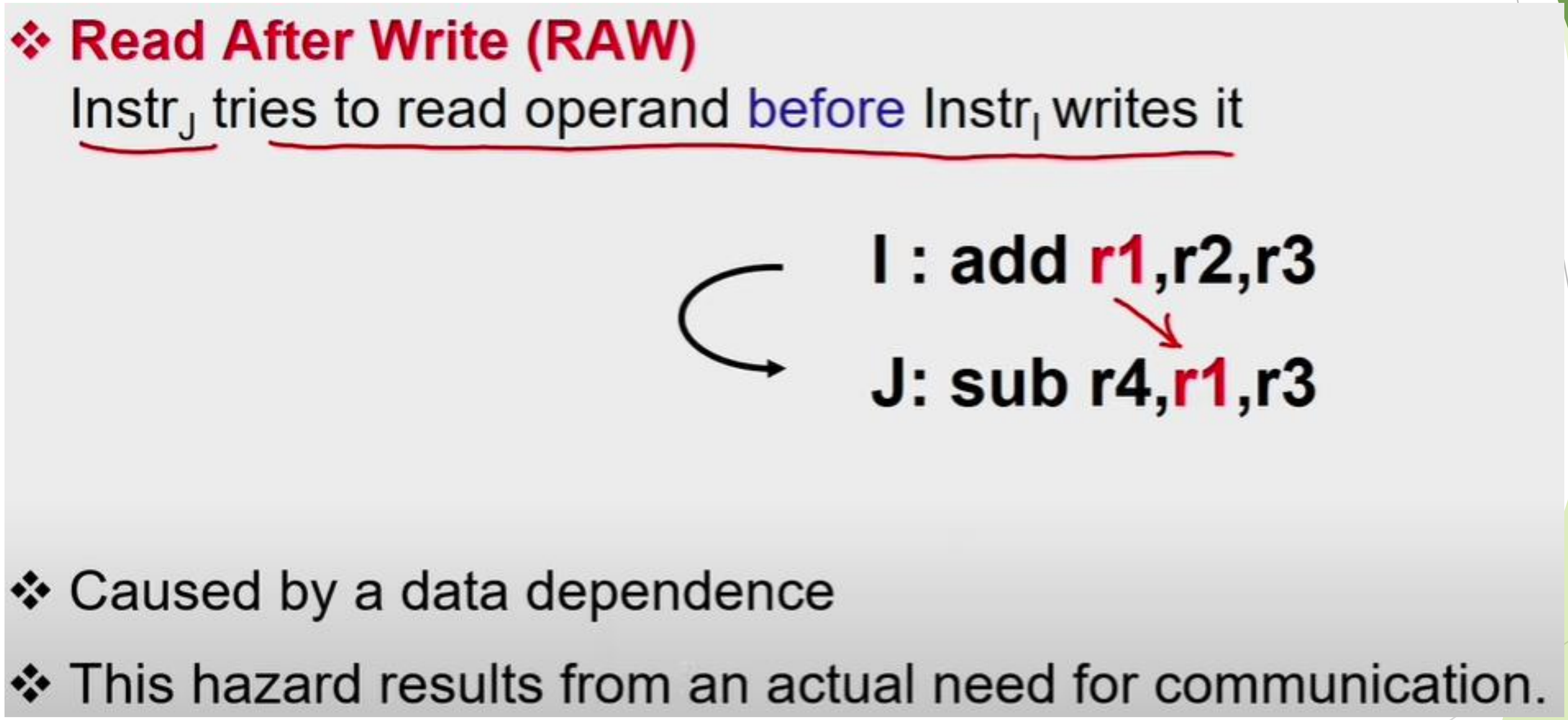
Instruction number	Clock number 1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

Pipeline Hazards



- ❖ **Hazards**: circumstances that would cause incorrect execution if next instruction is fetched and executed
  - ❖ **Structural hazards**: Different instructions, at different stages, in the pipeline want to use the same hardware resource
  - ❖ **Data hazards**: An instruction in the pipeline requires data to be computed by a previous instruction still in the pipeline
  - ❖ **Control hazards**: Succeeding instruction, to put into pipeline, depends on the outcome of a previous branch instruction, already in pipeline

Data Hazards (1)



❖ **Read After Write (RAW)**

Instr$_J$ tries to read operand before Instr$_I$ writes it

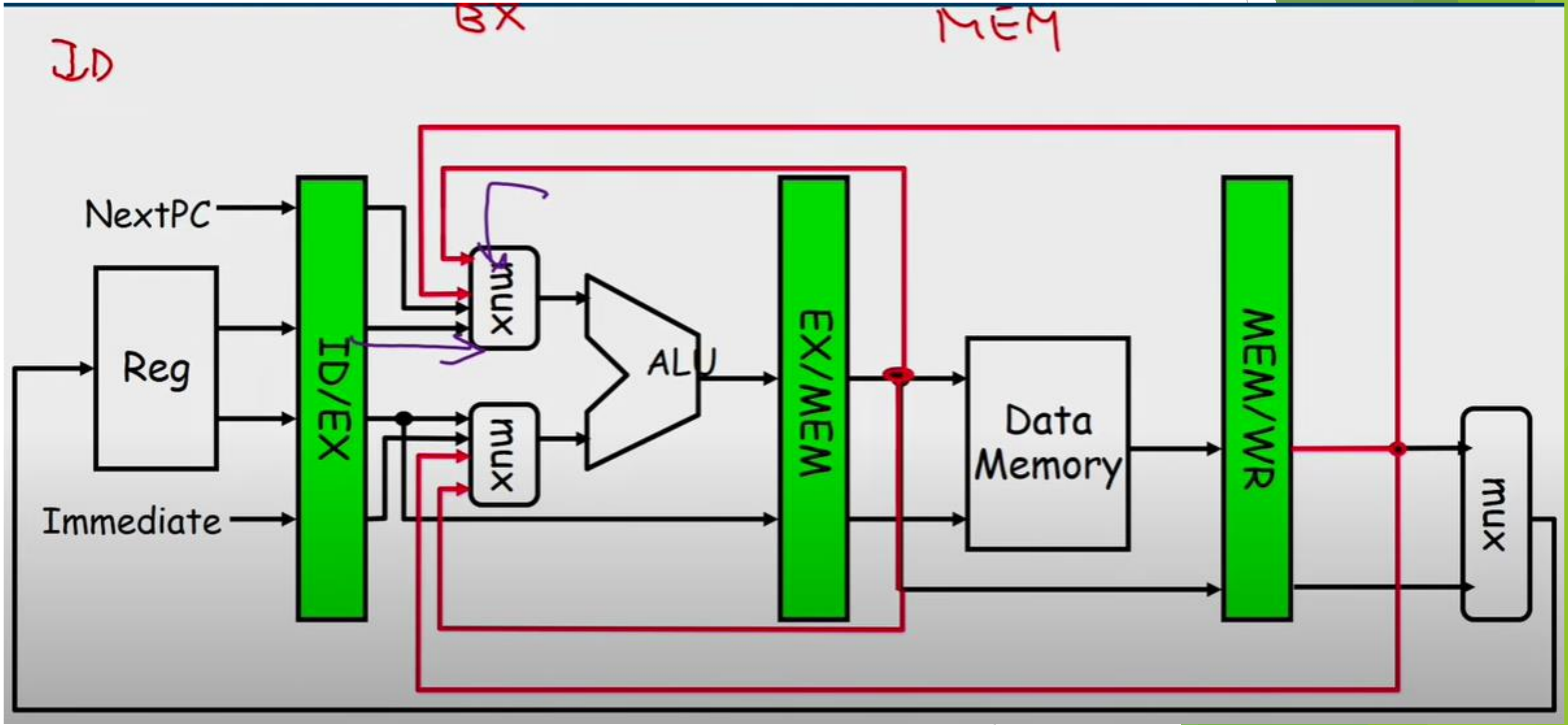
I : add r1,r2,r3

J: sub r4,r1,r3

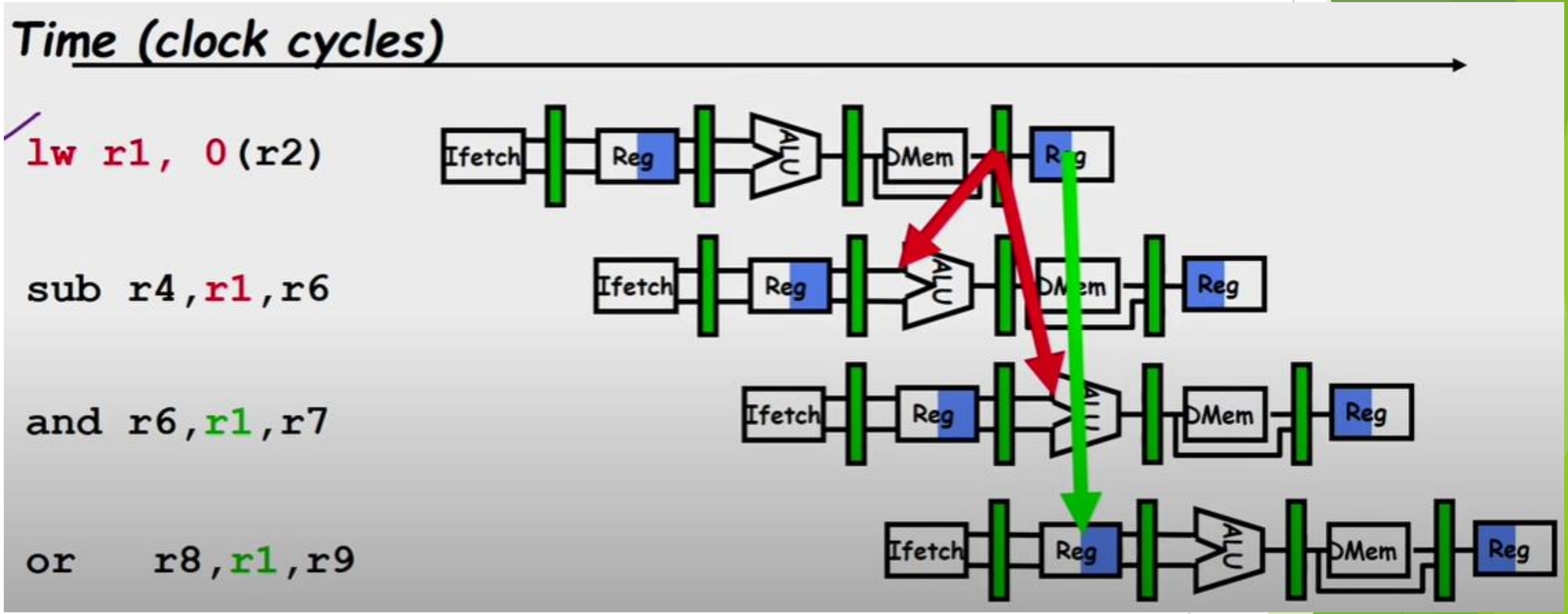
❖ Caused by a data dependence

❖ This hazard results from an actual need for communication.

