Computer Architecture (CS F342)

Design and Analysis of Instructions

Minimization of Structural & Data Hazards in Pipelined 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

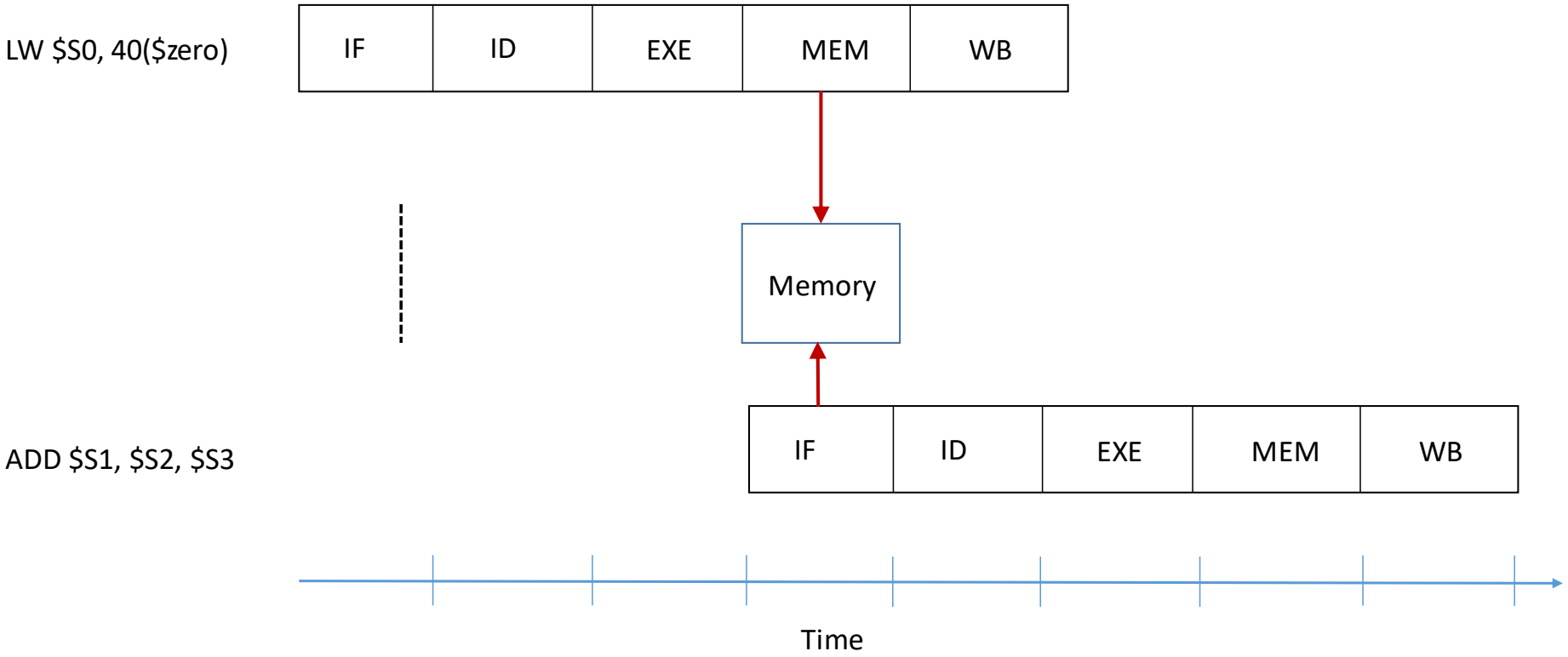
Structural Hazards

- 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

We used this in multicycle
Here it wouldn't work.

