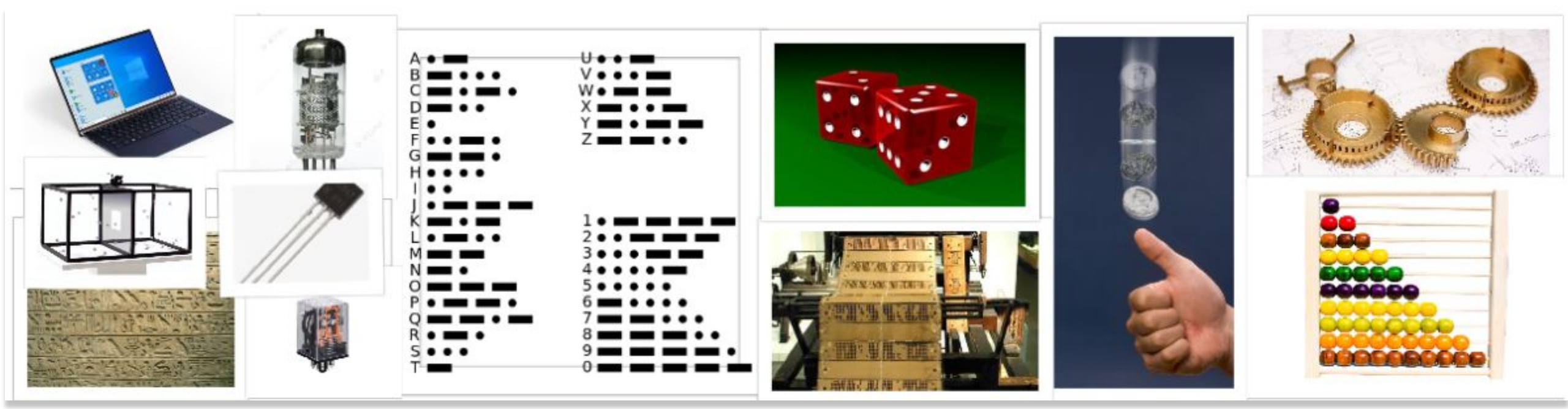


Computer Architecture

Design and Analysis of Instructions

Minimization of Structural & Data Hazards in Pipeline MIPS (RISC) Processor

Pipelined-based Processor

- Pipelining technique exploits parallelism
  - How?
- Can it faces any difficulties while doing so?

Hazards in Pipelined-based Processor

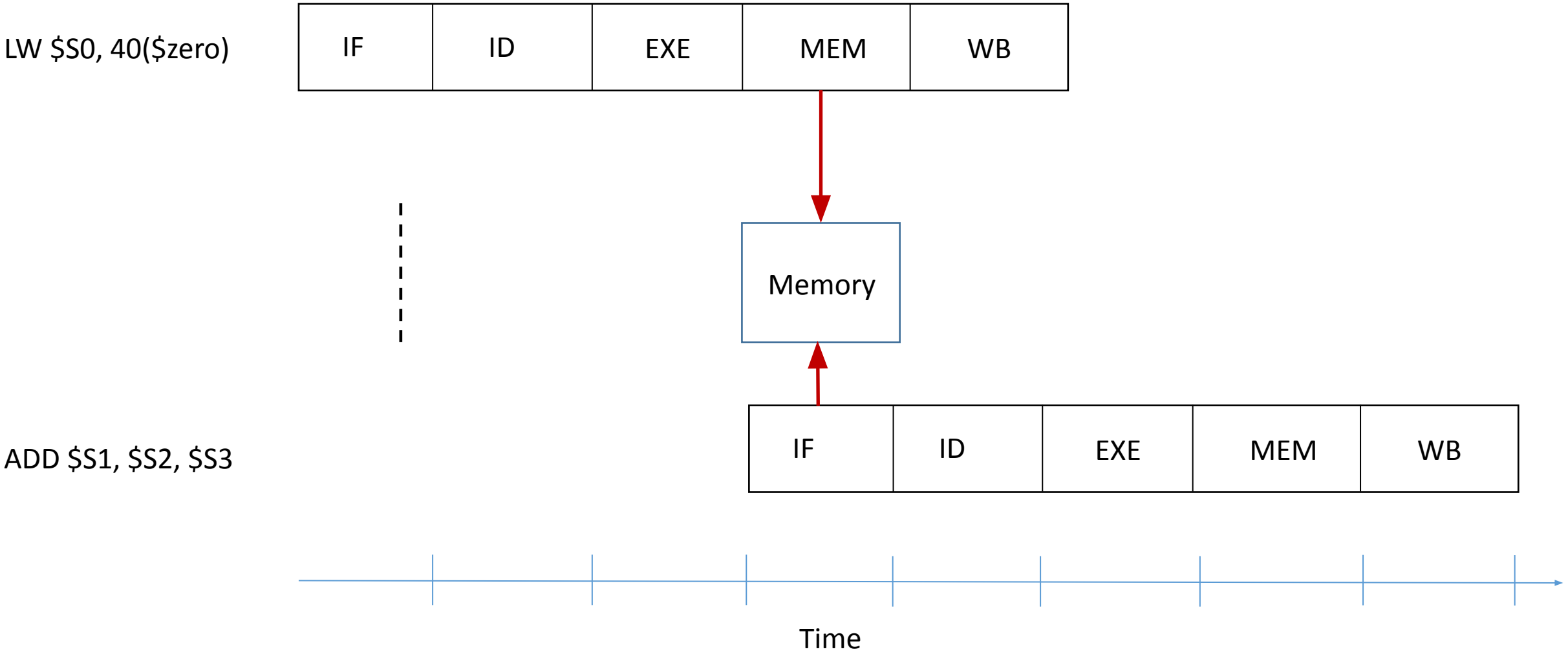
- Conditions that make pipeline to stall: Hazard
  - Structural Hazard
  - Data Hazard
  - Control Hazard

Structural Hazards

- Architectural component does not support parallelism, if we assume
  - Single memory unit
  - More than one instructions trying to write data in register file at the same time