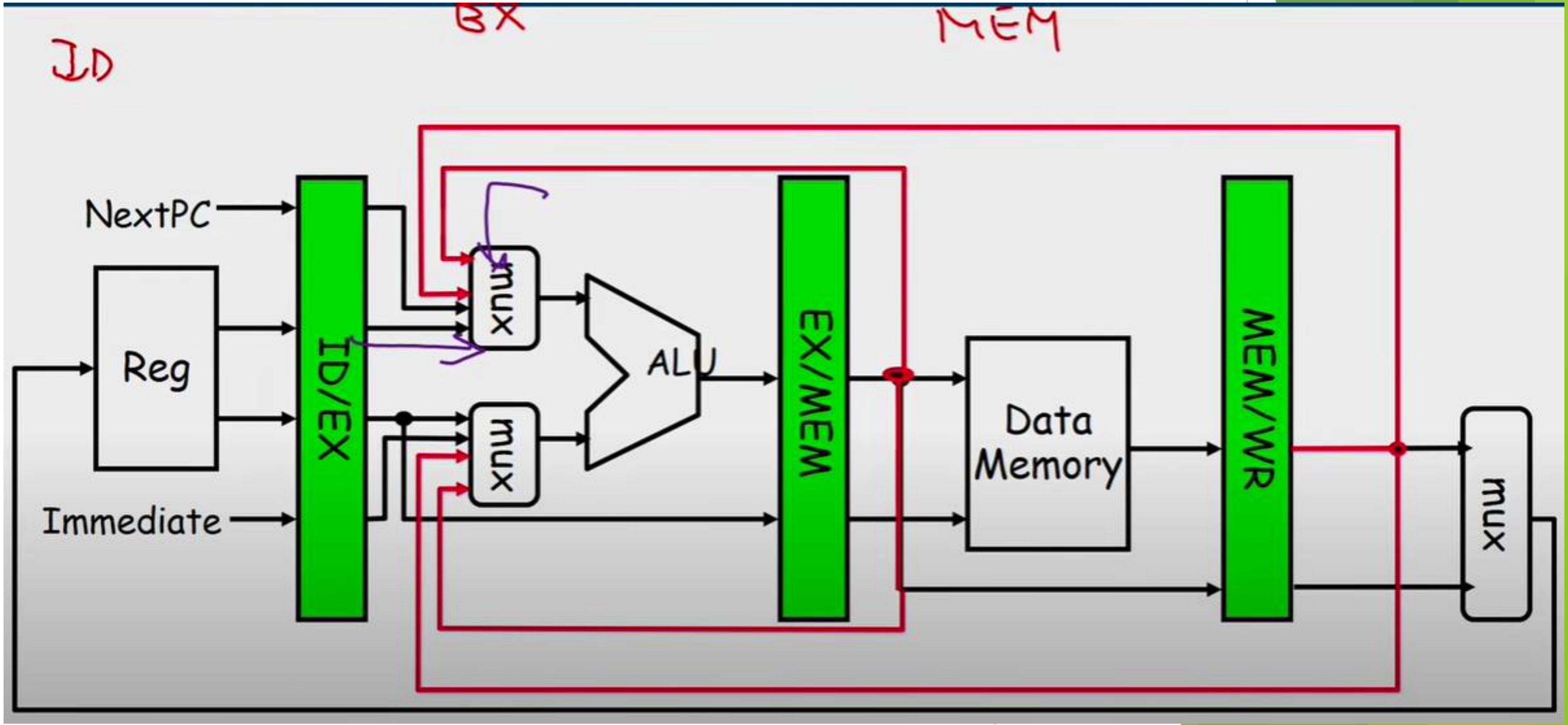
Data Hazards (2)
Hardware Change for Operand Forwarding



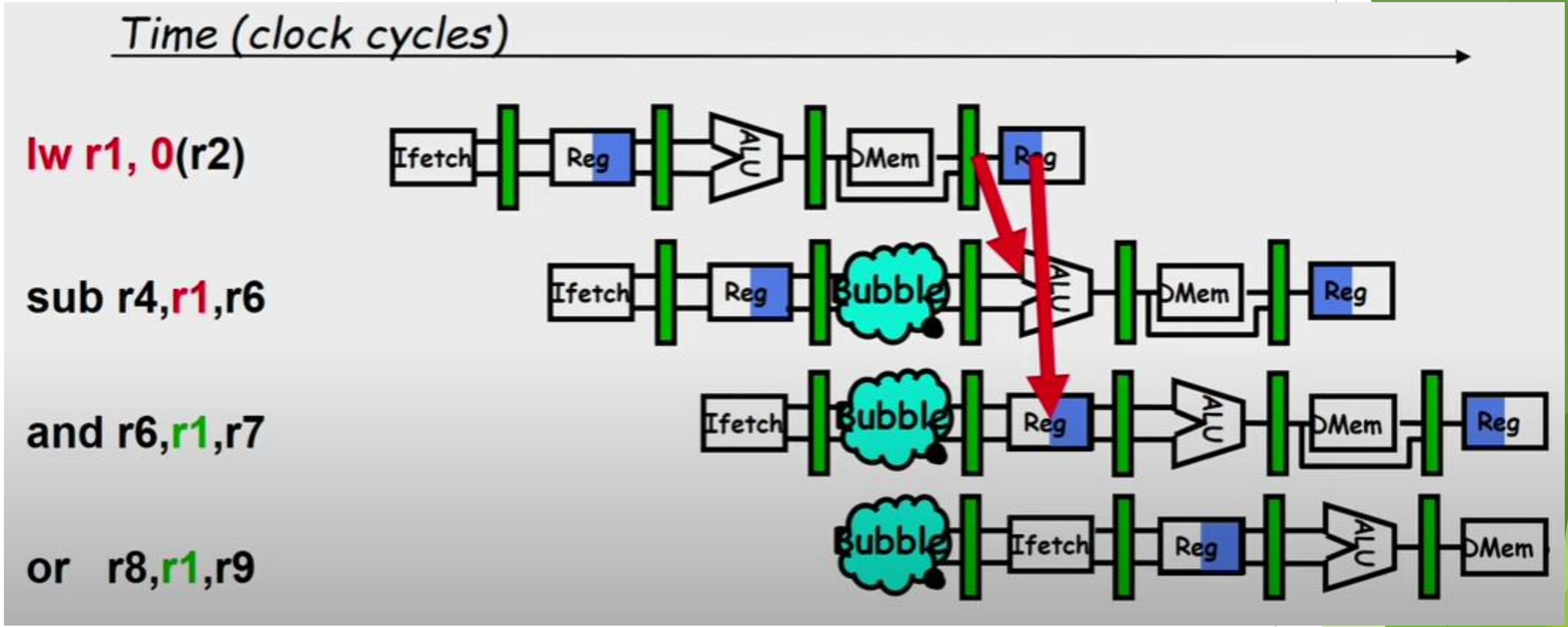
Data Hazards (3)
Data Hazards even with Operand Forwarding



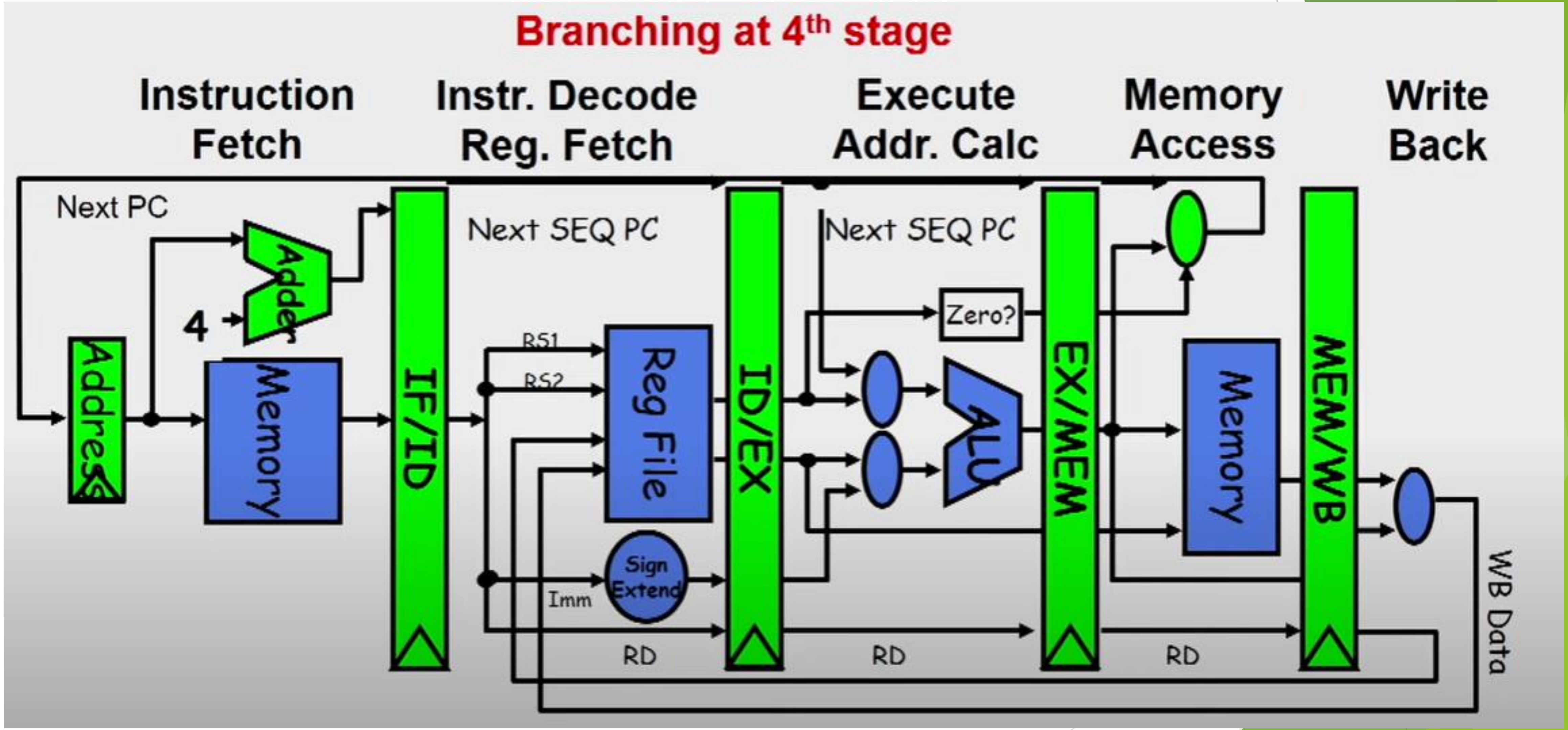
Data Hazards (4)
Hardware Change for Operand Forwarding



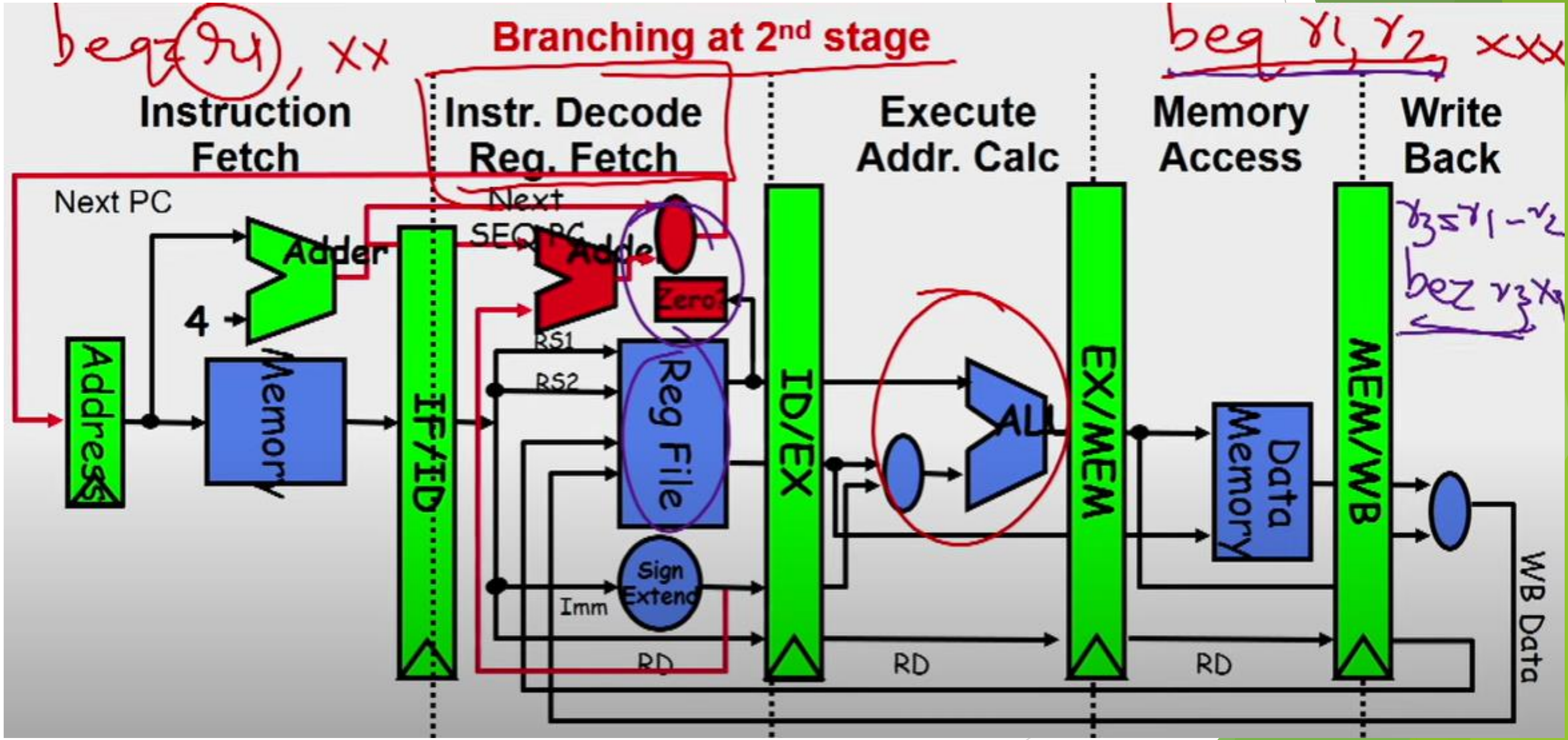
Data Hazards (5)
Resolving the Load-ALU Hazard



