TUTORIAL 11 THE PIPELINED PROCESSOR

Overview

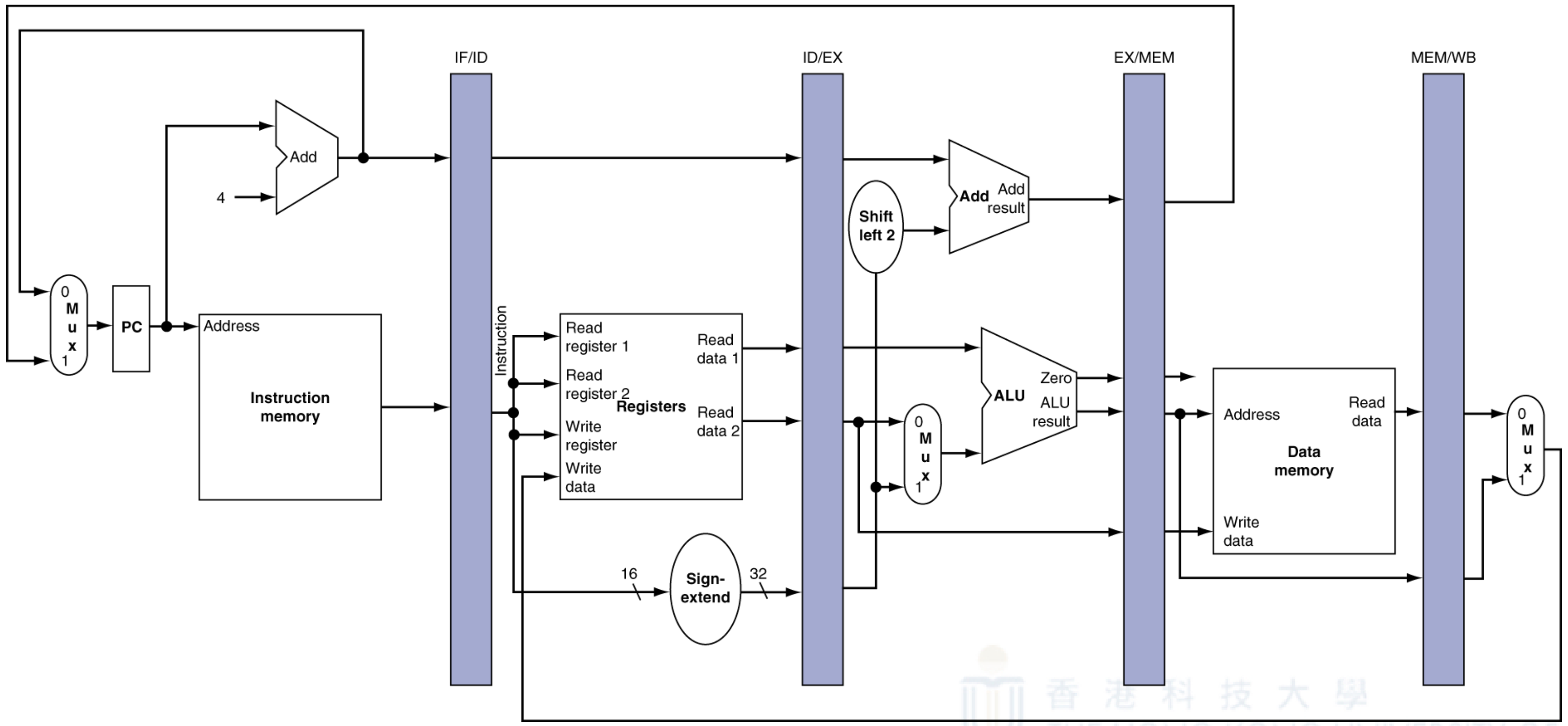
- We will review the following concept in this tutorial:
- MIPS pipeline datapath and control (ideal case)
  - IF, DE, EXE, MEM, WB stages
  - Inter-stage registers
- Pipeline hazards
  - Structural hazards, data hazards and control hazards
- Identification and solution for hazards
  - Forwarding and pipeline stalls
  - Code re-ordering

MIPS Pipeline Stages

- Execution of each instruction is broken into <u>5 stages</u>: (in the order of execution)
  - IF : Fetch the instruction from memory
  - ID : Instruction decode & register read
  - EX : Perform ALU operation
  - MEM : Memory access (if necessary)
  - WB : Write result back to register
- Each stage uses a <u>different hardware unit</u> and takes <u>one clock cycle</u> to complete.
- Instructions can **co-exist** in the datapath if all of them are in different stages of execution from one another