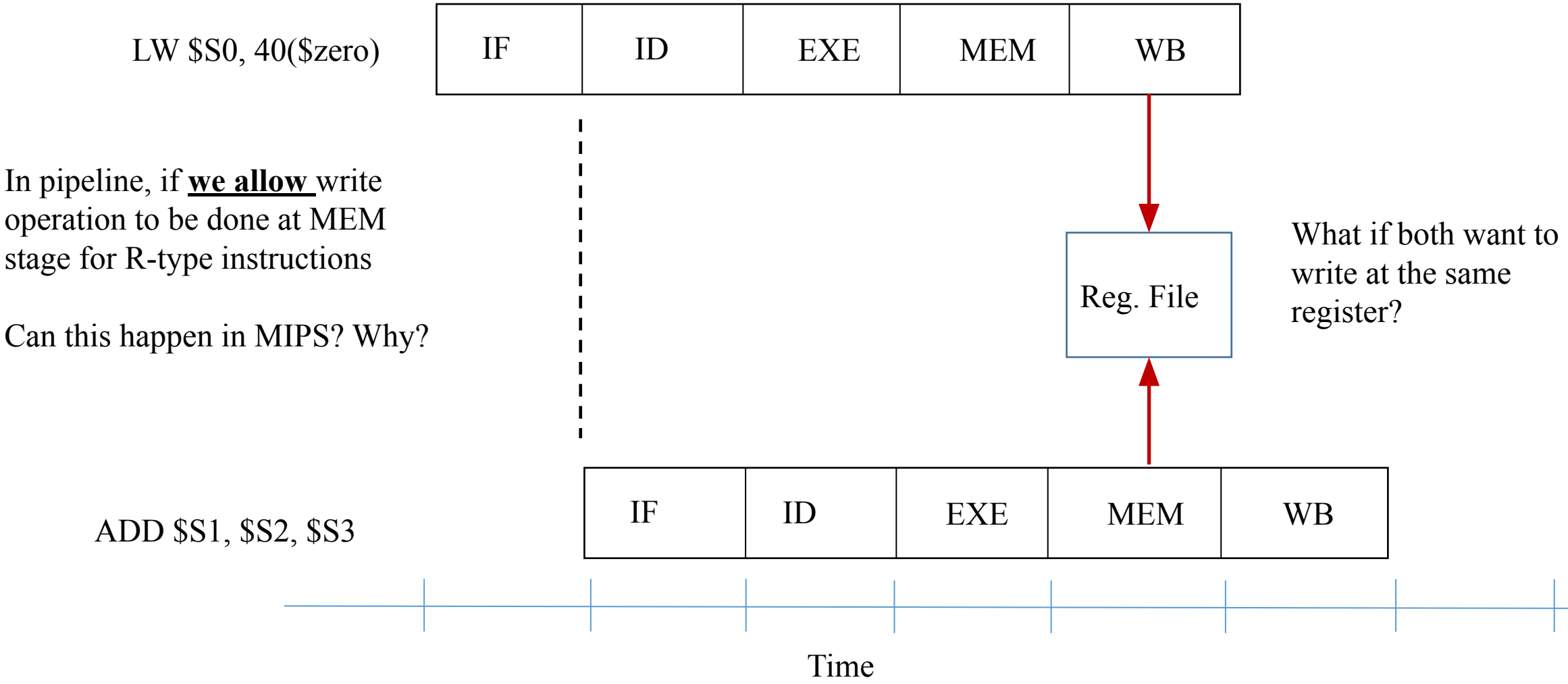
Structural Hazards

- Single memory unit



Structural Hazards

- Two instructions trying to write data in register file at the same time



Solution for Structural Hazard

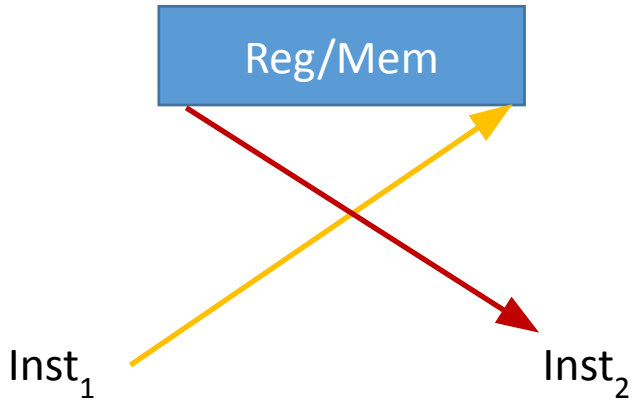
- Incorporate more resources
- Stall the operation
  - Arbitration with interlocking

Dependency between instructions/data

- What are the possible dependencies between two instructions: $Inst_1$ & $Inst_2$?

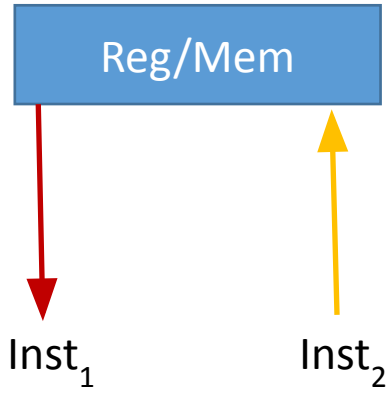
Dependency between instructions/data

- Assume: $Inst_1$ fetched prior to $Inst_2$
  - $Inst_2$ is <u>data dependent</u> on $Inst_1$
    - if $Inst_1$ writes its output in a register, Reg (or memory location)
    - $Inst_2$ reads as that as its input



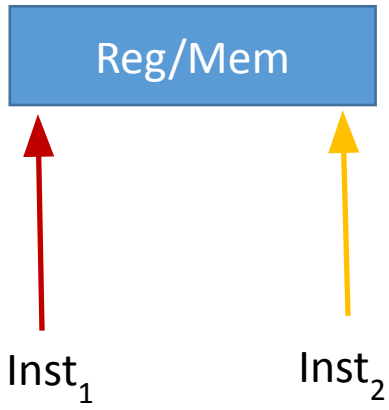
Dependency between instructions/data

- Assume: $Inst_1$ fetched prior to $Inst_2$
  - $Inst_2$ is anti-dependent on $Inst_1$
    - if $Inst_1$ reads data from a register *Reg* (or memory location) which is subsequently overwritten by $Inst_2$



Dependency between instructions/data

- Assume: $Inst_1$ fetched prior to $Inst_2$
  - $Inst_2$ is output dependent on $Inst_1$
    - if both write in the same register *Reg* (or memory location)
    - $Inst_2$ writes its output after $Inst_1$



Dependency between instructions/data

- Assume: $Inst_1$ fetched prior to $Inst_2$
  - $Inst_2$ is control dependent on $Inst_1$
    - if $Inst_1$ must completes before a decision can be made whether or not to execute $Inst_2$

Data Hazards

- Data dependences between instructions
  - True or real
  - False or name
- Inst1 & Inst2 are **so close** that their overlapping would change their access order to register, Reg.

Data Hazards

- Types of data hazards
  - Read after write (RAW)
    - caused by data dependency
  - Write after read (WAR)
    - caused by anti-dependence
  - Write after write (WAW)
    - caused by output dependence

Data Hazards

- WAW hazards occur
  - Write operation in more than one stages
  - Allow an instructions to proceed even when a previous instruction is stalled

- Will it occur in MIPS?