Control Hazards(1)
Conventional MIPS Pipeline

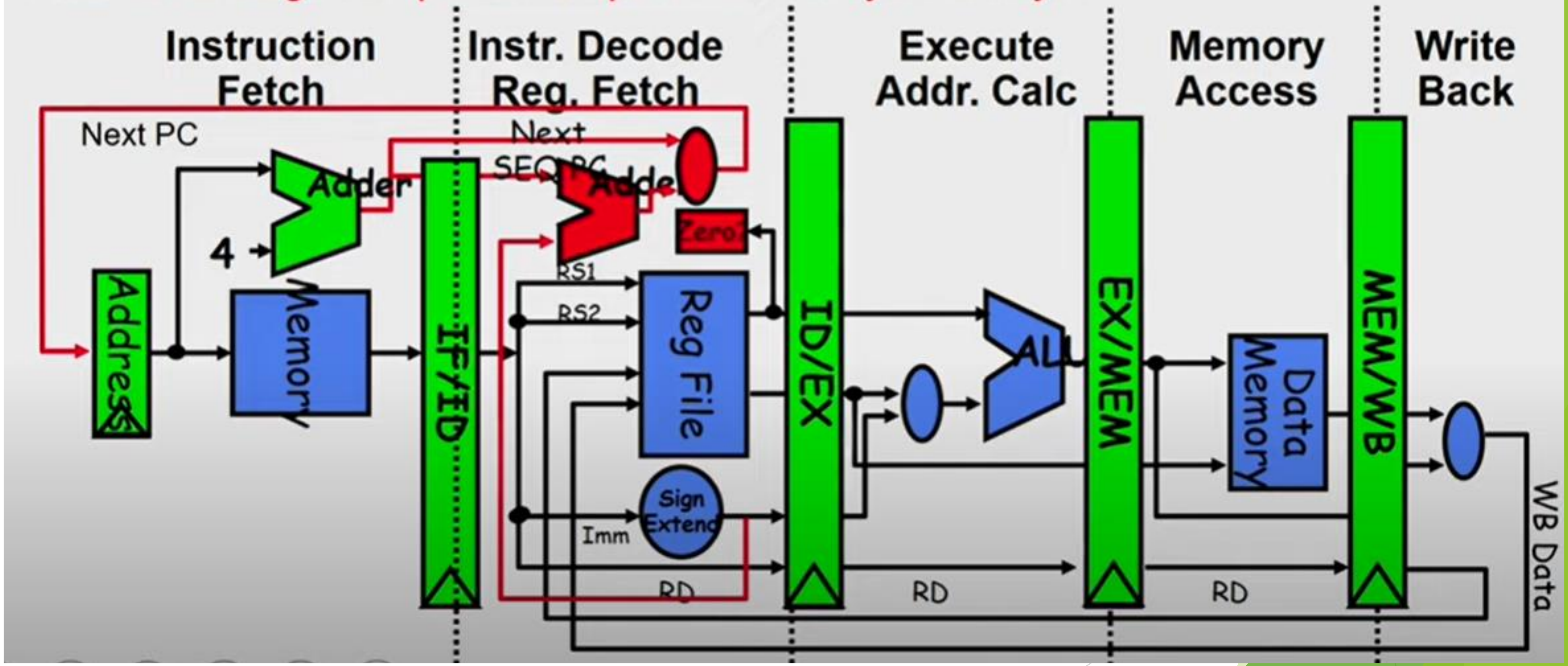


Control Hazards(2)
Branch Optimized MIPS Pipeline

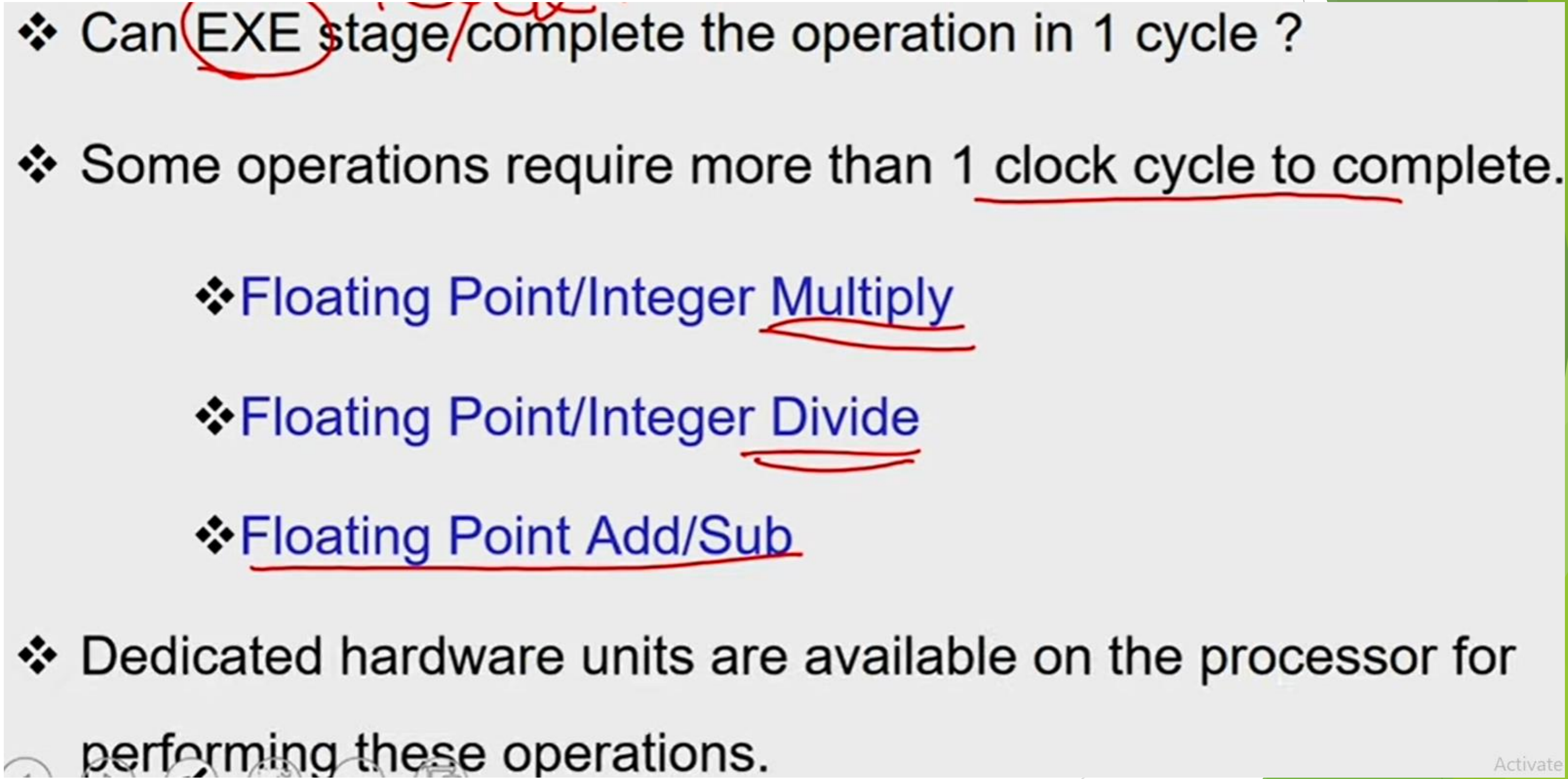


Multi-cycle Operations (1)

❖Can EXE stage complete the operation in 1 cycle always?



Multi-cycle Operations (2)



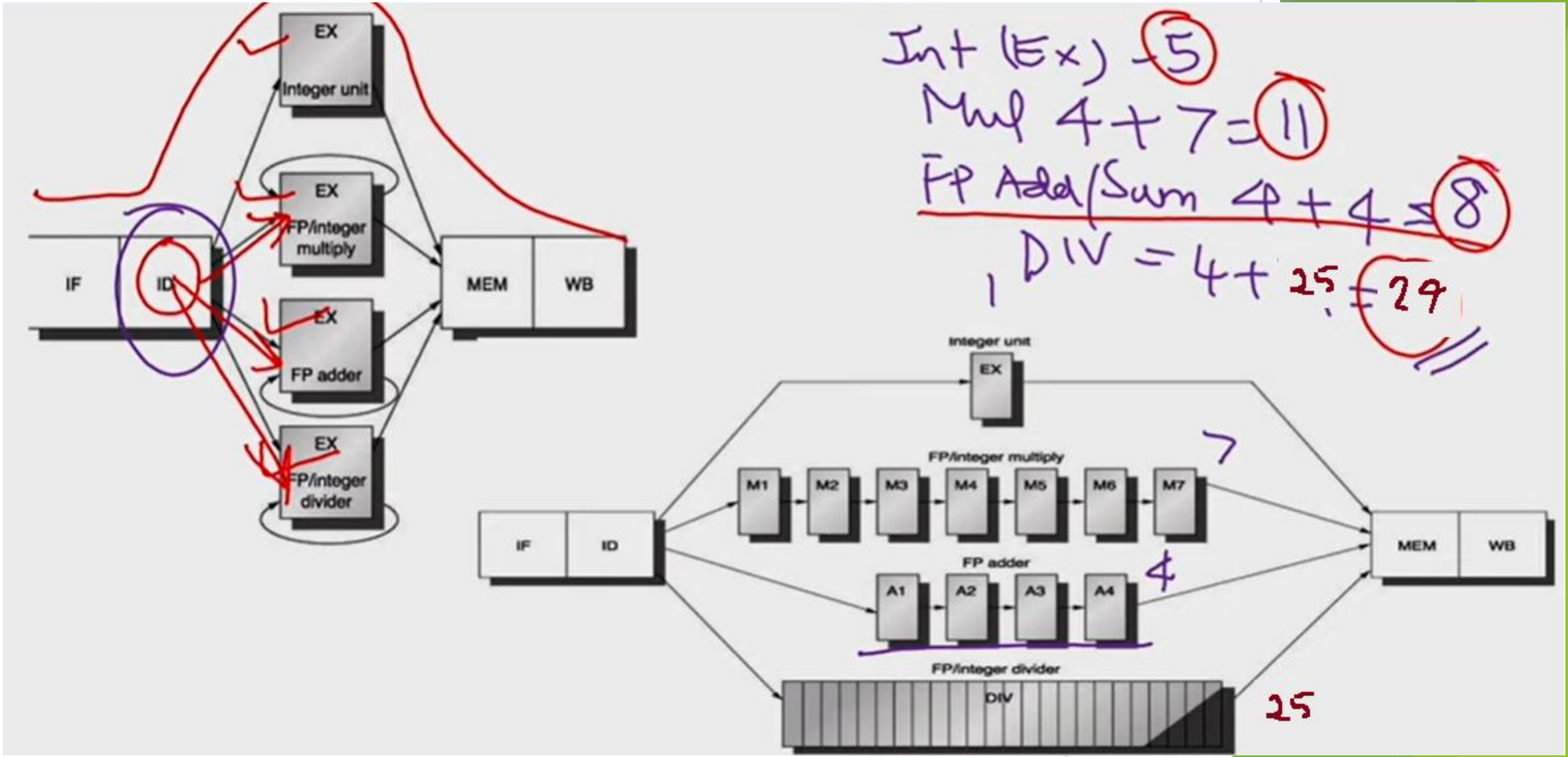
❖ Can EXE stage complete the operation in 1 cycle ?

❖ Some operations require more than 1 clock cycle to complete.

- ❖ Floating Point/Integer Multiply
- ❖ Floating Point/Integer Divide
- ❖ Floating Point Add/Sub

❖ Dedicated hardware units are available on the processor for performing these operations.