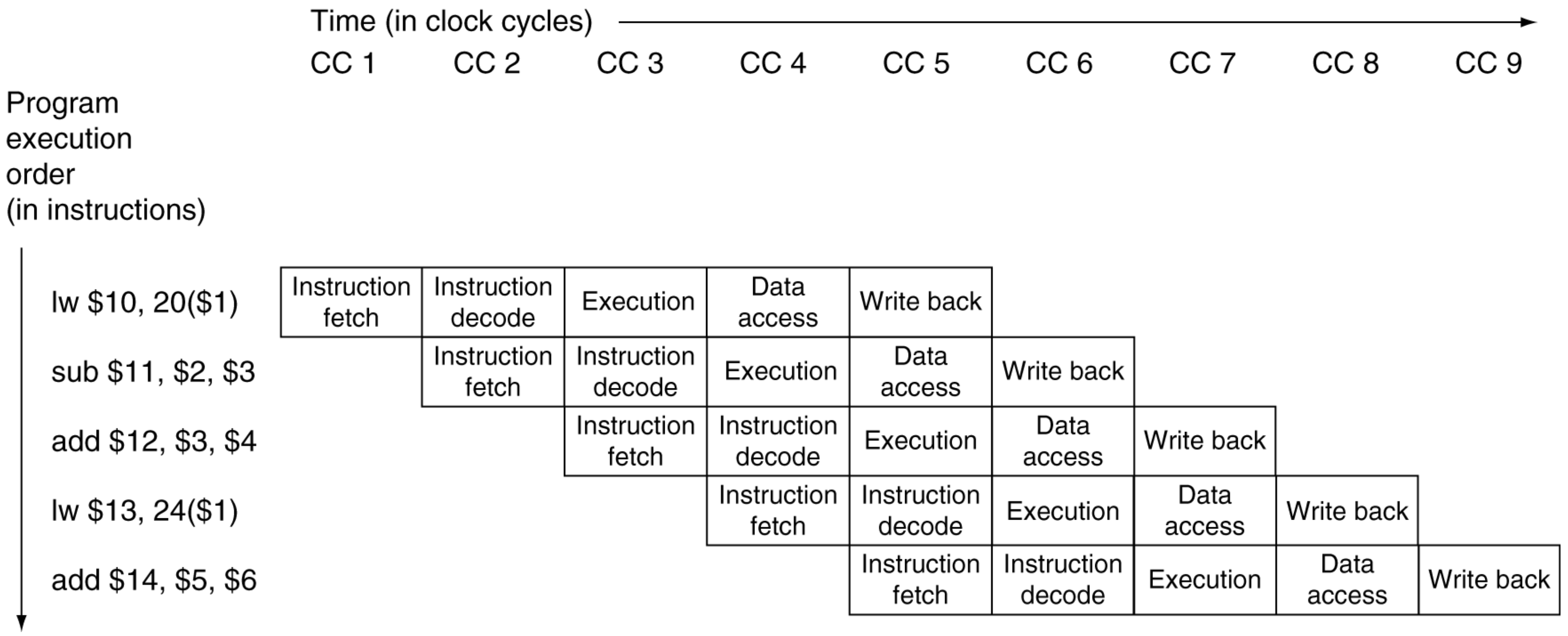
Pipeline Registers

- Additional pipeline registers are needed
- Located between the stages, i.e. IF/ID, ID/EX, EX/MEM, MEM/WB
- Hold information produced in the previous cycle



Multi-clock-cycle pipeline diagram : traditional view

- The following diagram shows the execution of a series of instructions in the ideal pipeline (no hazards)