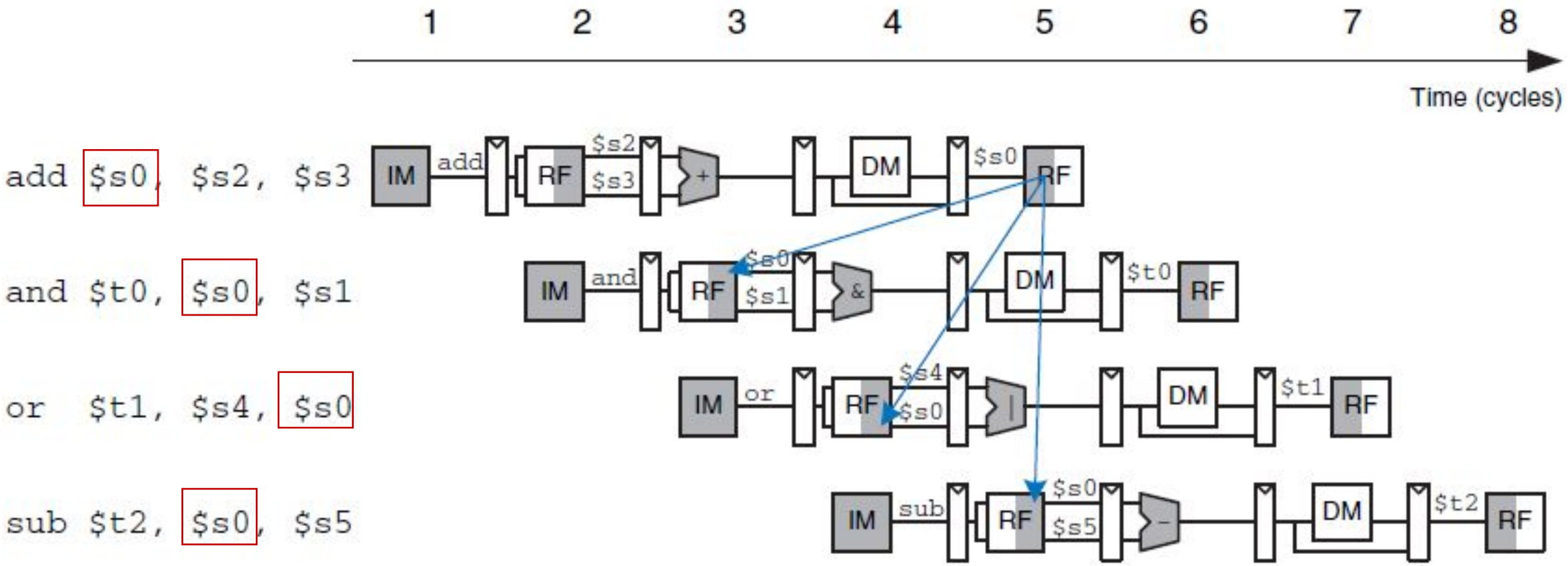
Data Hazards

- WAR hazards occur
  - Write stage precedes a read stage

- Will it occur in MIPS?

Data Hazards

- Only RAW hazard occurs in MIPS



How many clock-cycle do we need to wait to get the proper value?

Data Hazards

- How does one solve RAW hazards?
  - Software-based solutions
  - Hardware-based solutions

Data Hazards

- Software-based solution
  - No-op
  - How many clock-cycle or No-op do we need to wait to get the proper value?
  - Can we reduce the #of no-ops?

Data Hazards

- Compiler has to control
  - noop-instructions
  - Rearrange the program code (instruction scheduling or pipeline scheduling)

Data Hazards

- Instruction scheduling or pipeline scheduling

Original code	Insert the NOOP instruction	After ordering the code
LW R1, 40 ($40)	LW R1, 40 ($0)	LW R1, 40 ($0)
ADDI R1, R1, 5	NOOP	LW R2, 44 ($0)
SW R1, 50 ($0)	NOOP	NOOP
LW R2, 44 ($0)	ADDI R1, R1, 5	ADDI R1, R1, 5
ADD R1, R2, R3	SW R1, 50 ($0)	SW R1, 50 ($0)
SW R2, 54 ($0)	LW R2, 44 ($0)	ADD R1, R2, R3
	NOOP	SW R2, 54($0)
	NOOP	
	ADD R1, R2, R3	
	SW R2, 54 ($0)	

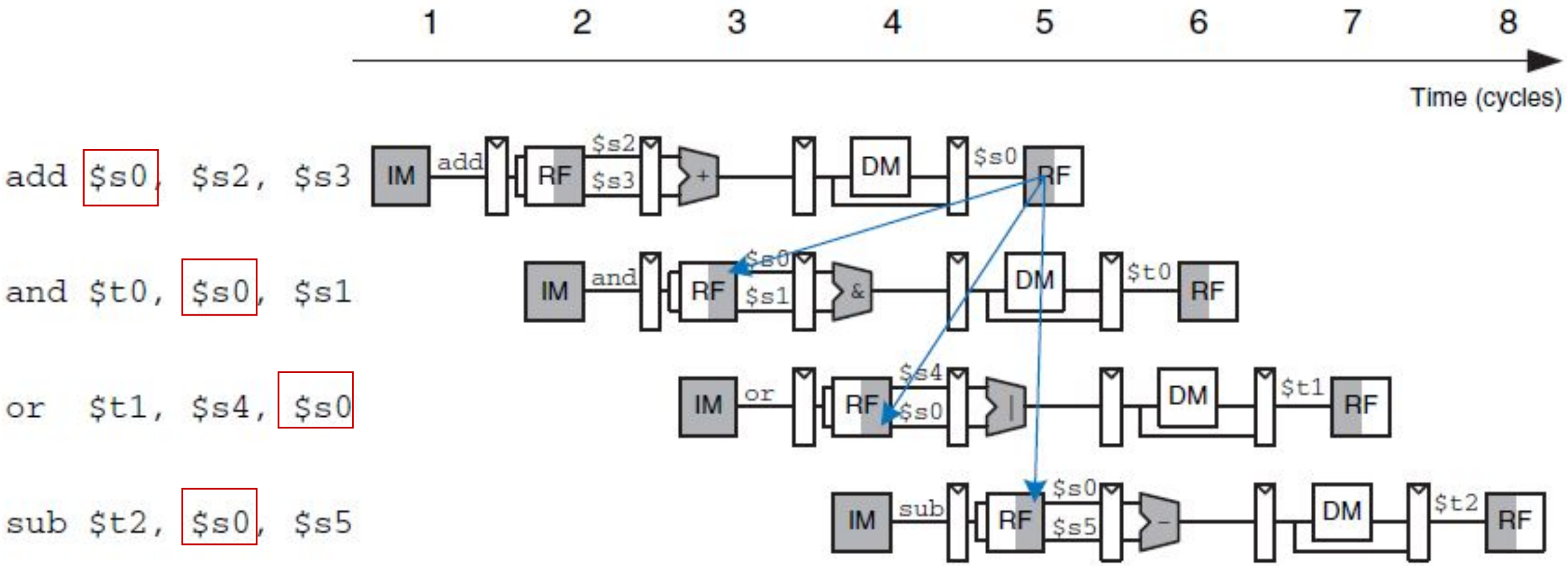
How does one decide no. of consecutive NOOP instruction to put after an instruction? It depends on delay (clock cycle) to produce the correct operand for the dependent instruction(s).

Data Hazards

- Is the NOOP instruction available in MIPS?
  - No
- How can one make such instruction?

Data Hazards

- Only RAW hazard occurs in MIPS



How many clock-cycle do we need to wait to get the proper value?