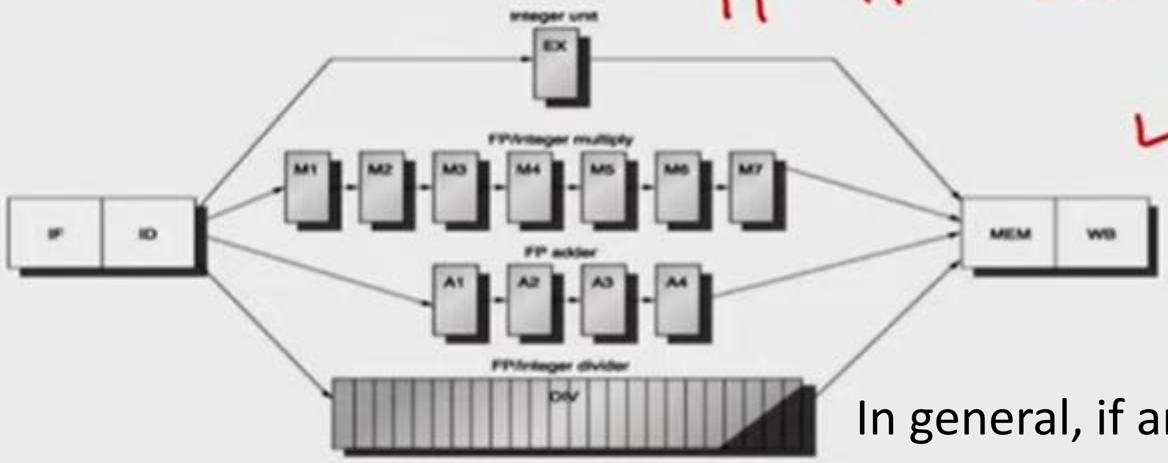
Multi-cycle Operations (3)



Multi-cycle Operations (4)

IF ID EX M W

IF ID EX M W



Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

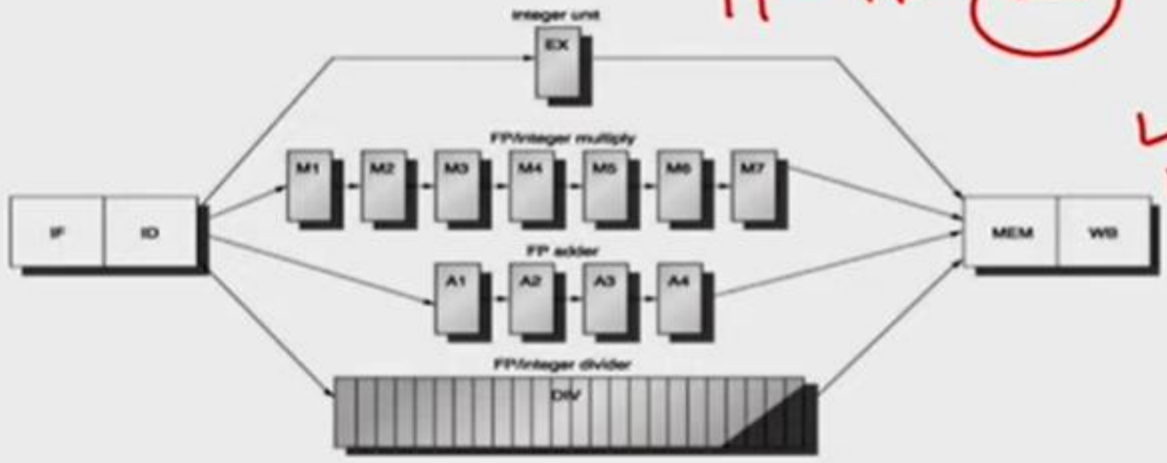
In general, if an operation requires ***N*** execution cycles, latency in ***N*-1** and initiation interval is **1** if the unit is pipelined and ***N*** if not pipelined.

- **Latency:** The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- **Initiation / Repeat Interval:** The number of cycles that must elapse between issuing two operations of a given type.

Multi-cycle Operations (5)

IF ID Ex M W

IF ID − Ex M W

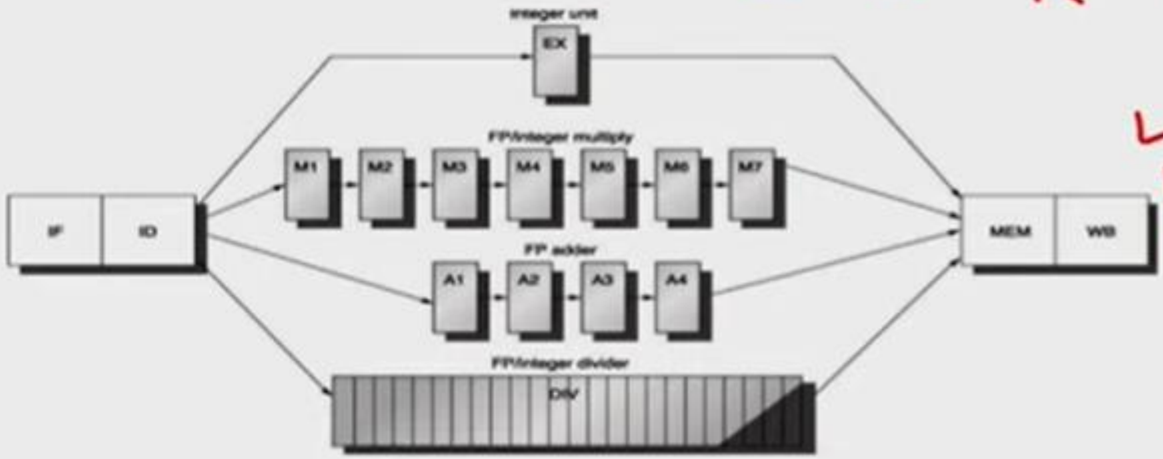


Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

- **Latency:** The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- **Initiation / Repeat Interval:** The number of cycles that must elapse between issuing two operations of a given type.

Multi-cycle Operations (6)

```
IF   ID   A1   A2   A3   A4 | M    W
     IF   ID   X    X    X    Ex   M    W
```

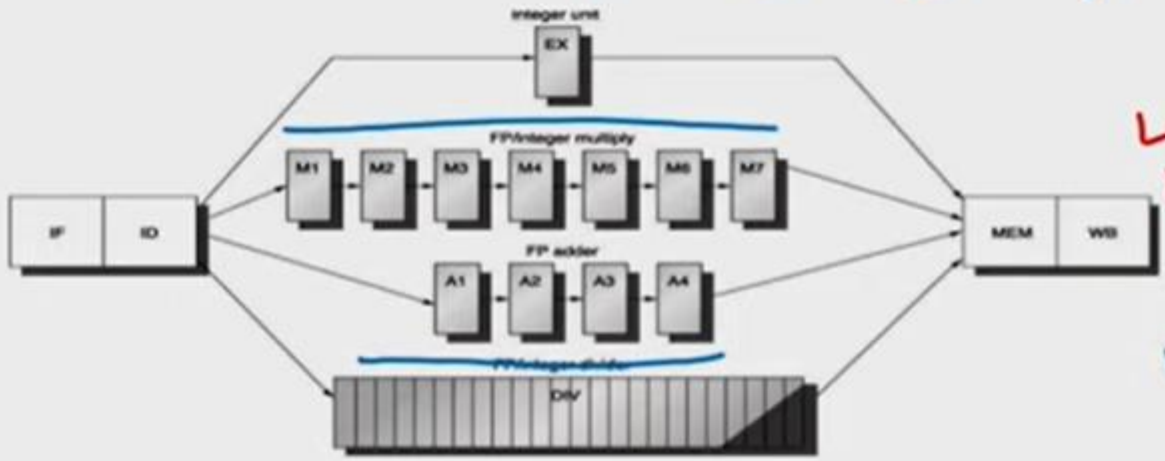


Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

- **Latency:** The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- **Initiation / Repeat Interval:** The number of cycles that must elapse between issuing two operations of a given type.

Multi-cycle Operations (7)

IF	ID	M1	M2	M3	M4	M5	M6	M7	M	W	
	IF	ID	x	x	x	x	x	x	EX	M	W

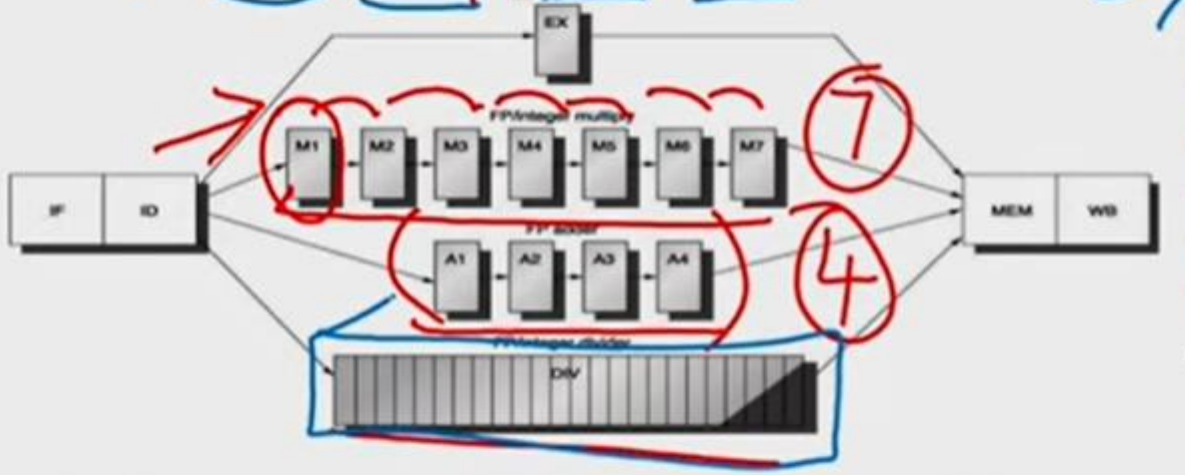


Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

- **Latency:** The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- **Initiation / Repeat Interval:** The number of cycles that must elapse between issuing two operations of a given type.

Multi-cycle Operations (8)

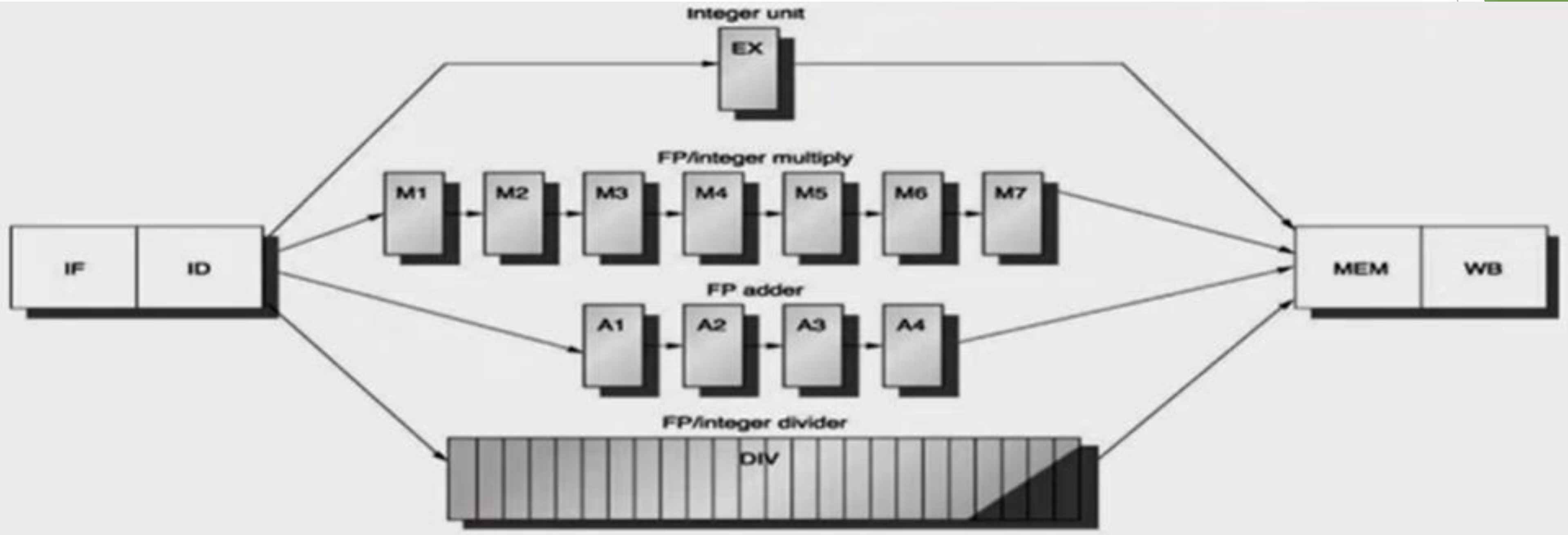
(DIV R1, R2, R3
DIV R4, R5, R6) — 24 25



Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

- **Latency:** The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
- **Initiation / Repeat Interval:** The number of cycles that must elapse between issuing two operations of a given type.