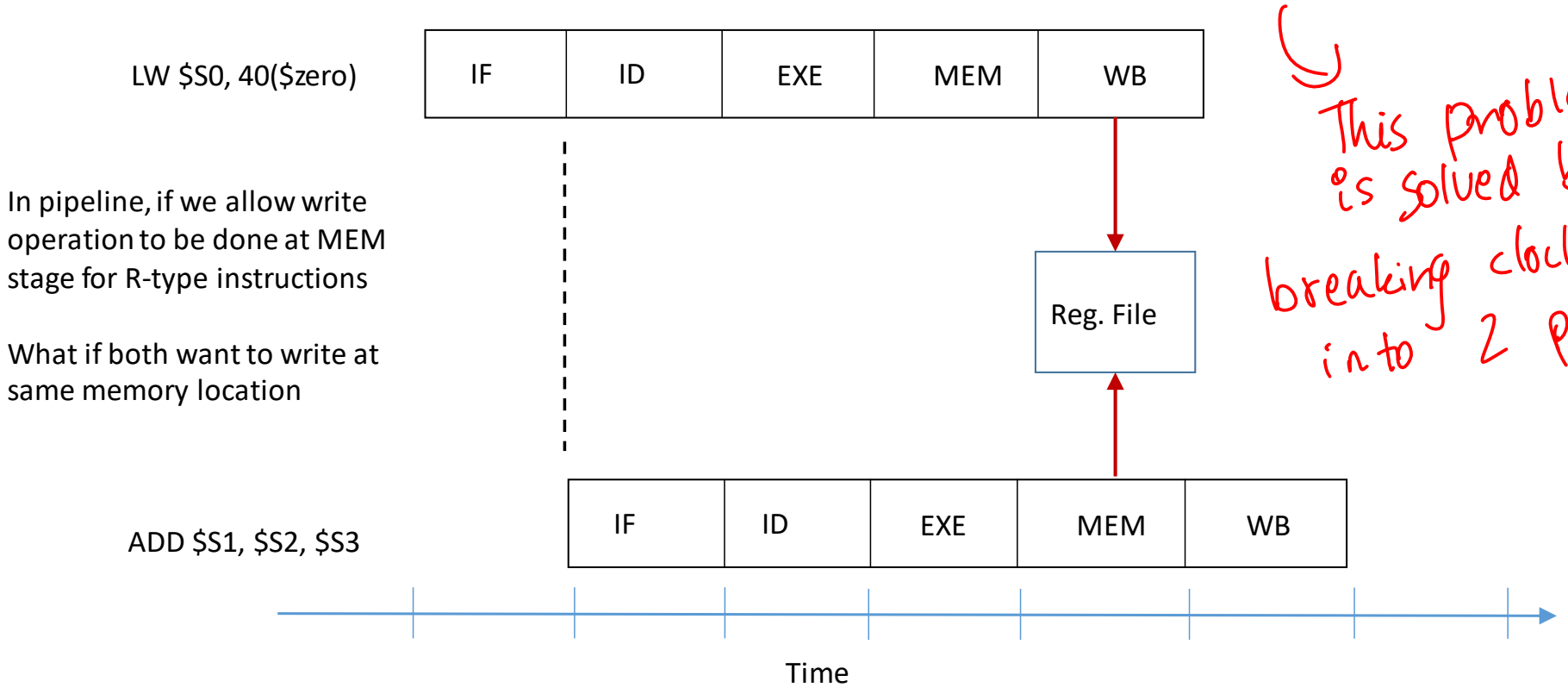
Structural Hazards

- Single memory unit



Structural Hazards

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



This problem is solved by breaking clock cycle into 2 parts.

Solution for Structural Hazard

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

For memory access problem as memory is cheap.

Use this. Break cycle Stall reading.

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 data dependent on $Inst_1$
    - if $Inst_1$ writes its output in a register, Reg (or memory location)
    - $Inst_2$ reads as that as its input
  - $Inst_2$ 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

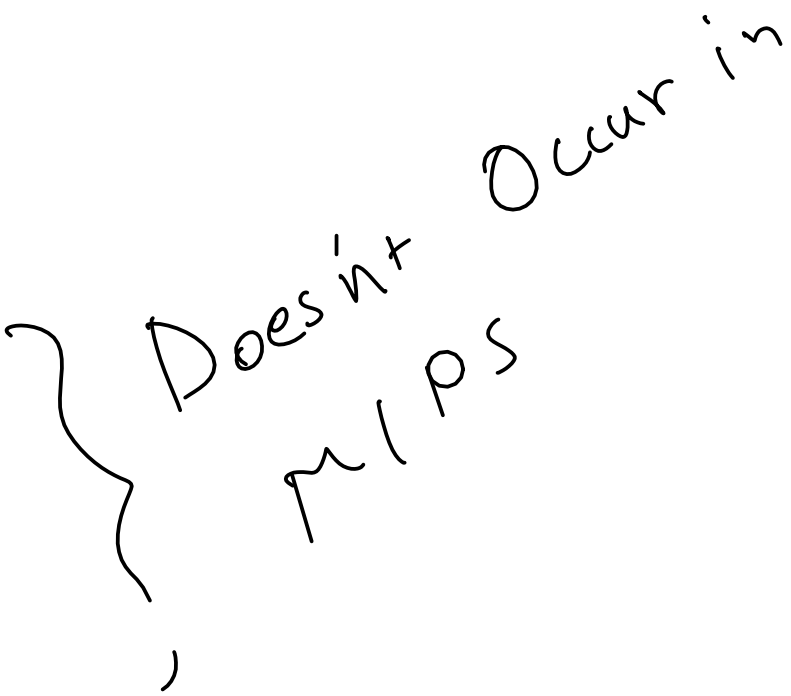
- Assume: $Inst_1$ fetched prior to $Inst_2$
  - $Inst_2$ is output dependent on $Inst_1$
    - if both write in the same register *Reg* (or memory location)
    - $Inst_2$ writes its output after $Inst_1$
  - $Inst_2$ is control dependent on $Inst_1$
    - if $Inst_1$ must complete before a decision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