Data Hazards: H/W-based Solution

- Interlocking -- a simple solution
  - Detect the hazard
  - Stall the pipeline
  - Degrade the speedup

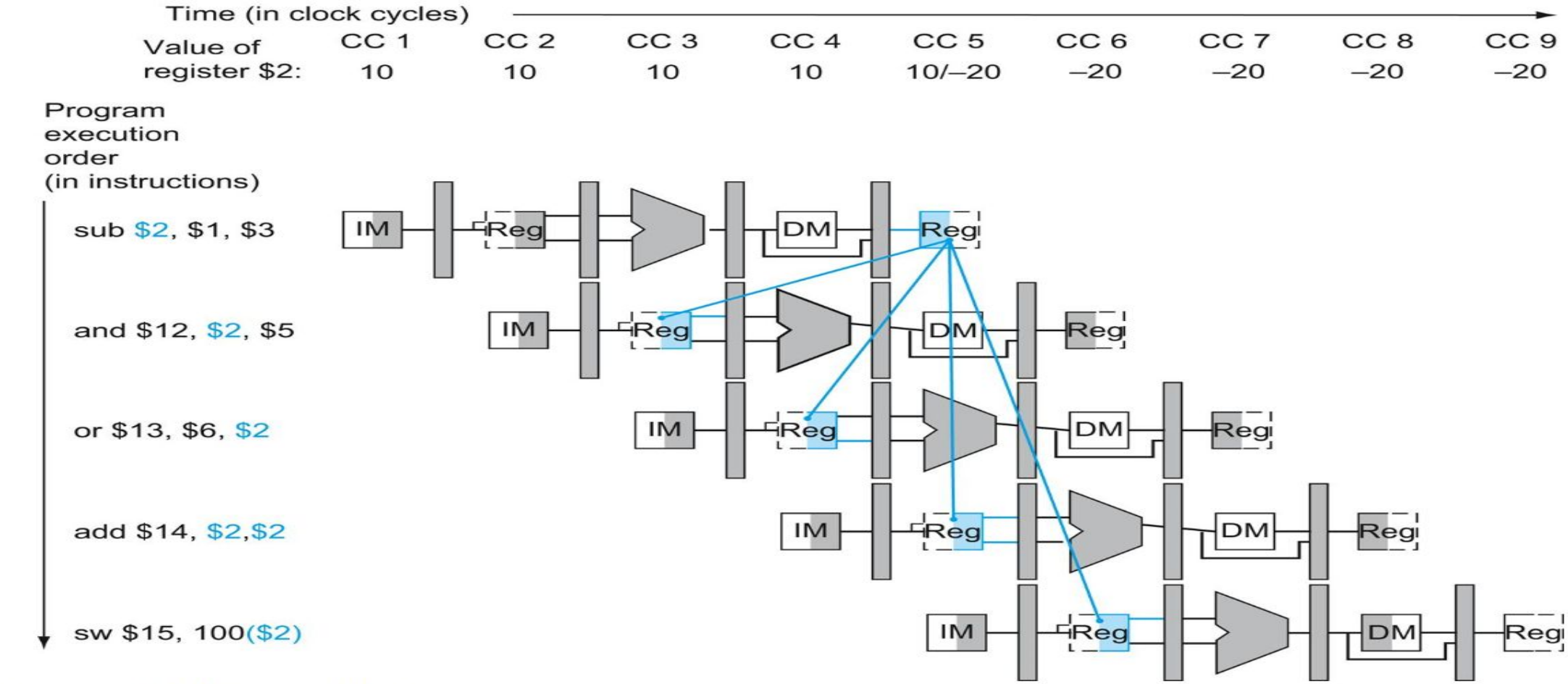
Data Hazards: H/W-based Solution

- Is there other solutions?
- Forwarding – a sophisticated solution
  - The result of the ALU output of $Inst_1$ in the EX stage can immediately forward back to ALU input of EX stage as an operand for $Inst_2$

An example

```
sub    $2, $1,$3        # Register $2 written by sub
and    $12,$2,$5        # 1st operand($2) depends on sub
or     $13,$6,$2        # 2nd operand($2) depends on sub
add    $14,$2,$2        # 1st($2) & 2nd($2) depend on sub
sw     $15,100($2)      # Base ($2) depends on sub
```

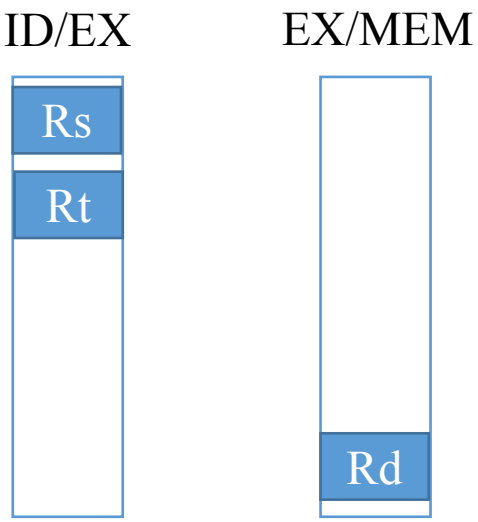
Data Hazard


Time (in clock cycles)
Value of register $2:
CC 1 10
CC 2 10
CC 3 10
CC 4 10
CC 5 10/–20
CC 6 –20
CC 7 –20
CC 8 –20
CC 9 –20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2,$2
sw $15, 100($2)
IM
Reg
DM

Dependency detection

op	rs	rt	rd	shamt	funct
6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)

- 1a: EX/MEM.RegisterRd == ID/EX.RegisterRs
- 1b: EX/MEM.RegisterRd == ID/EX.RegisterRt
- 2a: MEM/WB.RegisterRd == ID/EX.RegisterRs
- 2b: MEM/WB.RegisterRd == ID/EX.RegisterRt



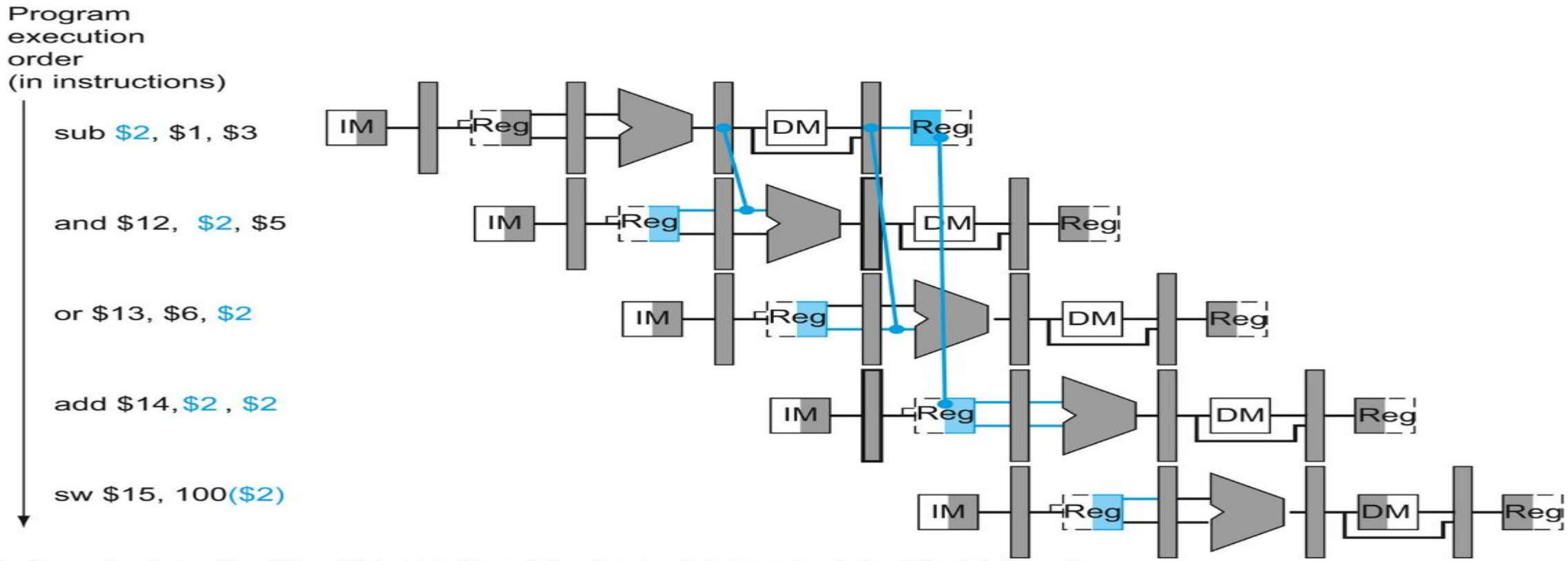
sub-and is a type-1a hazard
sub-or is a type-2b hazard
sub-add is not a hazard
sub-sw is not a hazard

```
sub    $2, $1,$3       # Register $2 written by sub
and    $12,$2,$5       # 1st operand($2) depends on sub
or     $13,$6,$2       # 2nd operand($2) depends on sub
add    $14,$2,$2       # 1st($2) & 2nd($2) depend on sub
sw     $15,100($2)     # Base ($2) depends on sub
```