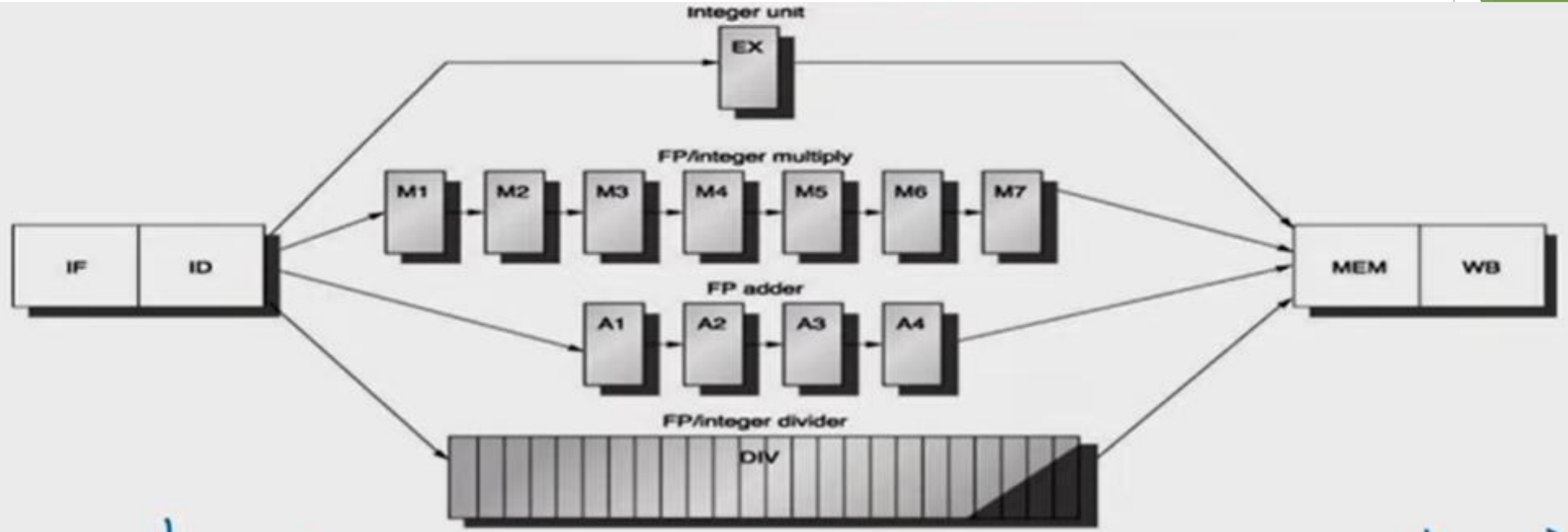
Multi-cycle Operations (9)



MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

The pipeline timing of a set of independent FP operations. The stages in italics show where data are needed, while the stages in bold show where a result is available. The ".D" extension on the instruction mnemonic indicates double-precision (64-bit) floating-point operations. FP loads and stores use a 64-bit path to memory so that the pipelining timing is just like an integer load or store.

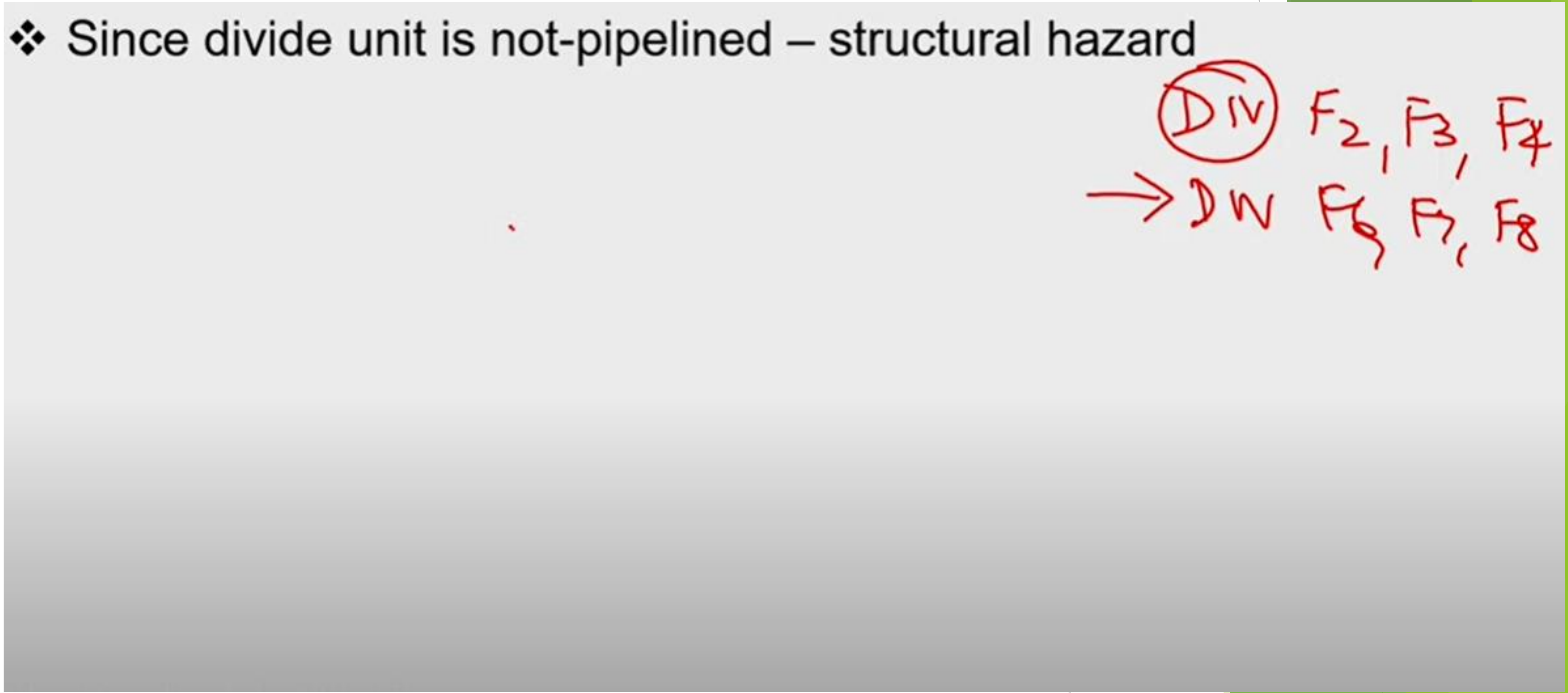
Multi-cycle Operations (10)



MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	M7	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	A4	MEM	WB		
L.D			IF	ID	*EX*	MEM	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

The pipeline timing of a set of independent FP operations. The stages in italics show where data are needed, while the stages in bold show where a result is available. The ".D" extension on the instruction mnemonic indicates double-precision (64-bit) floating-point operations. FP loads and stores use a 64-bit path to memory so that the pipelining timing is just like an integer load or store.

Issues in Longer Latency Pipelines (1)

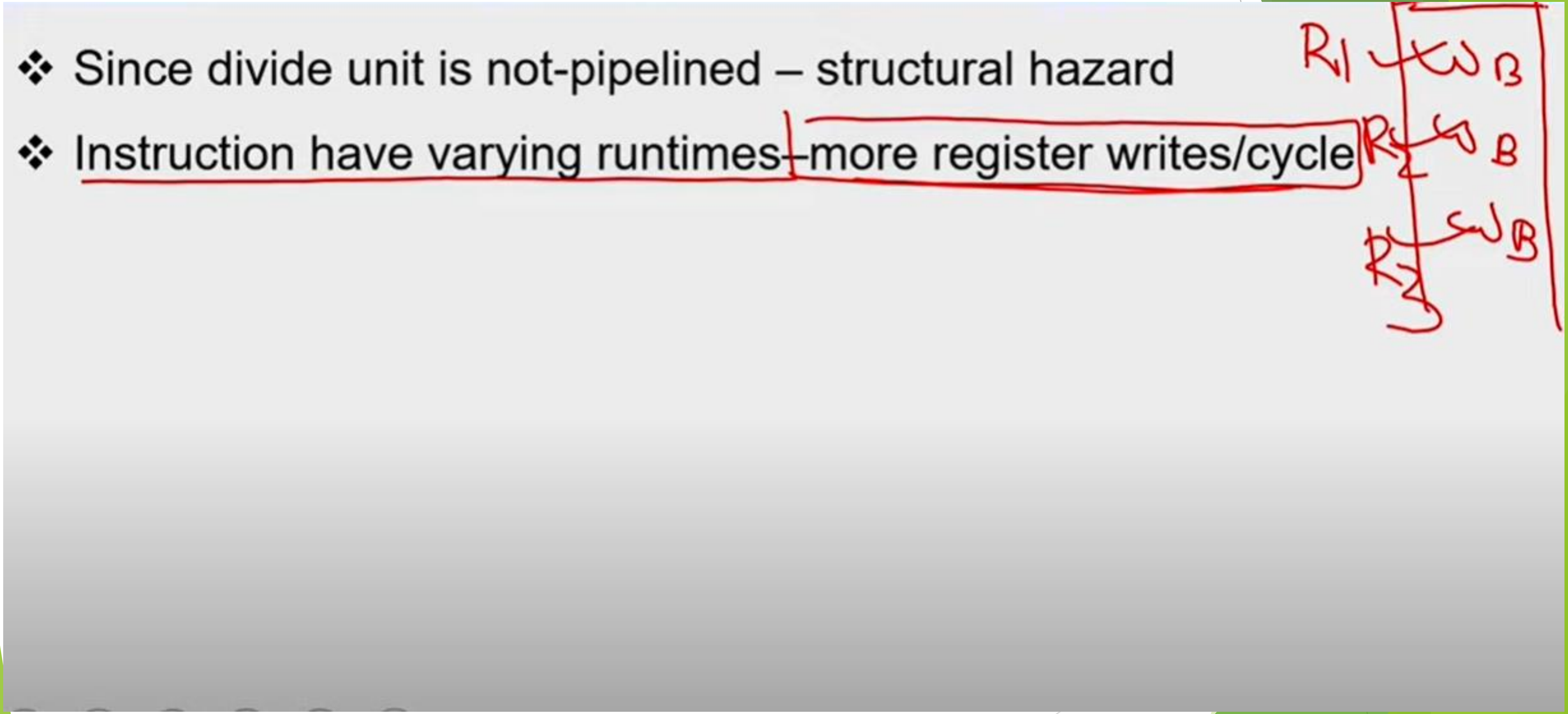


❖ Since divide unit is not-pipelined – structural hazard

DIV F2, F3, F4

→ DIV F6, F7, F8

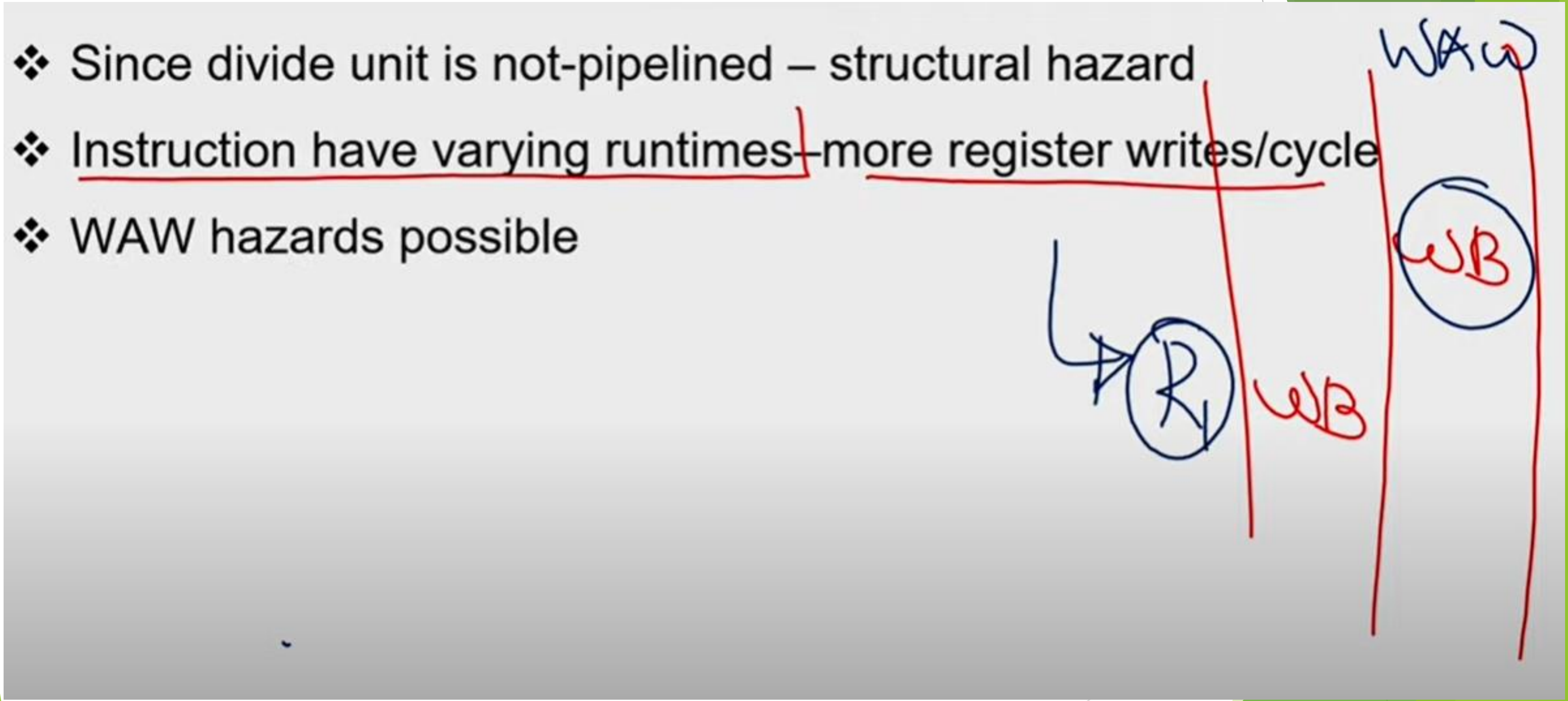
Issues in Longer Latency Pipelines (3)