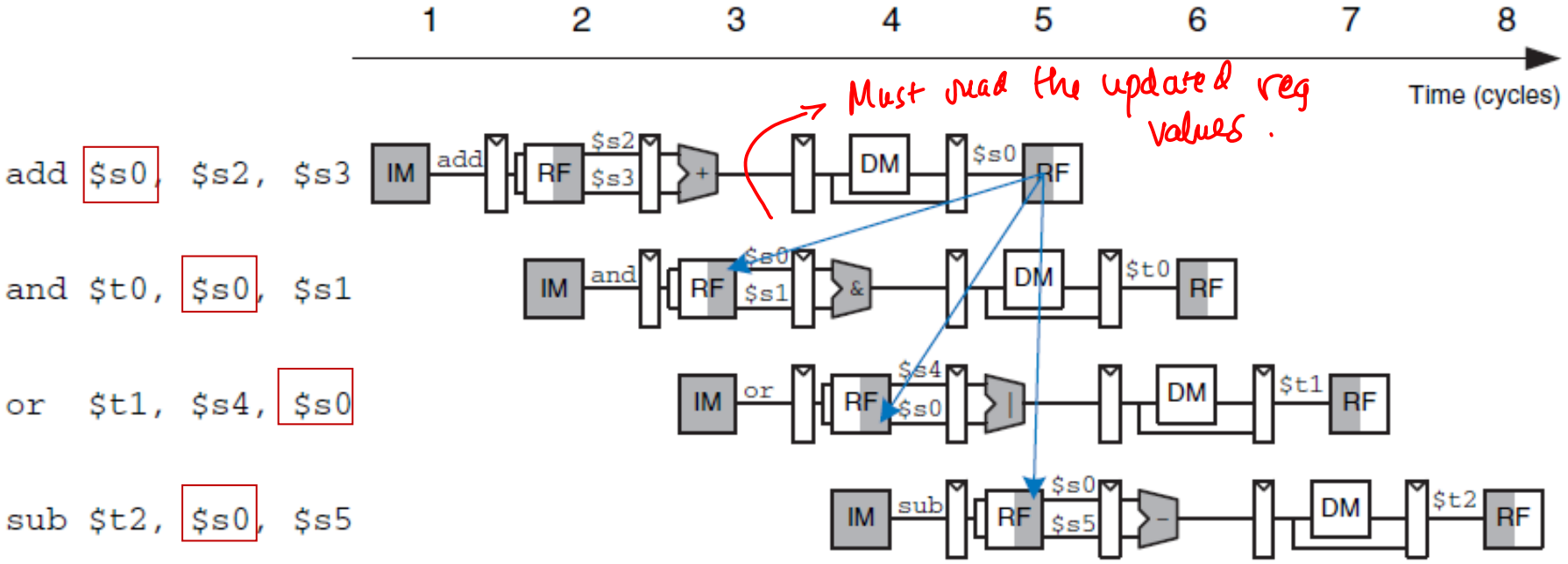
- WAR hazards occur
  - Write stage precedes a read stage

- Will it occur in MIPS?

Data Hazards

- Only RAW hazard occurs in MIPS

↳ read after write



Data Hazards

- How does one solve RAW hazards?