Data Hazards: H/W-based Solution

- Forwarding or bypassing

	Time (in clock cycles)								
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



Data Hazards: H/W-based Solution

- How does one perform forwarding or bypassing?

  - Put MUXs in front of ALU select

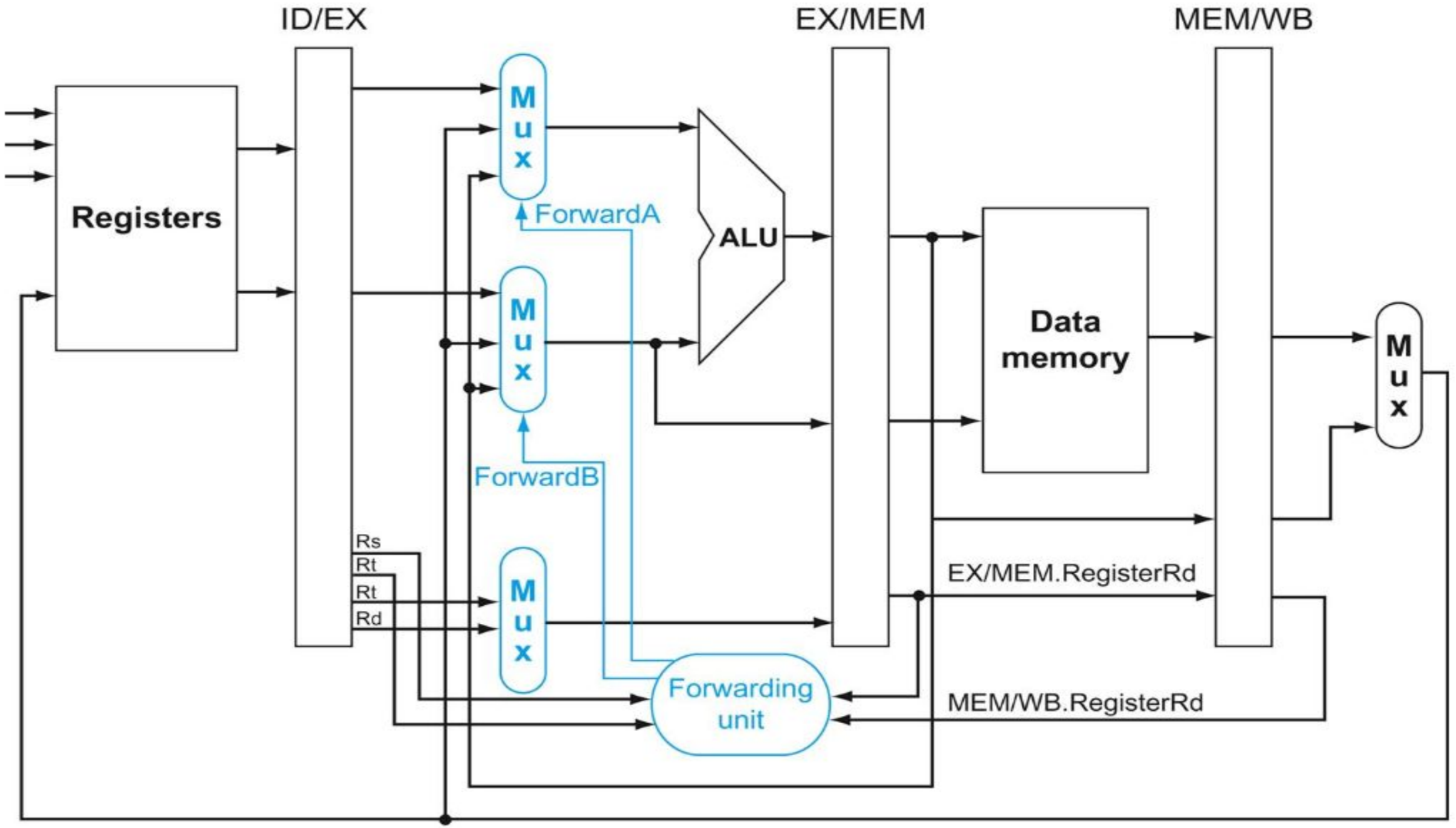
  - ALU's inputs:
    - Register file or Decode stage
    - Memory stage
    - Writeback stage

Data Hazards: H/W-based Solution

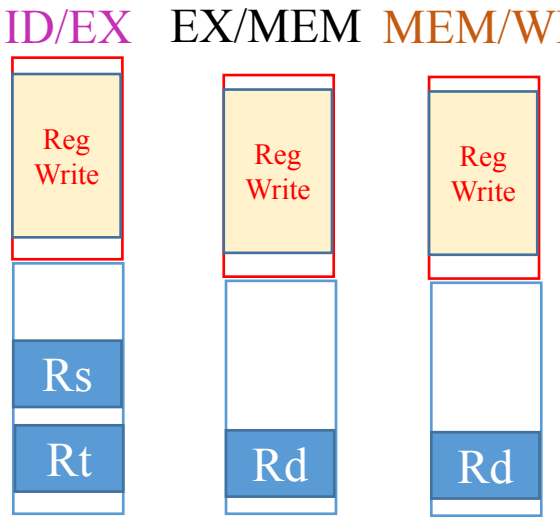
- How does one perform forwarding or bypassing?

  - An instruction in the Execute stage
    - A source register matching the destination register of an instruction in
      - Memory stage and/or
      - Writeback stage

Data Hazards: H/W-based Solution



Data Hazards: H/W-based Solution



- Hazard unit generates control signals for
  - Mux SrcA [ForwardA]
  - Mux SrcB [ForwardB]
- The control logic when updated data in EX-MEM stage

- if (EX/MEM.RegWrite AND (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd == ID/EX.RegisterRs)) then
  ForwardA = 10
- if (EX/MEM.RegWrite AND (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd == ID/EX.RegisterRt)) then
  ForwardB = 10

- The control logic when updated data in MEM-WB stage

- if (MEM/WB.RegWrite AND (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd == ID/EX.RegisterRs)) then
  ForwardA = 01
- if (MEM/WB.RegWrite AND (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd == ID/EX.RegisterRt)) then
  ForwardB = 01

Data Hazards: H/W-based Solution

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Complicated situation

- The result of the instruction in the WB stage, the result of the instruction in the MEM stage, and source operand of the instruction in the ALU stage.