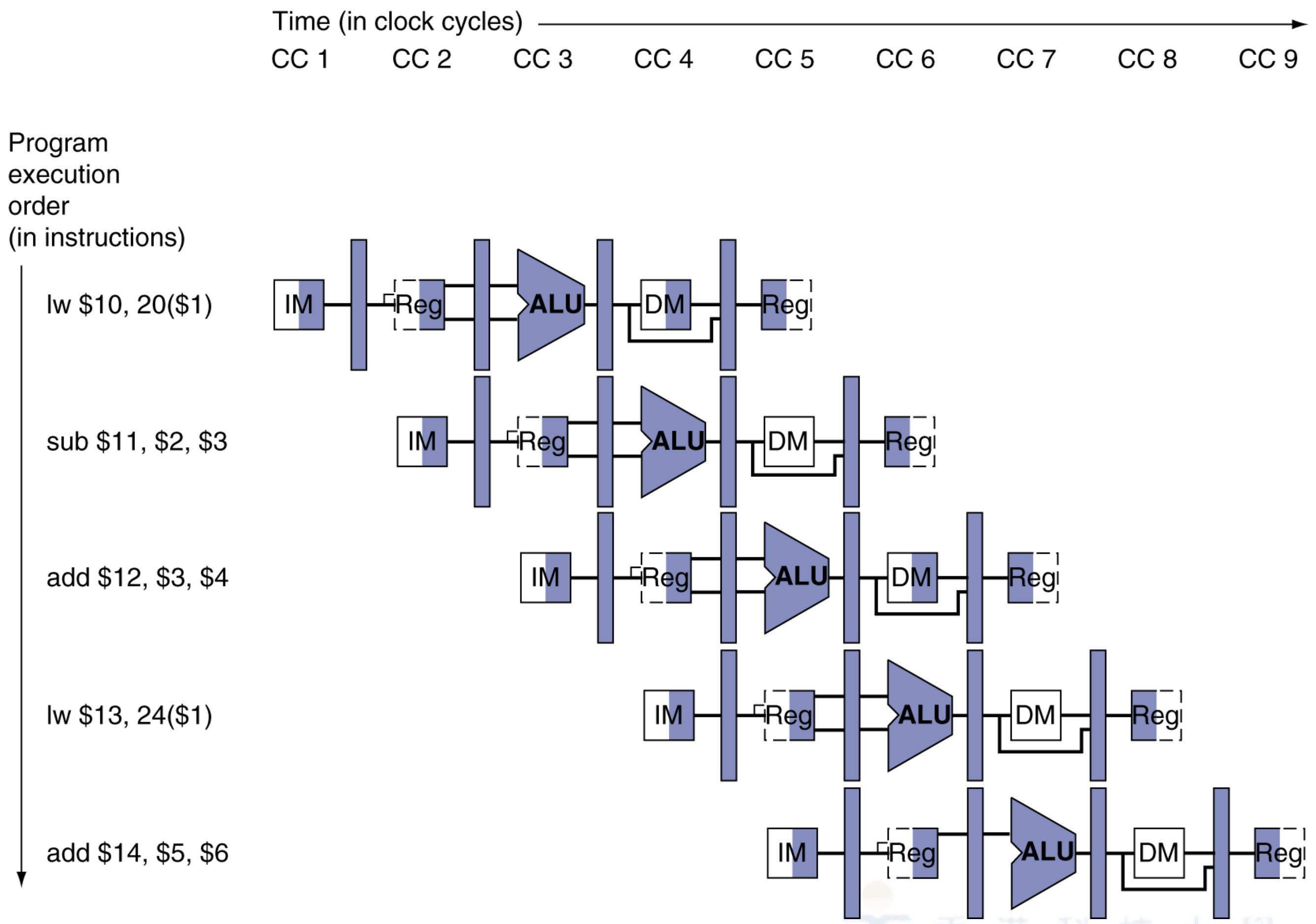


Time (in clock cycles)

Program execution order (in instructions)	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw \$10, 20(\$1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub \$11, \$2, \$3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add \$12, \$3, \$4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw \$13, 24(\$1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add \$14, \$5, \$6					Instruction fetch	Instruction decode	Execution	Data access	Write back

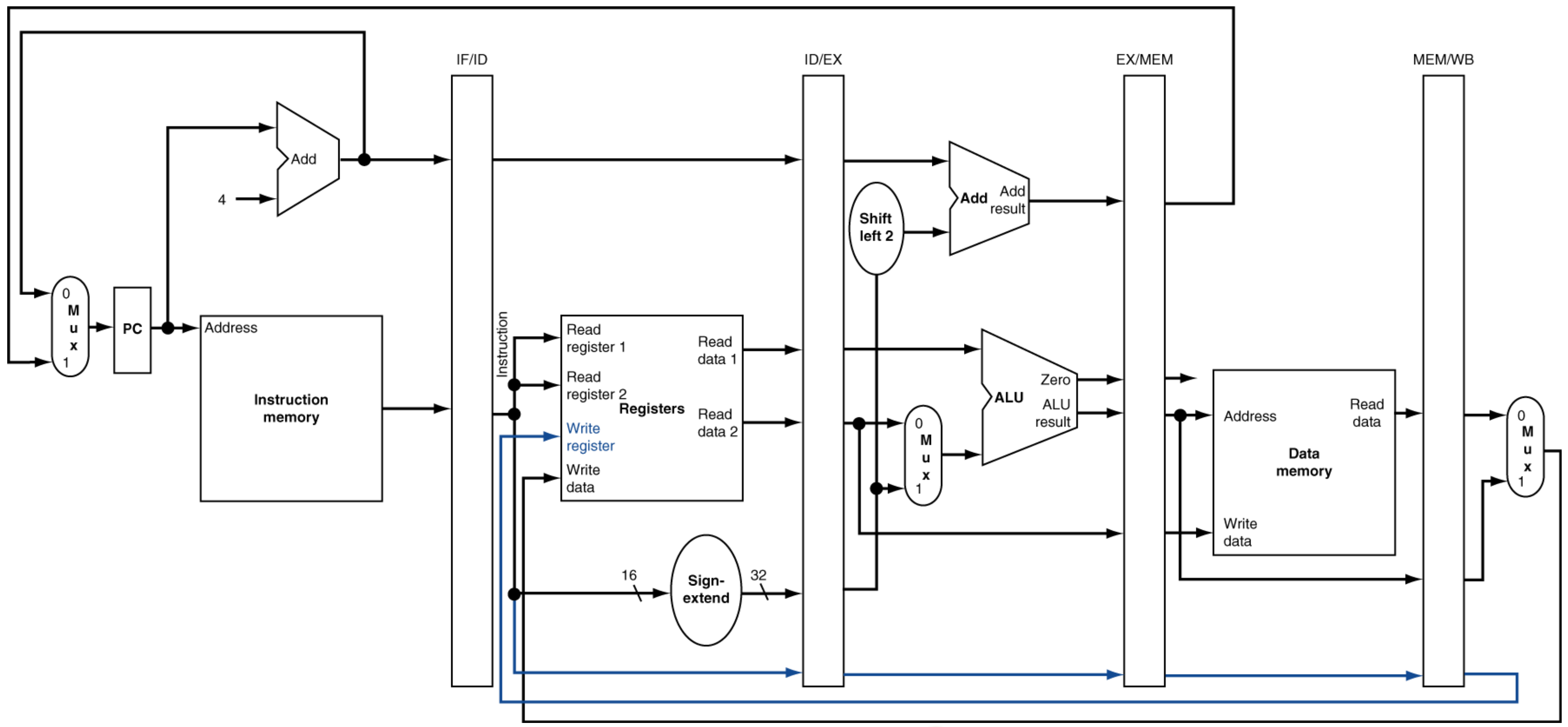
Multi-clock-cycle pipeline diagram: graphical view