Data Hazards

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

Data Hazards: H/W-based Solution

- Interlocking -- a simple solution
  - Detect the hazard
  - Stall the pipeline → This will cause loss of parallelism.
- Degrade the speedup

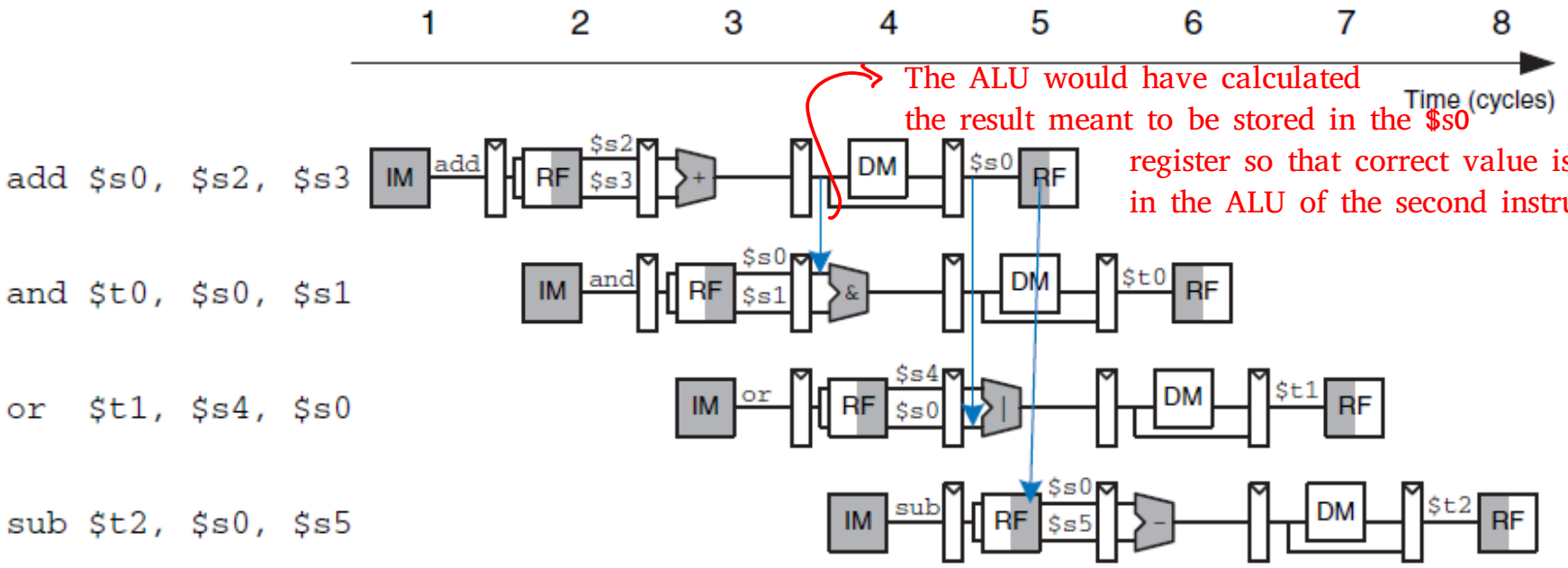
Data Hazards: H/W-based Solution

Use this

- 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 $Inst_2$

Data Hazards: H/W-based Solution

- Forwarding or bypassing



The ALU would have calculated the result meant to be stored in the $s0 register so that correct value is input in the ALU of the second instruction

The register file has already been updated in the first half of the clock. So reading won't be a problem in the second half.

Data Hazards: H/W-based Solution

- How does one perform forwarding or bypassing?

Data Hazards: H/W-based Solution

- How does one perform forwarding or bypassing?