The control logic for (MEM hazard)

```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
…
```

if (MEM/WB.RegWrite AND (MEM/WB.RegisterRd != 0)

AND ! (EX/MEM.RegWrite AND (EX/MEM.RegisterRd != 0) AND (EX/MEM.RegisterRd == ID/EX.RegisterRs)

AND (MEM/WB.RegisterRd == ID/EX.RegisterRs)) then
    ForwardA = 01

if (MEM/WB.RegWrite AND (MEM/WB.RegisterRd != 0)

AND ! (EX/MEM.RegWrite AND (EX/MEM.RegisterRd != 0) AND (EX/MEM.RegisterRd == ID/EX.RegisterRt)

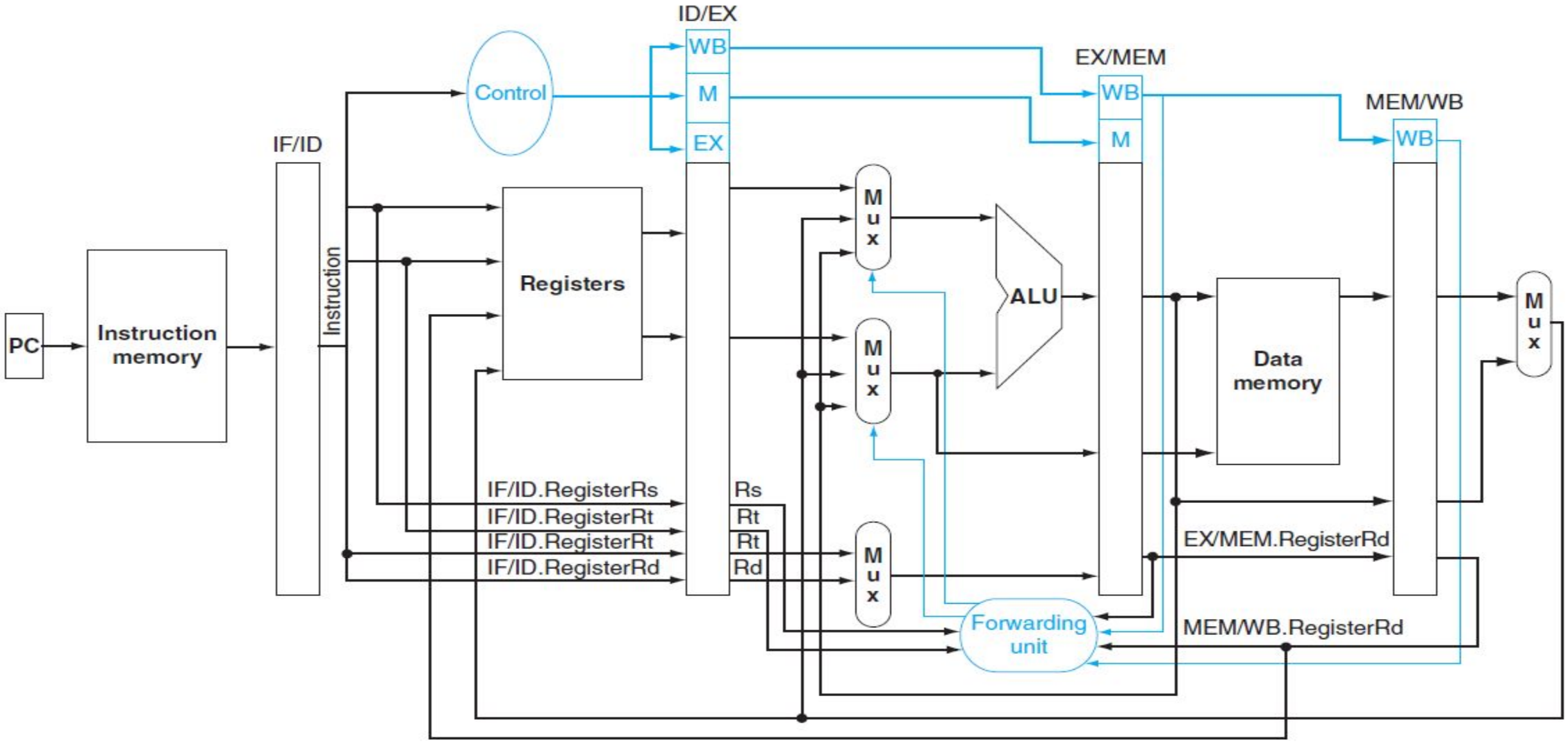
AND (MEM/WB.RegisterRd == ID/EX.RegisterRt)) then
    ForwardB = 01

Updated condition for Slide#33 (Mem-WB)

Hazard in WB stage

- No hazard in WB stage
- Reg. file supplies the correct result if the instruction in the ID stage reads the same written by the instruction in the WB stage