❖ Since divide unit is not-pipelined – structural hazard

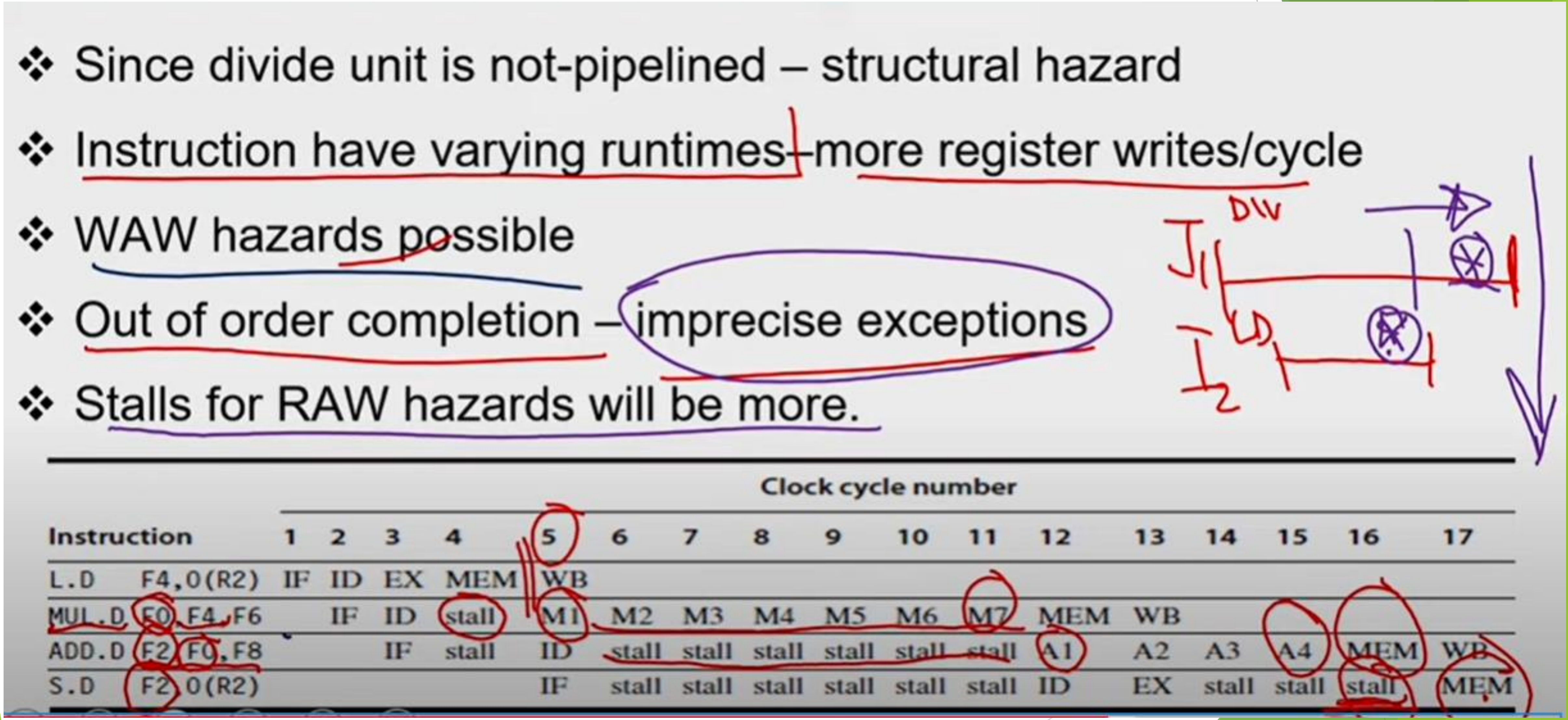
❖ Instruction have varying runtimes – more register writes/cycle

Issues in Longer Latency Pipelines (4)



- Since divide unit is not-pipelined – structural hazard
- Instruction have varying runtimes – more register writes/cycle
- WAW hazards possible

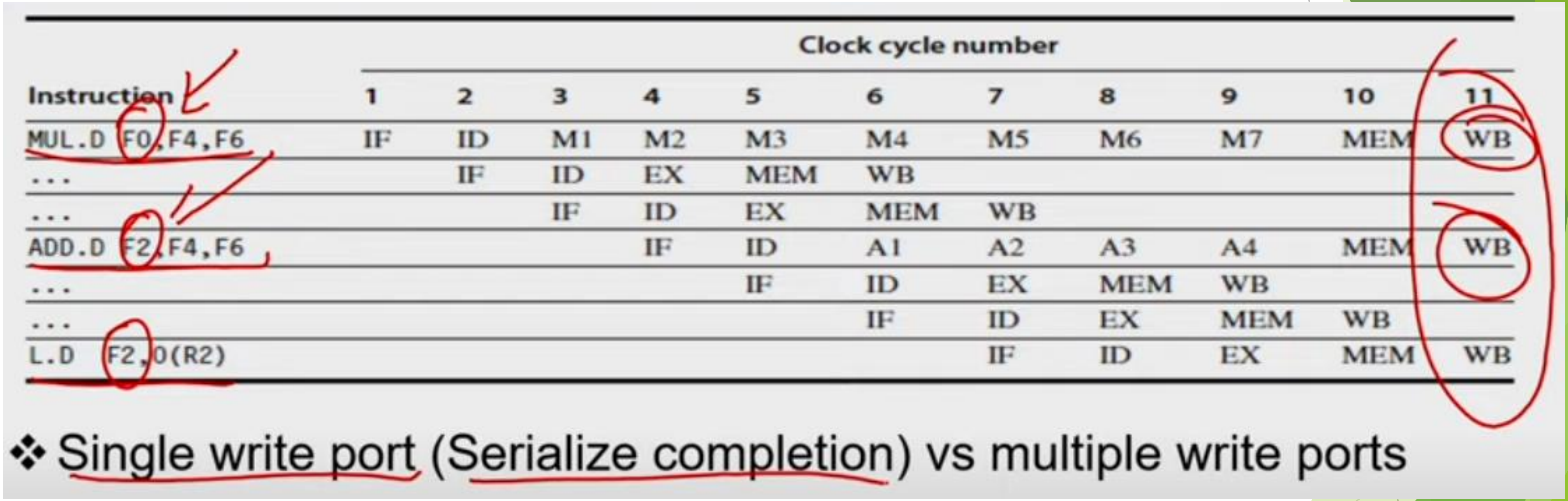
Issues in Longer Latency Pipelines (5)



- Since divide unit is not-pipelined – structural hazard
- Instruction have varying runtimes – more register writes/cycle
- WAW hazards possible
- Out of order completion – imprecise exceptions
- Stalls for RAW hazards will be more.

	Clock cycle number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4,0(R2)	IF	ID	EX	MEM	WB												
MUL.D F0,F4,F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F2,F0,F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	WB
S.D F2,0(R2)					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

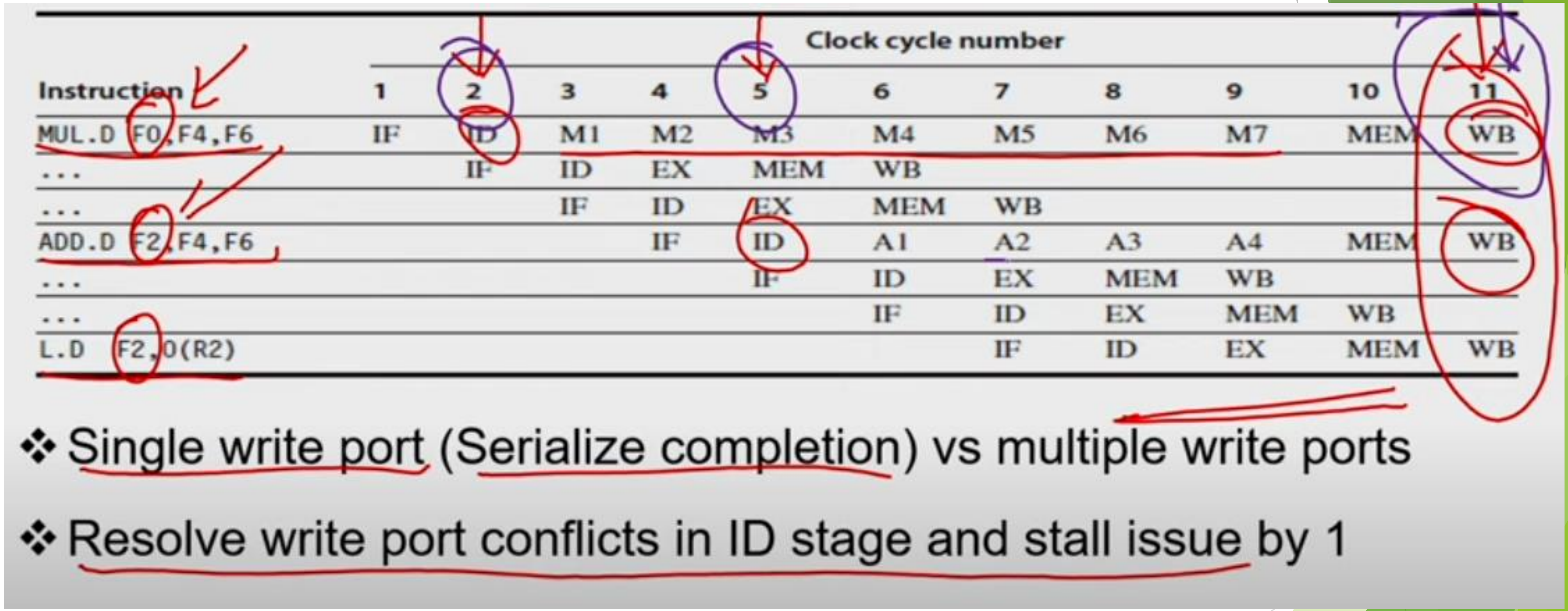
Issues in Longer Latency Pipelines (6)



Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
`MUL.D F0,F4,F6`	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
`...`		IF	ID	EX	MEM	WB					
`...`			IF	ID	EX	MEM	WB				
`ADD.D F2,F4,F6`				IF	ID	A1	A2	A3	A4	MEM	WB
`...`					IF	ID	EX	MEM	WB		
`...`						IF	ID	EX	MEM	WB	
`L.D F2,0(R2)`							IF	ID	EX	MEM	WB