  - Put MUXs in front of ALU select

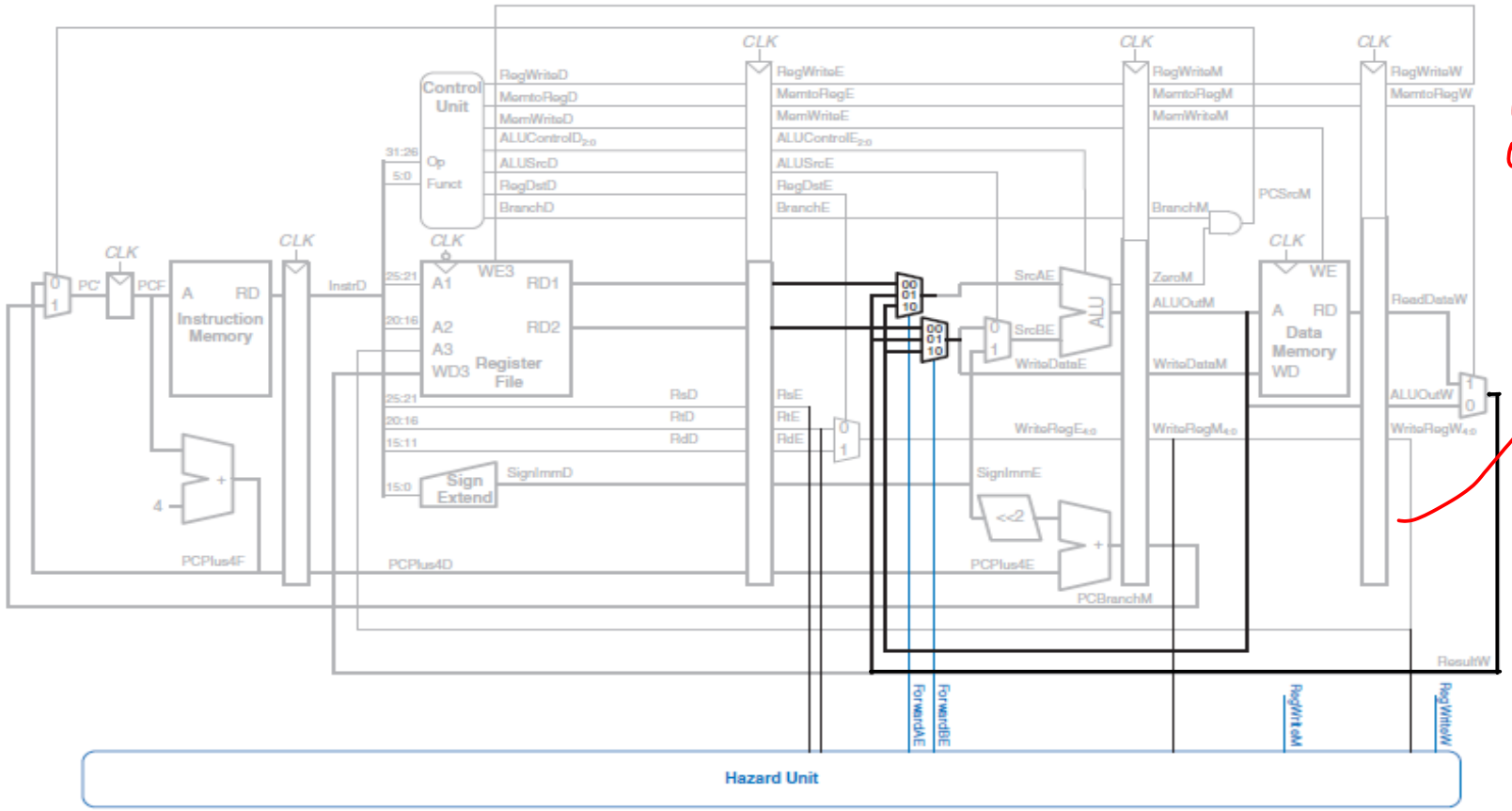
  - ALU's inputs:
    - Register file or Decode stage
    - Memory stage
    - Writeback stage

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



00 – rs, rt
01 – ALU
10 – From this latch.

For both the source registers rs and rt, they have been muxed with the ALU output.

Data Hazards: H/W-based Solution

- Hazard unit generates control signals for
  - Mux SrcA [ForwardAE]
  - Mux SrcB [ForwardBE]
- The control logic for ForwardAE
- if ((rsE!=0) AND (rsE == WriteRegM) and RegWriteM) then
  ForwardAE = 10
- else if ((rsE!=0) AND (rsE == WriteRegW) and RegWriteW) then
  ForwardAE = 01
- else
  ForwardAE = 00

Write Reg is register to be written in register file (see in latch of MEM stage)

See in the latch storing info for MEM stage

ALUOut goes in

See in latch of write stage

Output from Data Memory goes in.