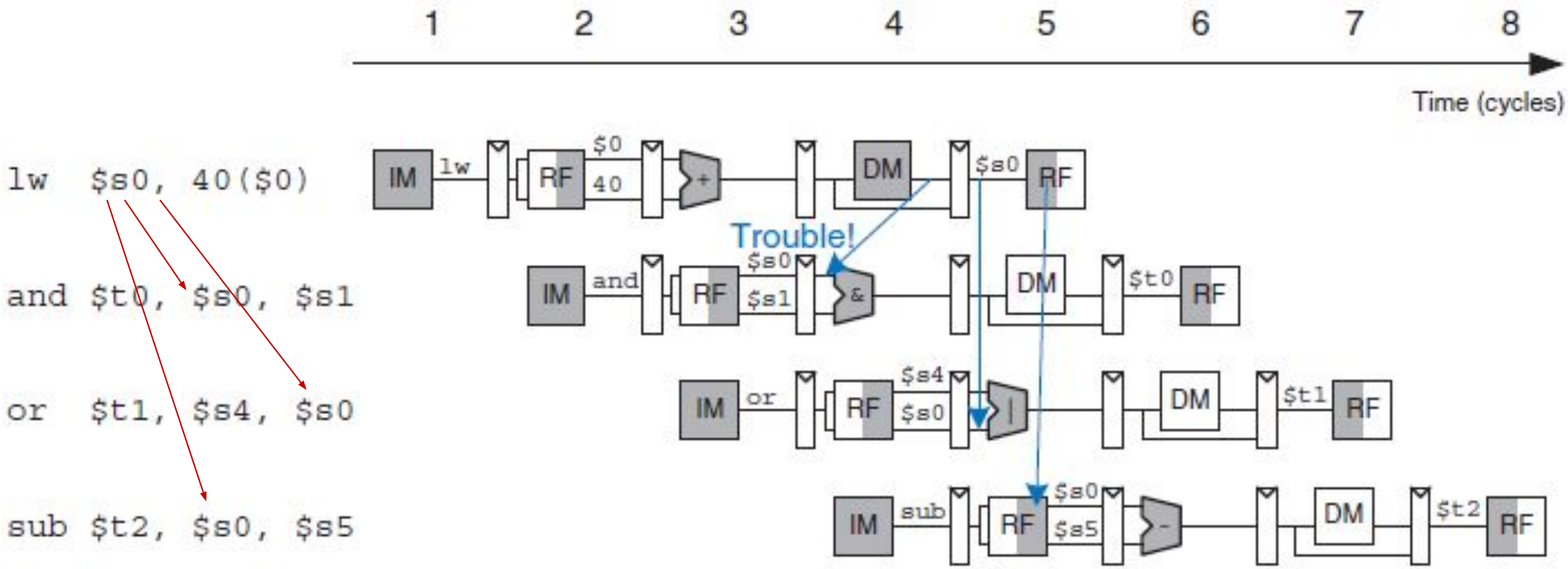
Data Hazards: H/W-based Solution



What if we want to copy data from one memory location to other memory location

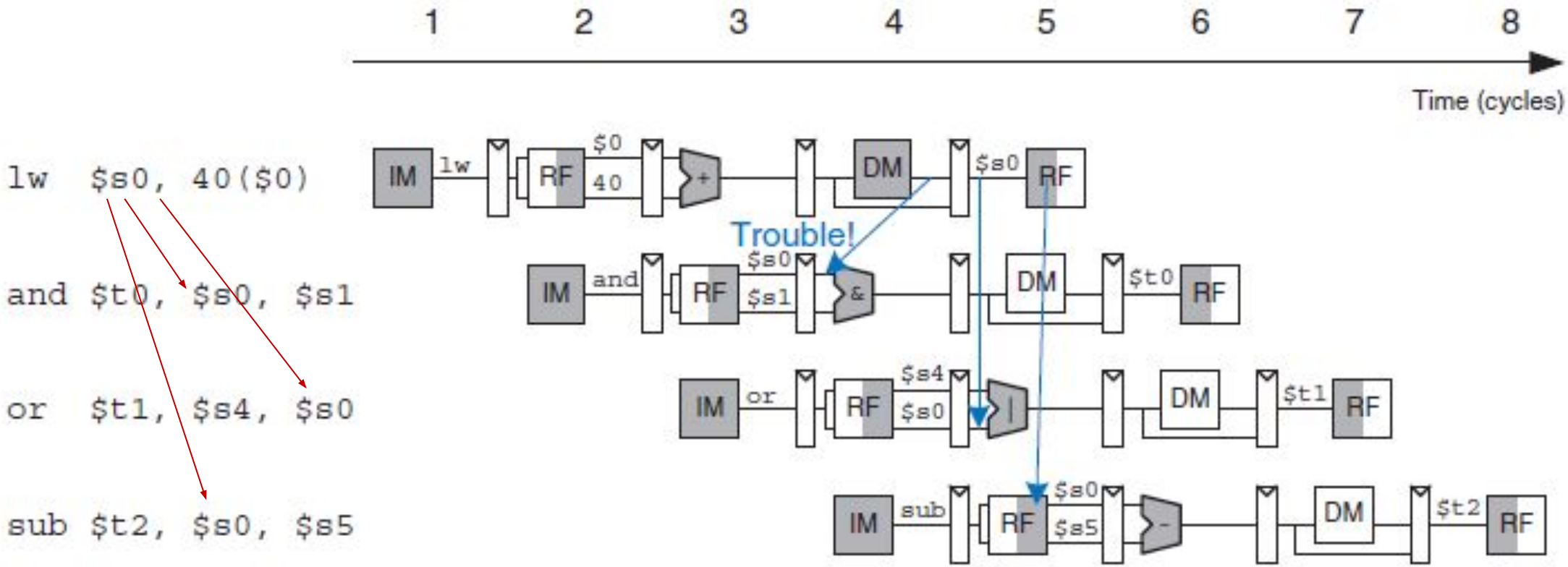
Data Hazards

- Can we solve this one by forwarding?
- Is lw-instruction only causes this?



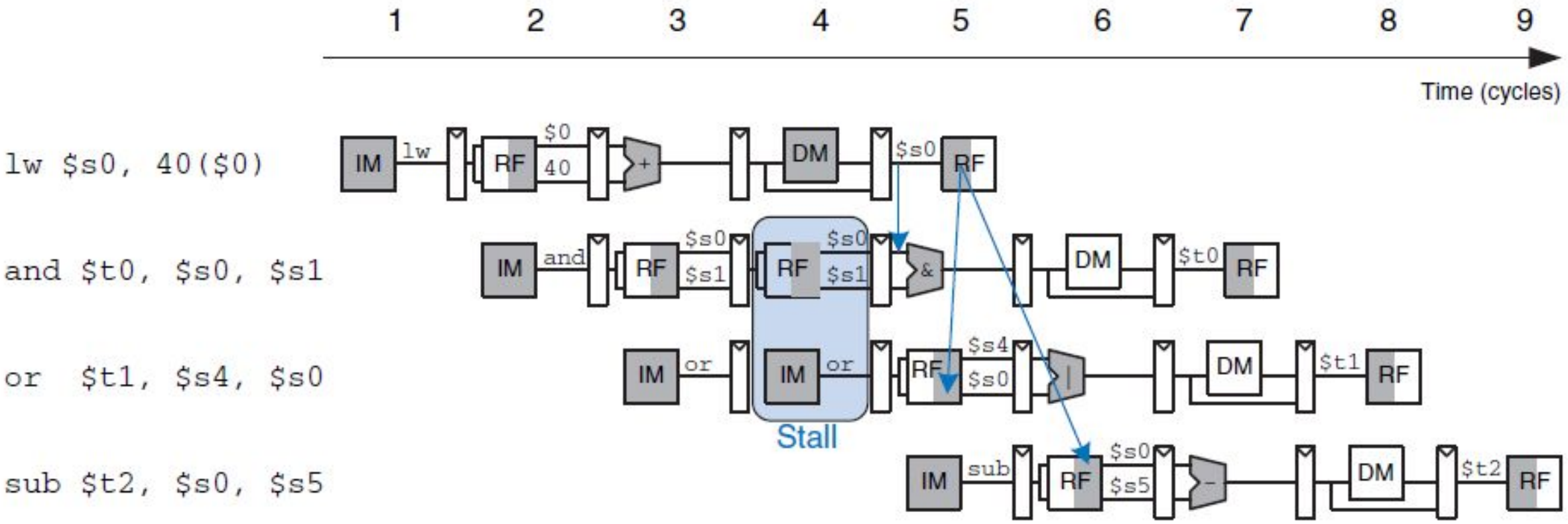
Data Hazards

- Can we solve this one by forwarding? [No]
- How does one solve it?