- The multi-clock-cycle form showing the resource usage.



The Corrected Datapath for lw

- To solve this problem: **the "write register" information is forwarded from the MEM/WB pipeline registers.**



Pipeline Hazards

- **Hazards are situations in pipelining when the next instruction cannot be executed in the following clock cycle.**

- **Three types of pipelined hazards**
  - **Structural hazards:** A required resource is busy
    - Already solved in modern processor
  - **Data hazards:** Need to wait for previous instruction to complete its data read/write
  - **Control hazards:** Deciding on control action depends on previous instruction

- **Hazards can always be resolved by waiting. But this slows down the pipeline.**

Data Hazards in ALU Instructions

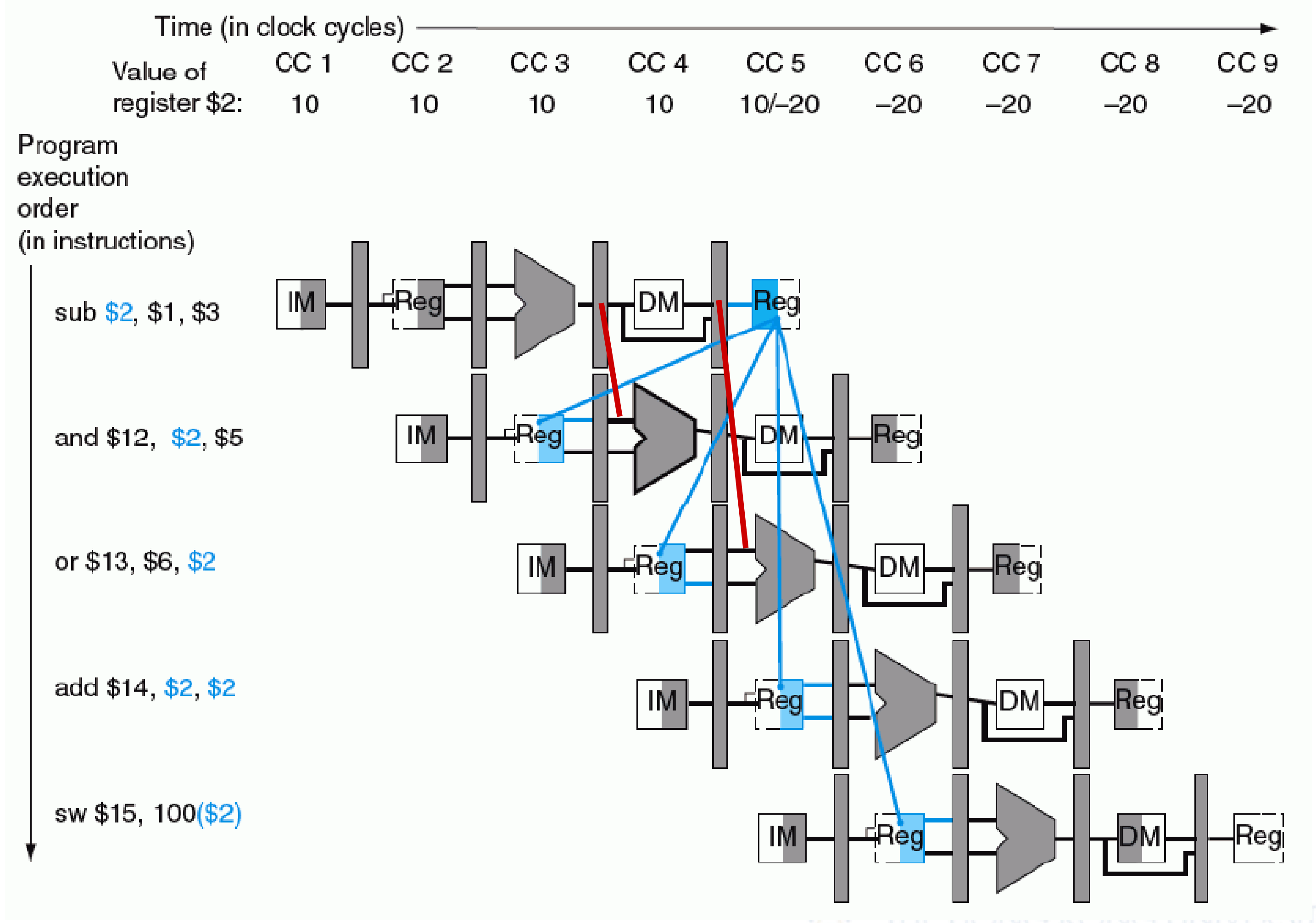
- Consider this sequence:

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- We can resolve (some) hazards with forwarding

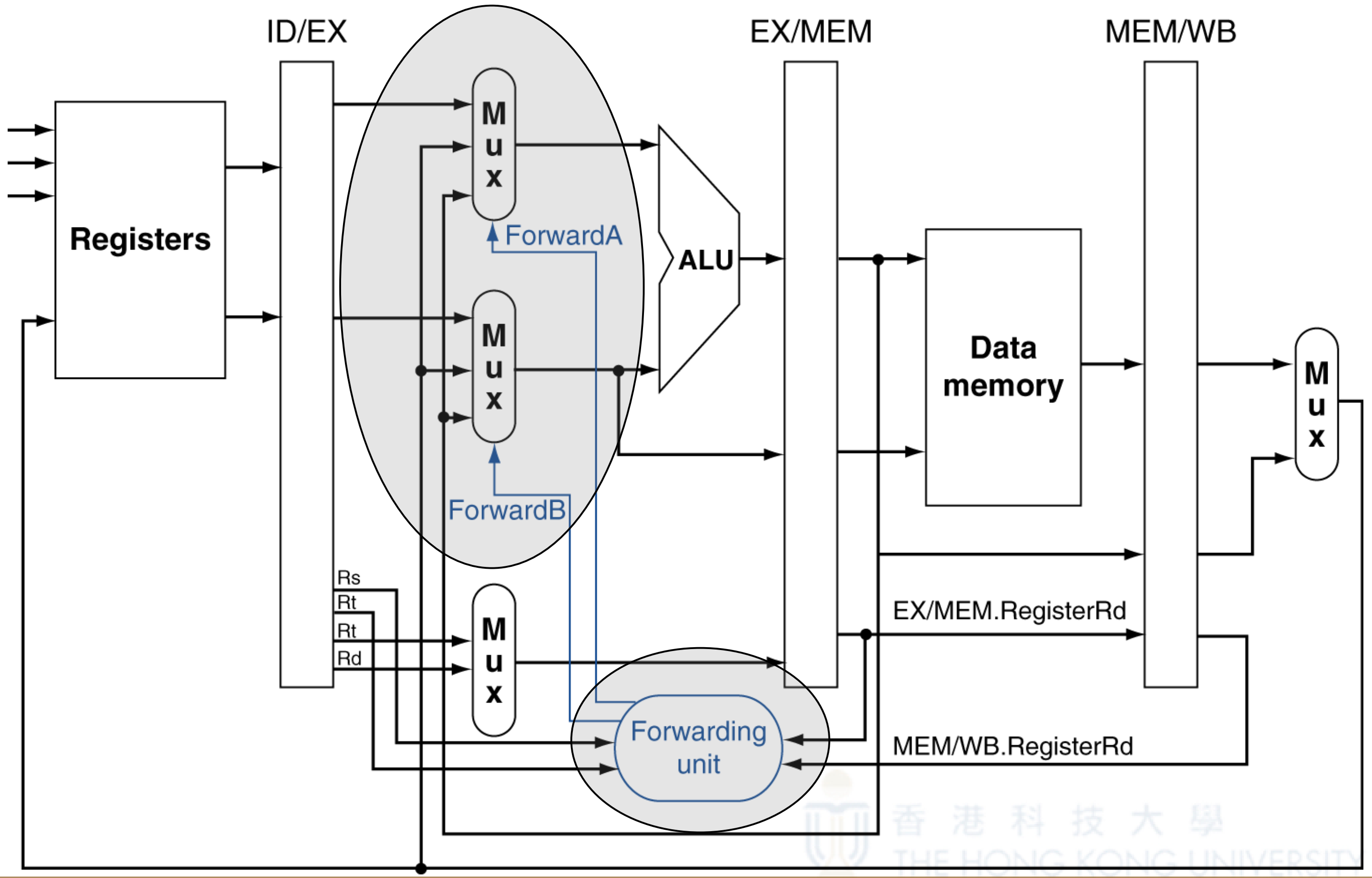
Dependencies and Forwarding

- From the figure the decision is simple (required "forwardings" are represented by the two red lines):