- Single write port (Serialize completion) vs multiple write ports

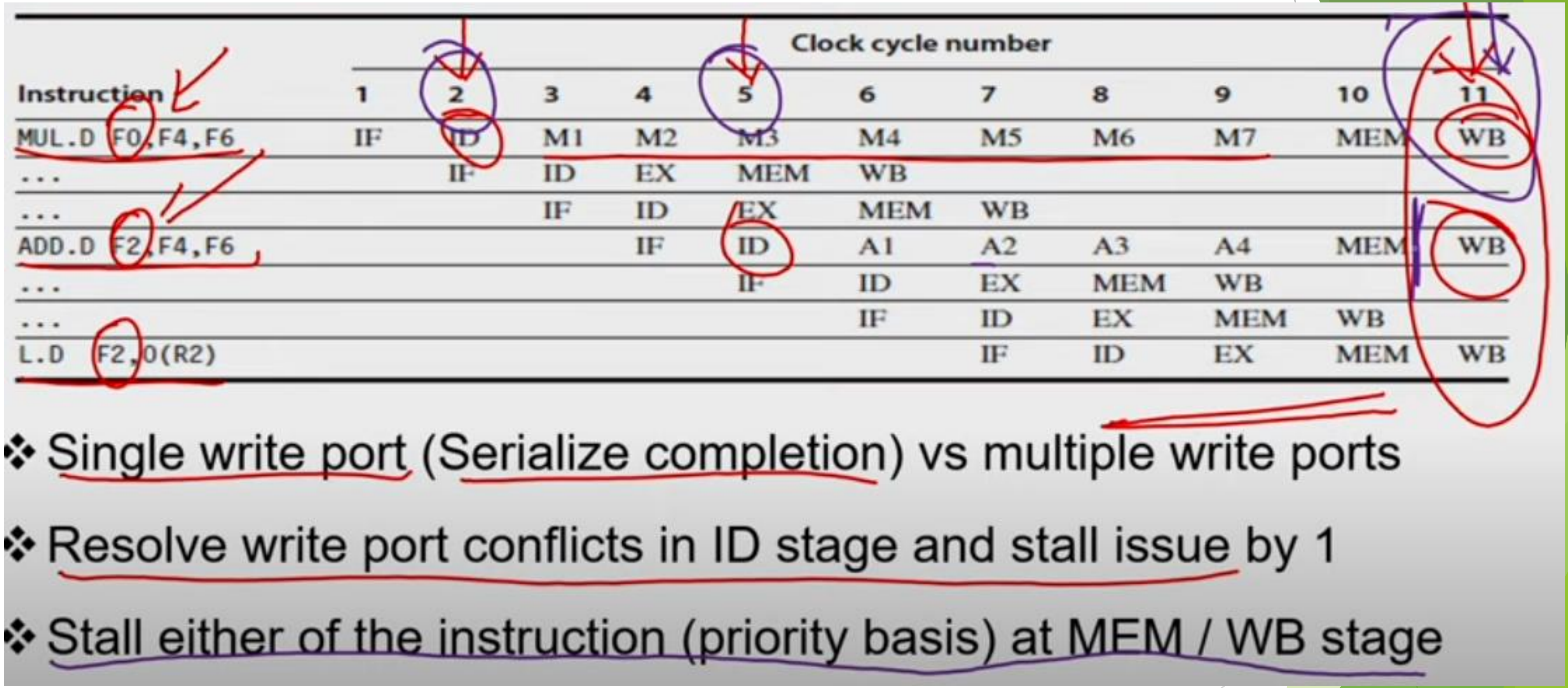
Issues in Longer Latency Pipelines (7)



Instruction	Clock cycle number 1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

- ❖ Single write port (Serialize completion) vs multiple write ports
- ❖ Resolve write port conflicts in ID stage and stall issue by 1

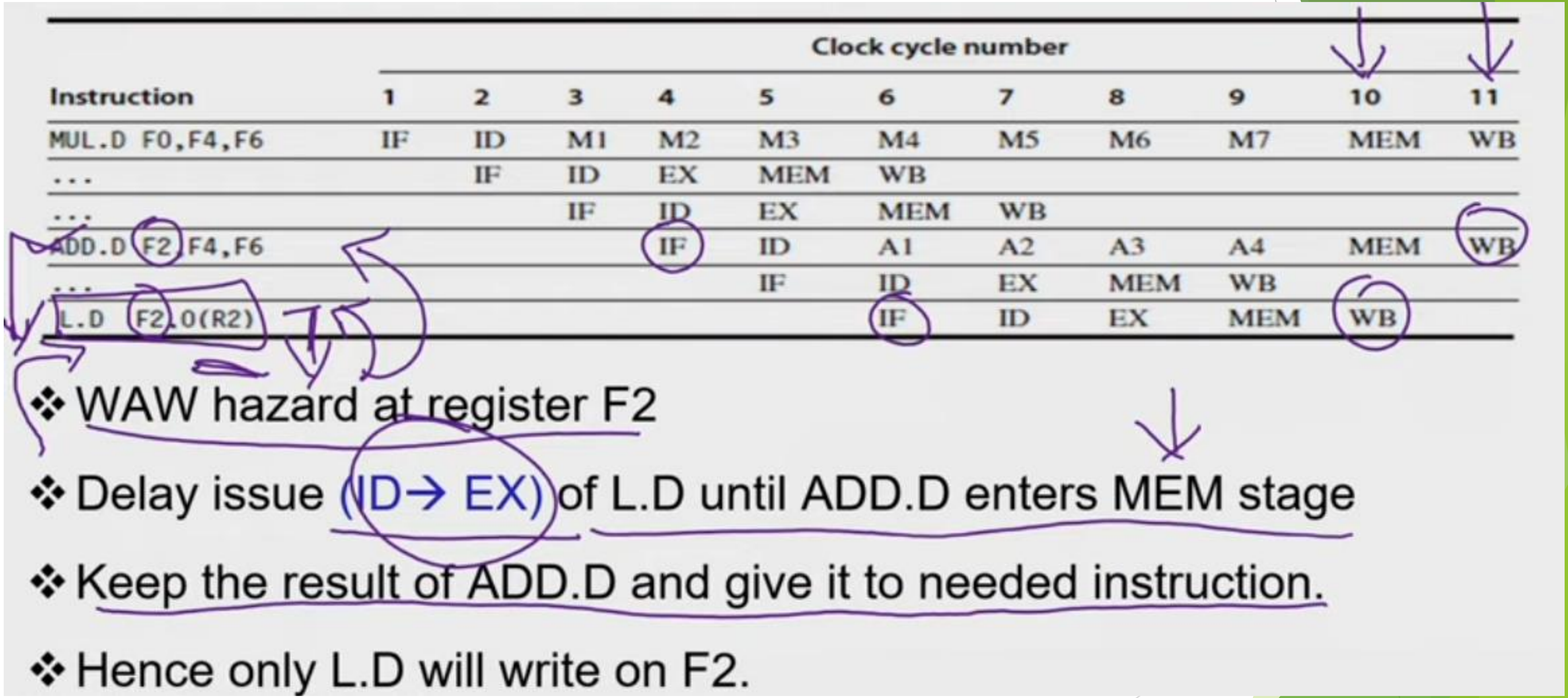
Issues in Longer Latency Pipelines (8)



	Clock cycle number										
Instruction	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

- ❖ Single write port (Serialize completion) vs multiple write ports
- ❖ Resolve write port conflicts in ID stage and stall issue by 1
- ❖ Stall either of the instruction (priority basis) at MEM / WB stage

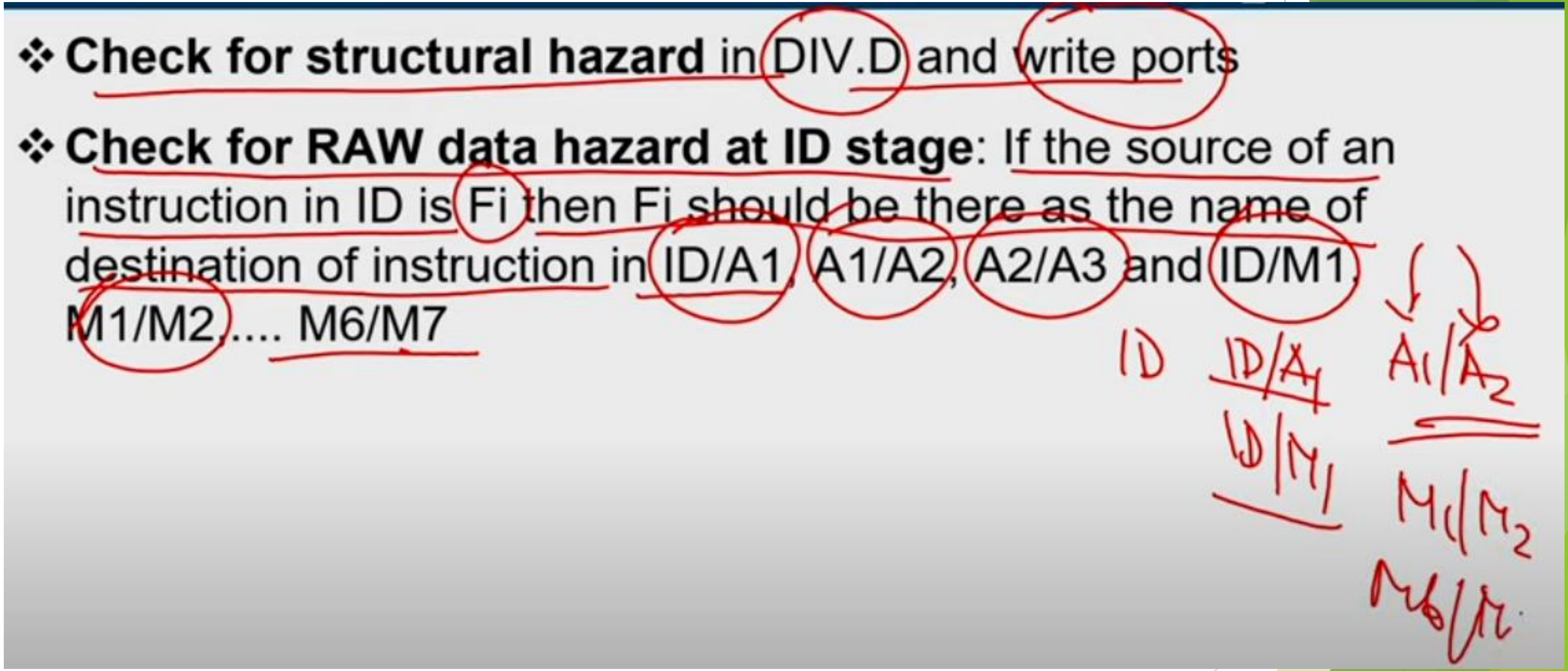
Issues in Longer Latency Pipelines (9)



	Clock cycle number										
Instruction	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
L.D F2,0(R2)						IF	ID	EX	MEM	WB	

- ❖ WAW hazard at register F2
- ❖ Delay issue (ID→ EX) of L.D until ADD.D enters MEM stage
- ❖ Keep the result of ADD.D and give it to needed instruction.
- ❖ Hence only L.D will write on F2.

How to Handle Issues in Longer Latency Pipelines (1)



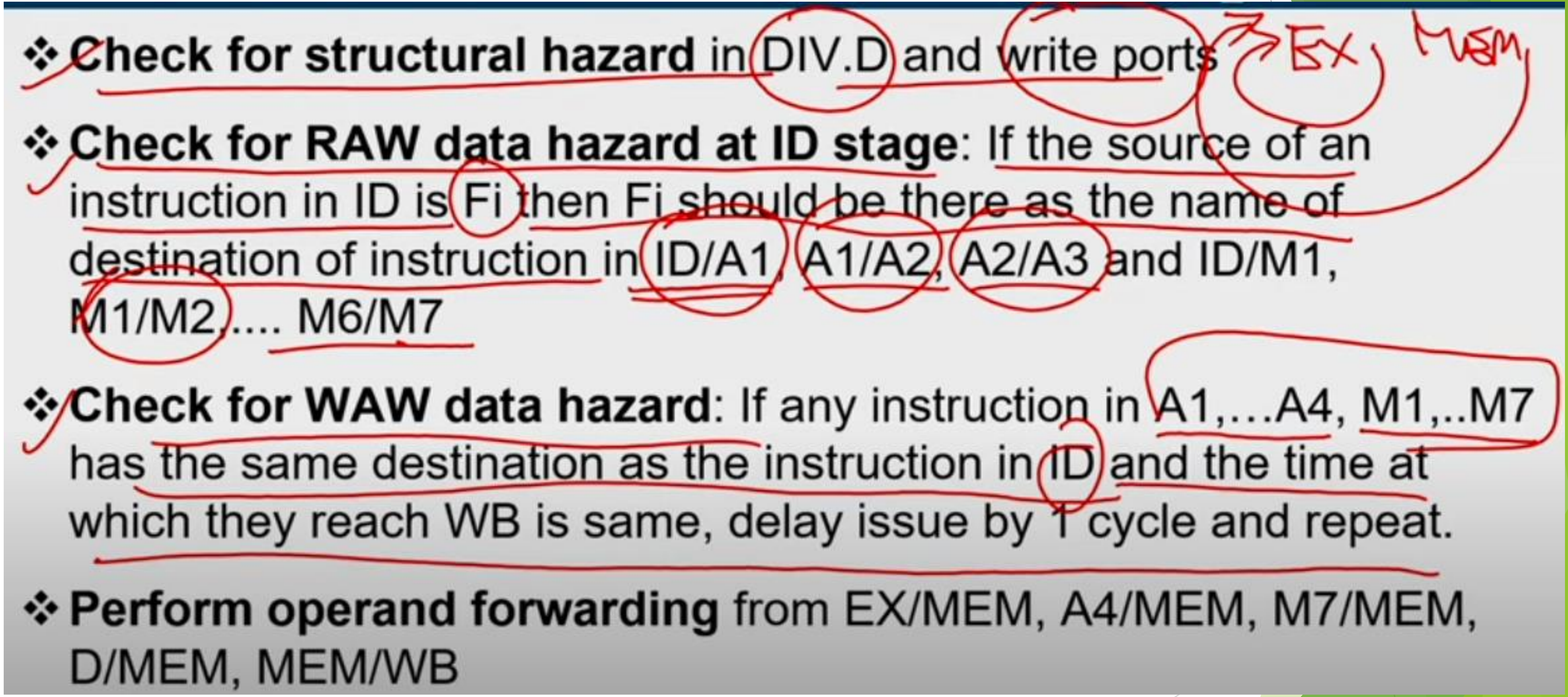
- **Check for structural hazard** in DIV.D and write ports
- **Check for RAW data hazard at ID stage**: If the source of an instruction in ID is Fi then Fi should be there as the name of destination of instruction in ID/A1, A1/A2, A2/A3 and ID/M1, M1/M2..... M6/M7