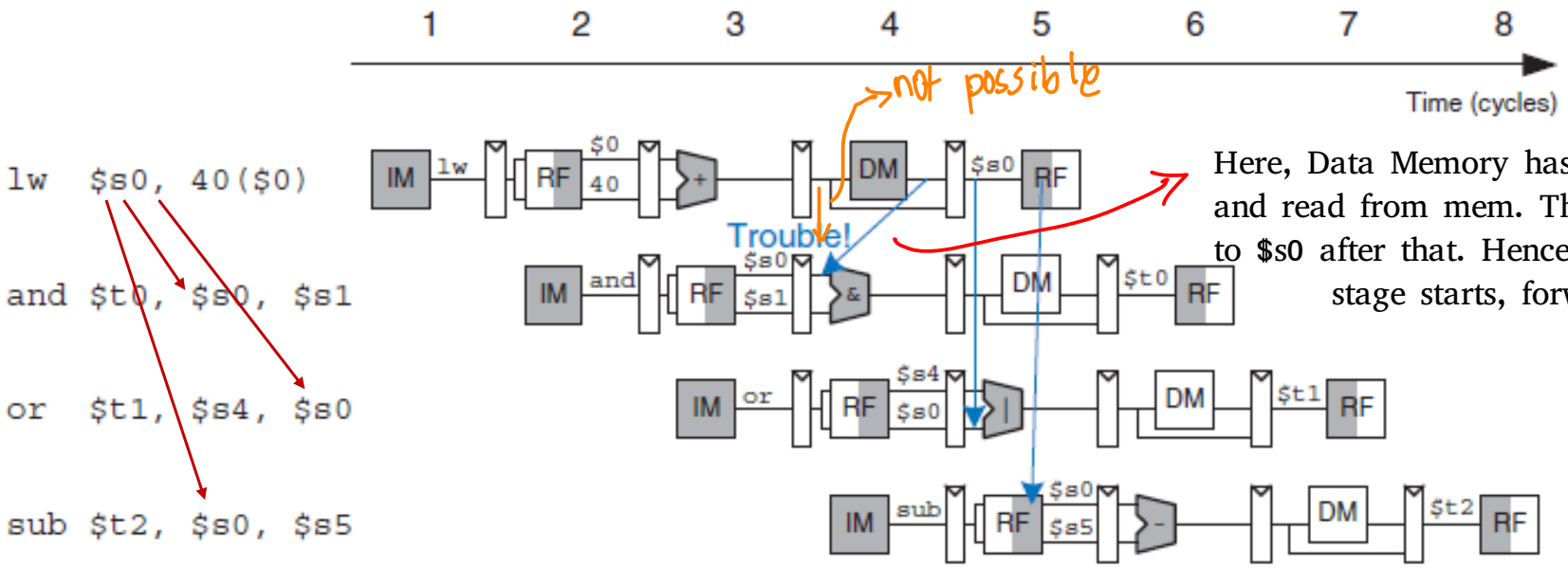
rs, rt ftw.

Data Hazards: H/W-based Solution

- Hazard unit generates control signals for
  - Mux SrcA [ForwardAE]
  - Mux SrcB [ForwardBE]
- The control logic ForwardBE (same as in ForwardAE except rt, instead of rs)
- if ((rtE!=0) AND (rtE == WriteRegM) and RegWriteM) then
  ForwardBE = 10
- else if ((rtE!=0) AND (rtE == WriteRegW) and RegWriteW) then
  ForwardBE = 01
- else
  ForwardBE = 00

Data Hazards

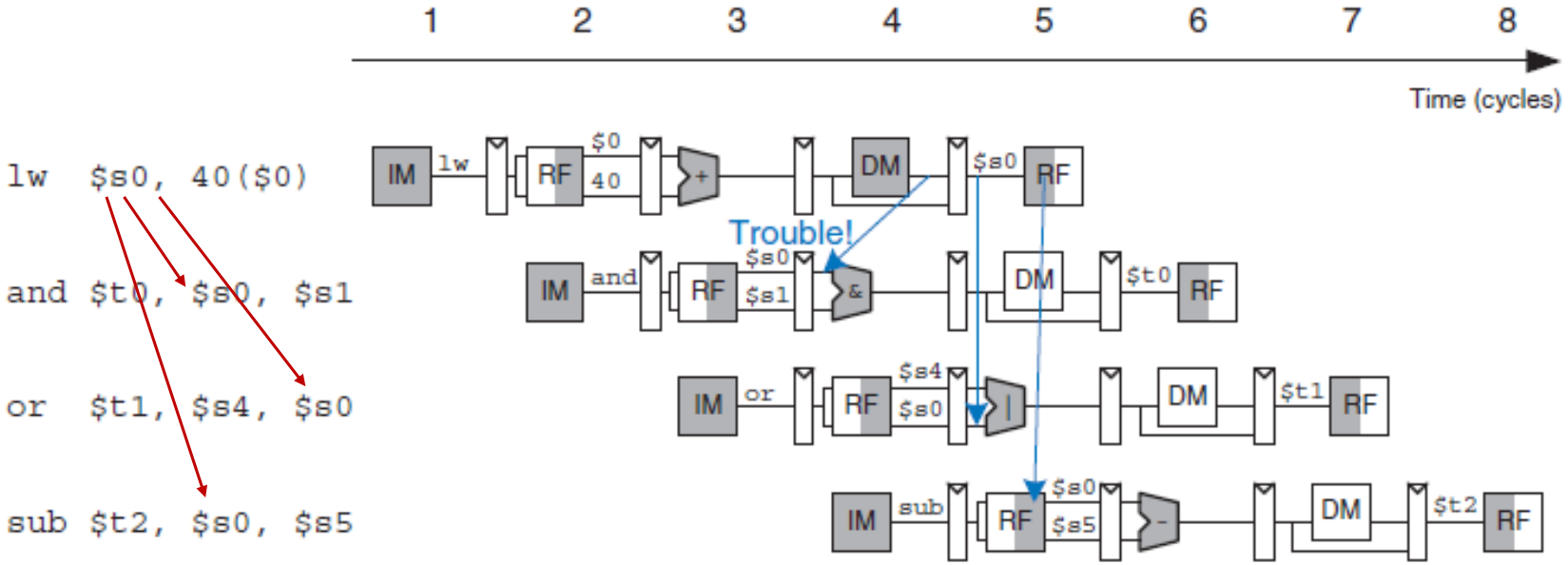
- Can we solve this one by forwarding?
- Is lw-instruction only causes this?