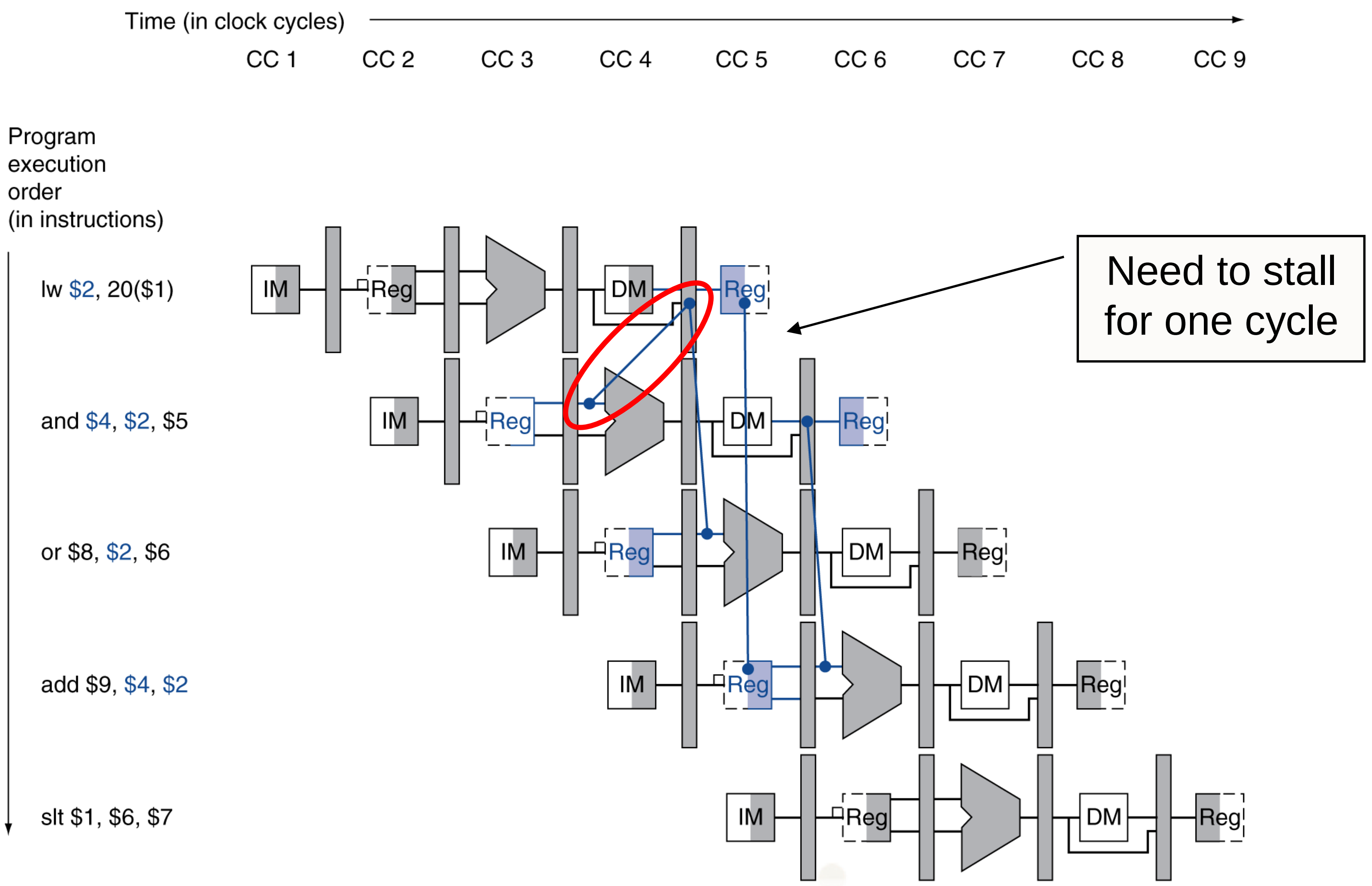


Forwarding Paths

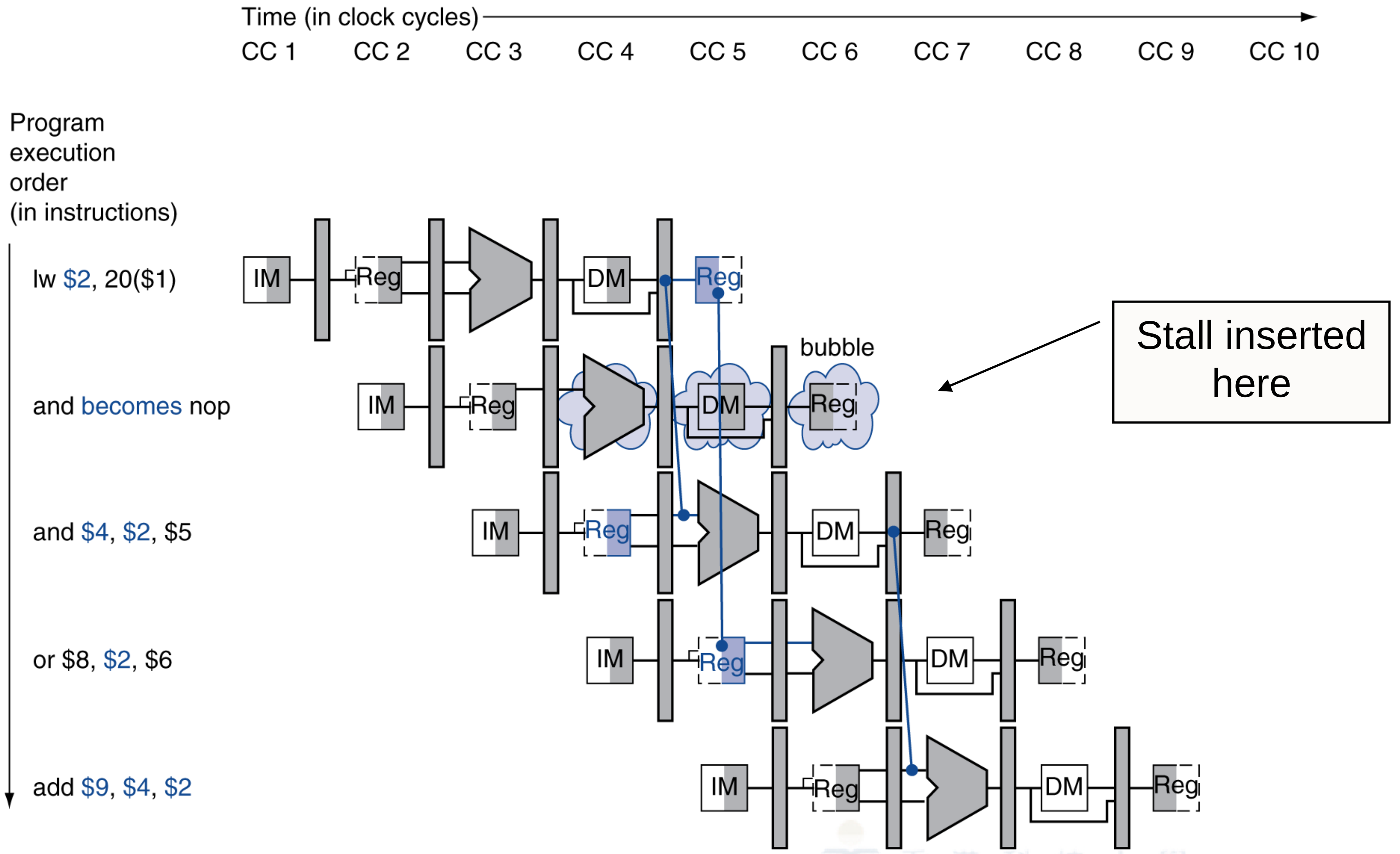
- **Forwarding always takes place to EX stage**
  - Using two multiplexers to decide what is the input of operands A and B of the ALU
  - Forwarding control unit makes the decision