Data Hazards

- Forwarding with pipeline interlocking (stalling)



Data Hazards

LW	op	rs	rt	offset		
	6-bits(31-26)	5-bits(25-21)	5-bits(20-16)	5-bits(15-11)	5-bits(10-6)	6-bits (5-0)

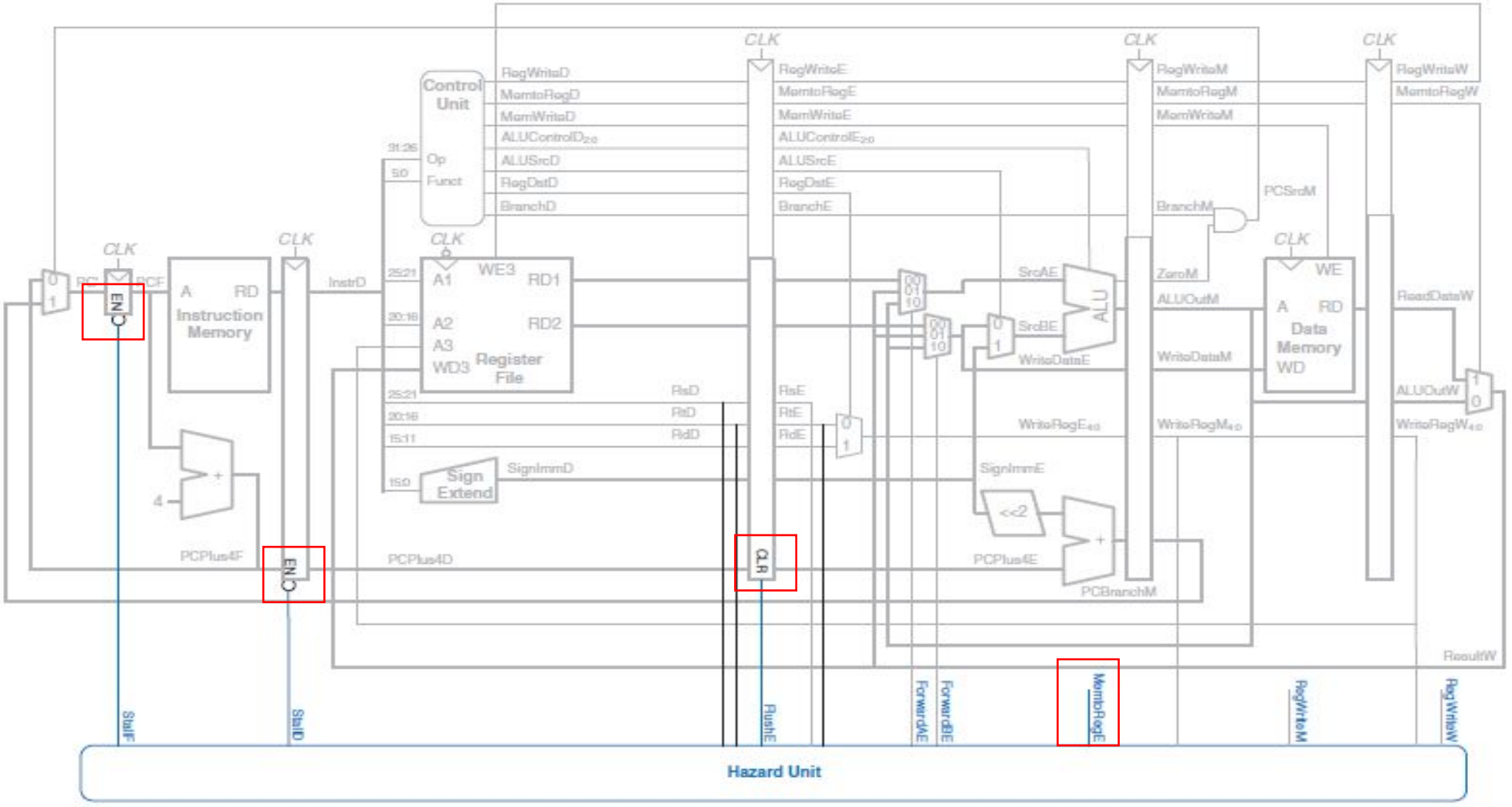
- Forwarding with pipeline interlocking (stalling)
- Hazard unit checks
  - Is it the lw-instruction?
    - Destination register (EXE/MEM.**rt**) matches with
    - Source operand in the Decode stage (ID/EXE.rs or ID/EXE.rt)
    - Stall the Decode stage (how long?)
- How does one perform stall operation?

Data Hazards

- How does one perform stall operation?
  - Incorporate the Enable (EN) control signal
    - Fetch pipeline register
    - Decode pipeline register
  - Incorporate a synchronous reset/clear (CLR)
    - Execute pipeline register

Data Hazards

- Forwarding with pipeline interlocking (stalling)



Data Hazards

- Control logic for forwarding with pipeline interlocking (stalling)
  - If lw-instruction
    - Asserted MemtoRegE
  - lwstall = ((rsD == rtE) OR (rtD == rtE)) AND MemtoRegE
  - StallF = lwstall
  - StallD = lwstall
  - FlushE = lwstall // or set 0's to all control signals

E: EXE/MEM, D: ID/EXE

Performance of pipelines with stalls

- A stall causes the pipeline performance to degrade from the ideal performance (1 instr. per cycle time)

- $\text{Speedup} = \frac{Avg.\ instr.\ time\ unpipelined}{Avg.\ instr.\ time\ pipelined}$

$$= \frac{CPI\ unpipelined \times Clock\ cycle\ unpipelined\ (T)}{CPI\ pipelined \times Clock\ cycle\ pipelined\ (\frac{T}{K})}$$

K = depth of pipeline

- $\text{CPI pipelined} = Ideal\ CPI + Pipeline\ stall\ clock\ cycle\ per\ instr.$

$$= 1 + pipeline\ stall\ clock\ cycle\ per\ instruction$$

- $\text{Speedup} = \frac{CPI\ unpipelined}{1 + pipeline\ stall\ cycle\ per\ instruction}$

$$= \frac{Pipeline\ depth\ (K)}{1 + pipeline\ stall\ cycle\ per\ instruction}$$

If we ignore the cycle time overhead of pipelining and assume that the stages are perfectly balanced, then the cycle time of the two processors can be equal

Homework

- Show the modified datapath and control in the pipeline for resolving memory-to-memory operation (lw then sw)
- How many ways one stalls the pipeline?
- Write a Verilog code for 5-stage pipelined MIPS with data forwarding technique
- Write a C++ program which can read MIPS assembly program and find out
  - How many instructions are valid?
  - How many registers are used?
  - RAR, RAW, WAR & WAW hazards for a given window size/number of stage *n*

Homework

- Write a C++ program which will take any assembly program written in 32 bit MIPS ISA and find out the 4 hazards (RAR, RAW, WAW, WAR) and control hazard in 5-stages scalar pipelined MIPS processor without hazard detection unit. To resolve the 4 hazards, it will rearrange the instructions. In the worst case, to resolve the hazards, it will insert the NOP instructions. Similarly, to resolve the control hazards, it will use delayed branch techniques: from before, from target and from fall through. In the worst case, to resolve control hazards, it will insert the NOP instructions. Finally, it will generate hazard free assembly code. Use MARS simulator to verify the output of the newly created file.

Summary

- Types of hazards
- Minimization of structural hazards with
  - Increased resources
  - Interlocking technique
- Types of data hazards
- Minimization of data hazards with
  - Forwarding technique
  - Forwarding with interlocking technique
  - Performance analysis
  - Compiler-based technique