Here, Data Memory has to process the address and read from mem. This value is written to **$s0** after that. Hence, as soon as the MEM stage starts, forwarding can't be done.

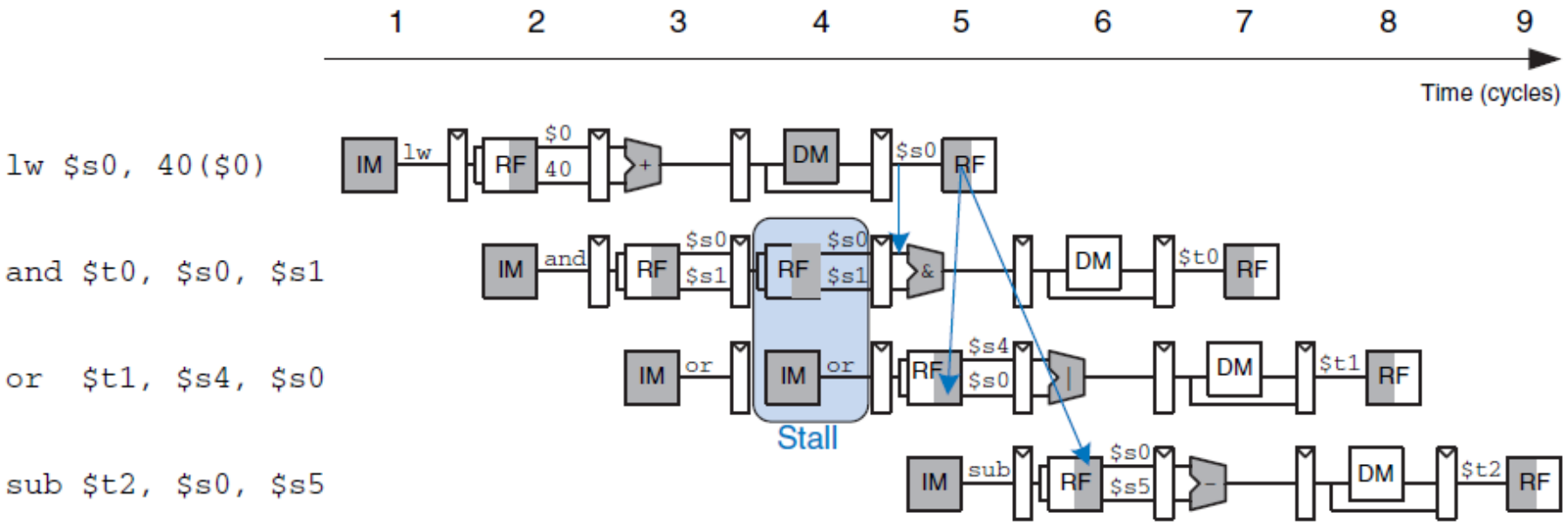
Data Hazards

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



Data Hazards

- Forwarding with pipeline interlocking (stalling)



Have to stall all the following instructions. For this software implementation is NOP