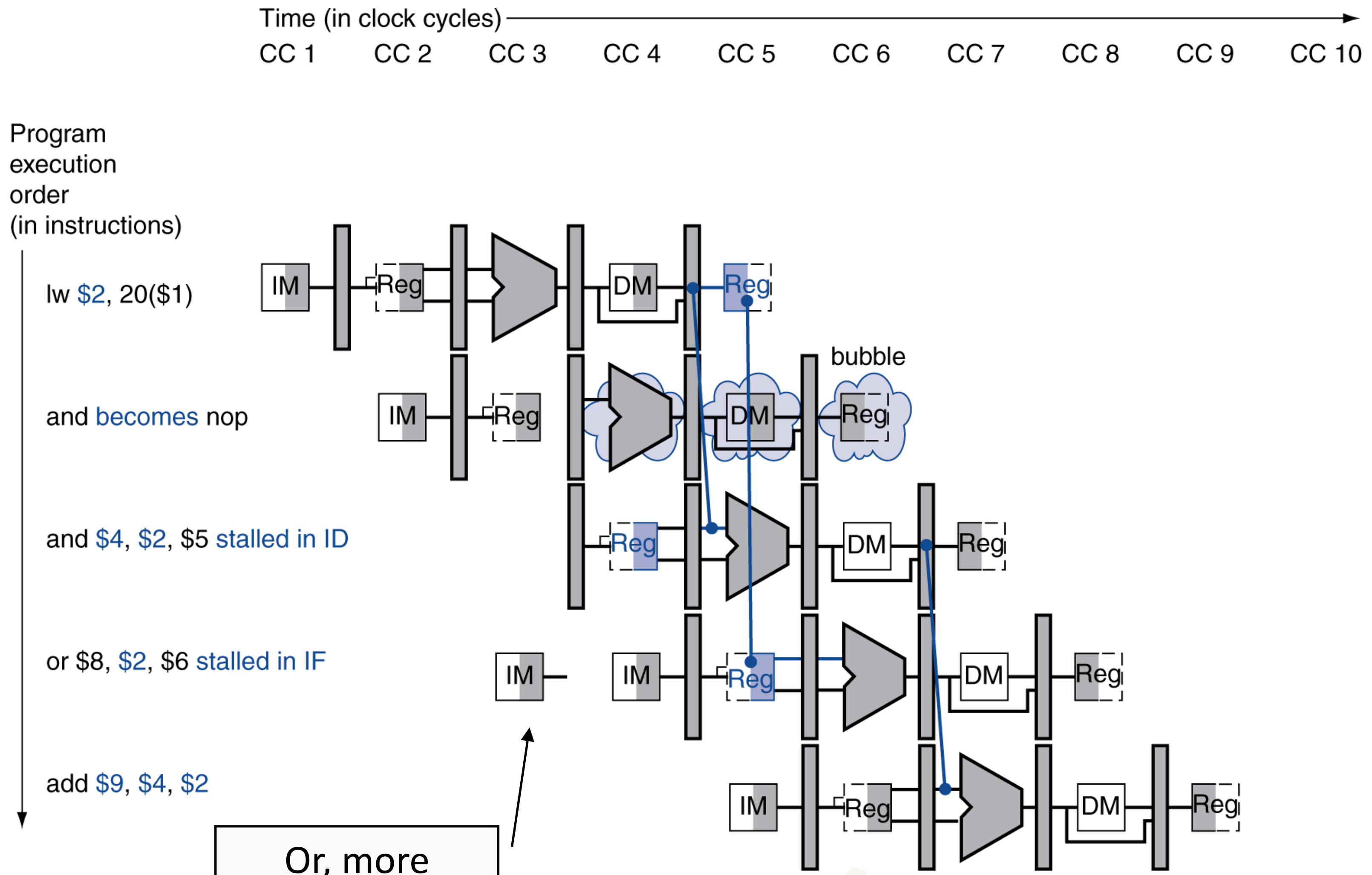
Load-Use Data Hazard



Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline (cont.)



Or, more accurately…

Exercise 1 Hazard Detection

- Identify if there are any pipeline hazards in each of the following sequences of MIPS instructions
- Specify the type of the hazard (if any) and explain its cause

a)
```
sw  $s1, 0($t0)
add $s2, $s0, $s1
add $s2, $s3, $s4
```

b)
```
lw  $s1, 0($t0)
add $s2, $s2, $s2
add $s2, $s3, $s3
```

c)
```
lw $s1, 0($t0)
lw $s2, 4($s1)
add $s2, $s3, $s3
```

d)
```
lw $s1, 0($t0)
sub $s3, $s4, $s2
bne $t0, $t1, target
add $s2, $s5, $s6
```

e)
```
add $s1,$s1,$2
add $s1,$s1,$3
add $s1,$s1,$s4
```

Exercise 2 Pipeline Stalls

- Suppose the pipeline uses no forwarding, register writes in the first half of the cycle and register reads in the second half.
- Fill in the table below with the appropriate pipeline stages (IF, ID, EXE, MEM, WB) or bubbles (BUB).

`sub $s1, $t0, $t1`	IF	ID	EXE	MEM	WB							
`lw  $s2, 0($s1)`												
`add $s5, $s3, $s4`												
`add $s7, $s6, $s5`												

Exercise 3 Forwarding

- Assume there are forwarding paths from the output of the ALU to the inputs of the ALU, and from memory output to inputs of the ALU
- Fill in the table below with the appropriate pipeline stages (IF, ID, EXE, MEM, WB) or bubbles (BUB). Mark the forwarding if it's used.

`sub $s1, $t0, $t1`	IF	ID	EXE	MEM	WB					
`add $s2, $s0, $s1`										
`add $s5, $s3, $s4`										
`add $s5, $s6, $s2`										

Exercise 4 Code Re-ordering (self-practice)

- Some data dependency can be resolved by re-ordering the instructions
- For following sequence of instructions, calculate the total number of clock cycles needed with and without code re-ordering

```
sub $s1, $t0, $t1
add $s2, $s0, $s1
add $s5, $s3, $s4
add $s5, $s6, $s6
```

Exercise 5 Forwarding and Pipeline Stalls (self-practice)

- Assume there is only forwarding path from the output of the ALU to the inputs of the ALU
- Fill in the table below with the appropriate pipeline stages (IF, ID, EXE, MEM, WB) or bubbles (BUB). Mark the forwarding if it's used.

`lw $s1, 4($s0)`	IF	ID	EXE	MEM	WB						
`add $s2, $s3, $s4`											
`add $s5, $s1, $s2`											
`add $s7, $s5, $s2`											