ID ID/A1 A1/A2

ID/M1 M1/M2

M6/M7

How to Handle Issues in Longer Latency Pipelines (2)

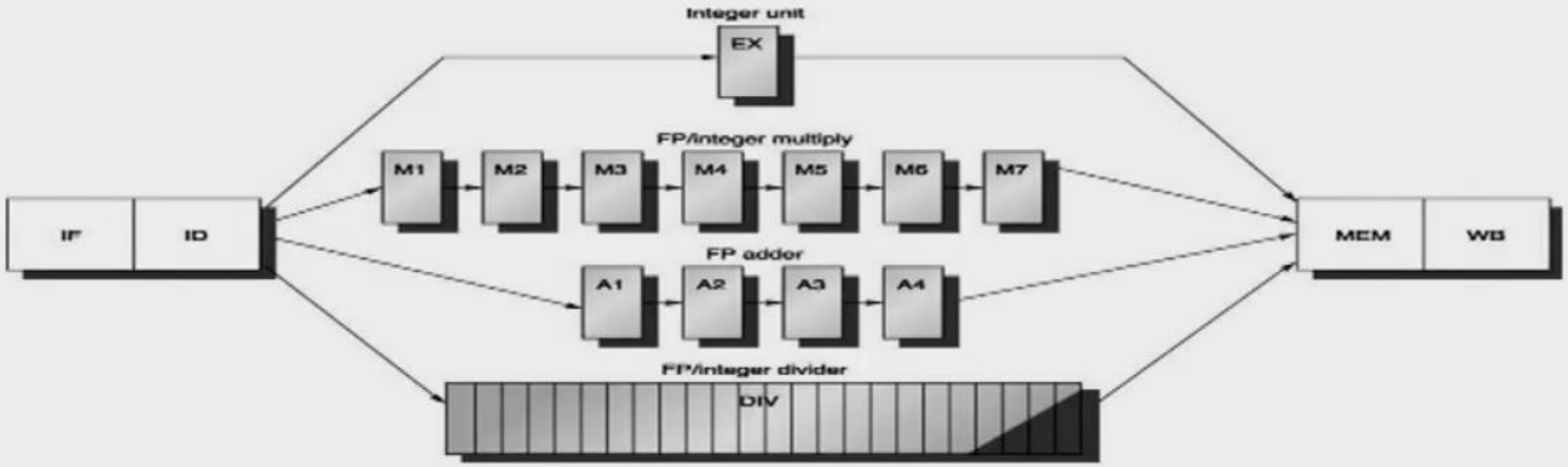


- **Check for structural hazard** in DIV.D and write ports → EX, MEM
- **Check for RAW data hazard at ID stage**: If the source of an instruction in ID is Fi then Fi should be there as the name of destination of instruction in ID/A1, A1/A2, A2/A3 and ID/M1, M1/M2,.... M6/M7
- **Check for WAW data hazard**: If any instruction in A1,…A4, M1,..M7 has the same destination as the instruction in ID and the time at which they reach WB is same, delay issue by 1 cycle and repeat.
- **Perform operand forwarding** from EX/MEM, A4/MEM, M7/MEM, D/MEM, MEM/WB

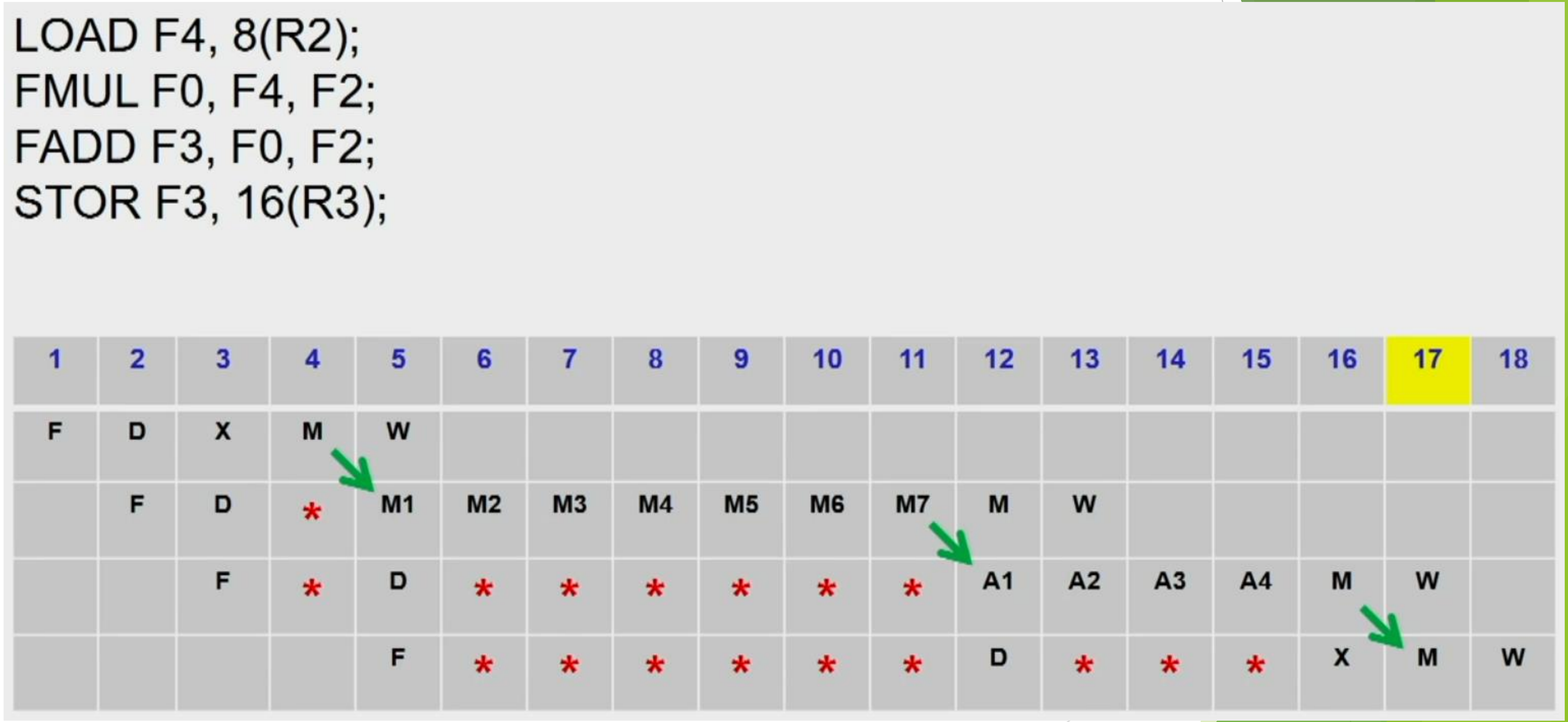
Problems Related to Multi-Cycle Pipeline

Consider the following instruction sequence executed on a MIPS floating point pipeline. Operand forwarding is implemented. [R indicates integer registers and F indicates floating point registers]. Find the clock cycle in which STOR instruction reaches MEM stage. If 8(R2) contains value 'X' and F2 contains value 'A', then what is stored in 16(R3)

LOAD F4, 8(R2);
FMUL F0, F4, F2;
FADD F3, F0, F2;
STOR F3, 16(R3);



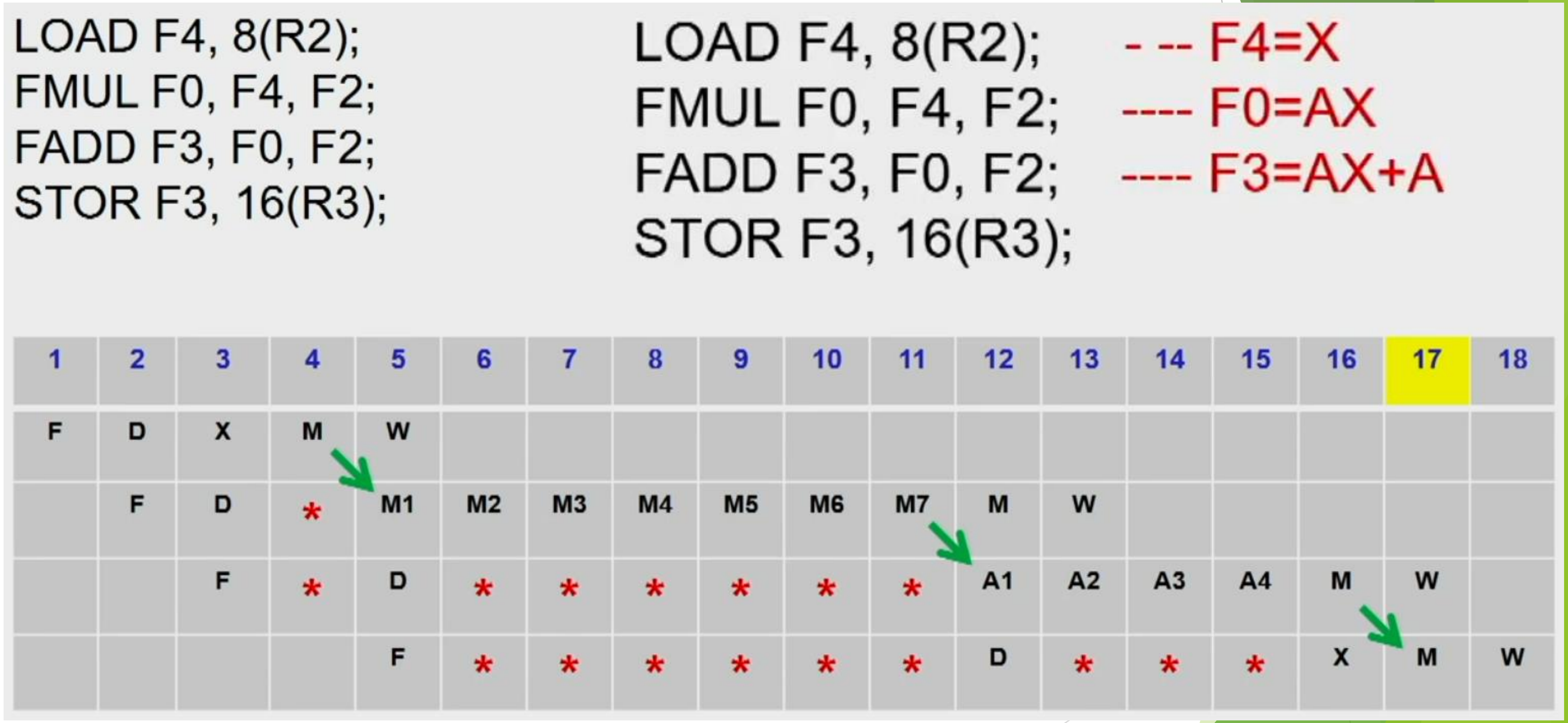
Problems Related to Multi-Cycle Pipeline



```
LOAD F4, 8(R2);
FMUL F0, F4, F2;
FADD F3, F0, F2;
STOR F3, 16(R3);
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
F	D	X	M	W													
	F	D	*	M1	M2	M3	M4	M5	M6	M7	M	W					
		F	*	D	*	*	*	*	*	*	A1	A2	A3	A4	M	W	
				F	*	*	*	*	*	*	D	*	*	*	X	M	W

Problems Related to Multi-Cycle Pipeline



```
LOAD F4, 8(R2);
FMUL F0, F4, F2;
FADD F3, F0, F2;
STOR F3, 16(R3);
```

```
LOAD F4, 8(R2);      - -- F4=X
FMUL F0, F4, F2;     ---- F0=AX
FADD F3, F0, F2;     ---- F3=AX+A
STOR F3, 16(R3);
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
F	D	X	M	W													
	F	D	*	M1	M2	M3	M4	M5	M6	M7	M	W					
		F	*	D	*	*	*	*	*	*	A1	A2	A3	A4	M	W	
				F	*	*	*	*	*	*	D	*	*	*	X	M	W