Data Hazards

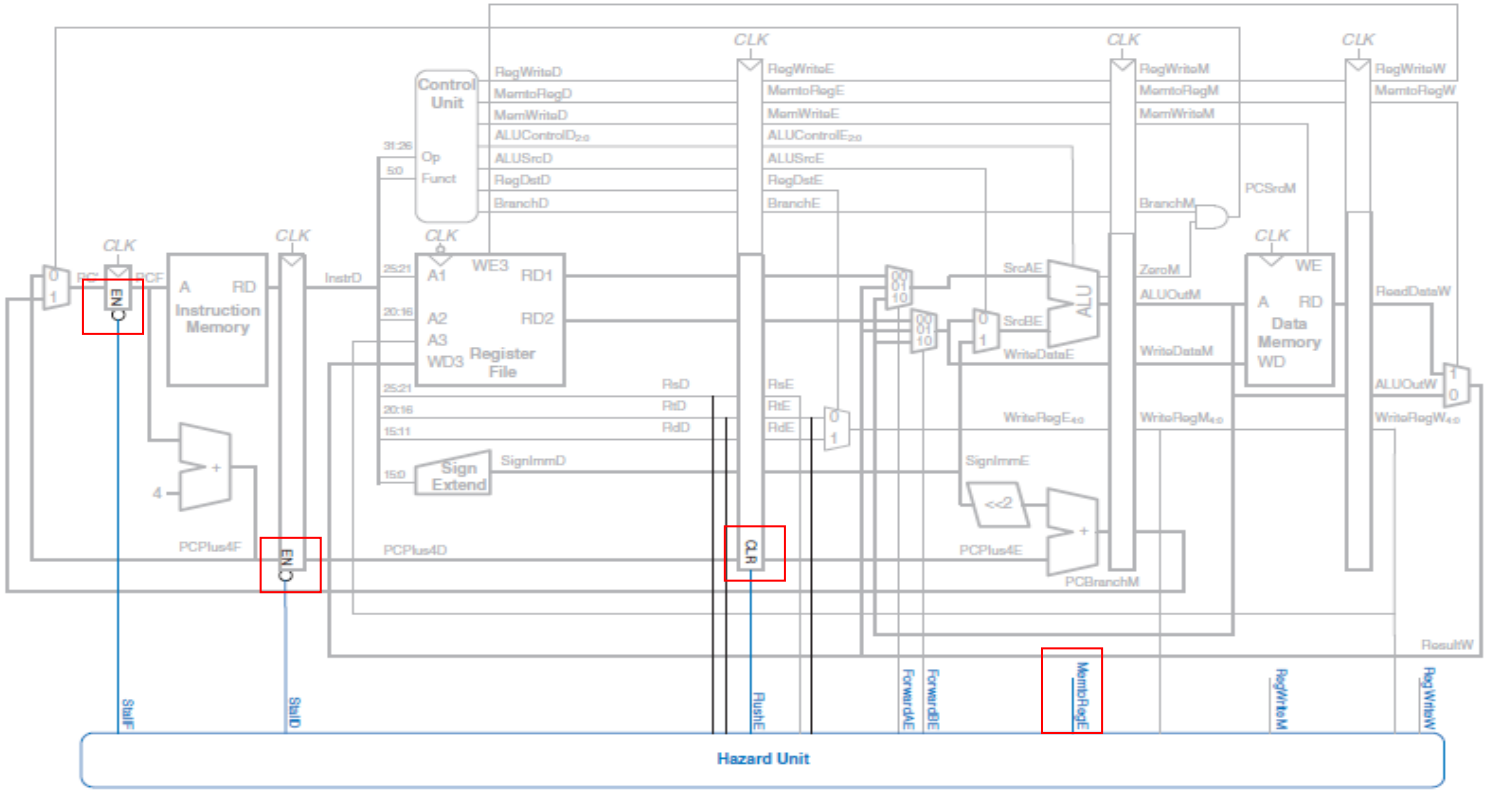
- Forwarding with pipeline interlocking (stalling)
- Hazard unit checks
  - Is it the lw-instruction?
    - Destination register (rtE) matches with
    - Source operand in the Decode stage (rsD or rtD)
    - Stall the Decode stage (how long?)
- How does one perform stall operation?

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 Switch off the program counter
  - StallD = lwstall Switch off the latch after instruction stage
  - FlushE = lwstall Clear the values in the latch after decode stage

Note that rt value provides the reg no. used by lw.

Data Hazards

- What if hazard unit unable to detect

Data Hazards

- What if hazard unit unable to detect
  - Compiler has to control
    - noop-instruction
    - 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 R3, 54($0)
	NOOP	
	ADD R1, R2, R3	
	SW R3, 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).

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
  - Compiler-based technique