PIPELINED PROCESSOR (II)

Lecturer: Hui Annie Guo
h.guo@unsw.edu.au
K17-501F

Lecture overview

- Topics
  - Overview of pipeline hazards
  - Design solutions to structural hazard
  - Design solutions to data hazard

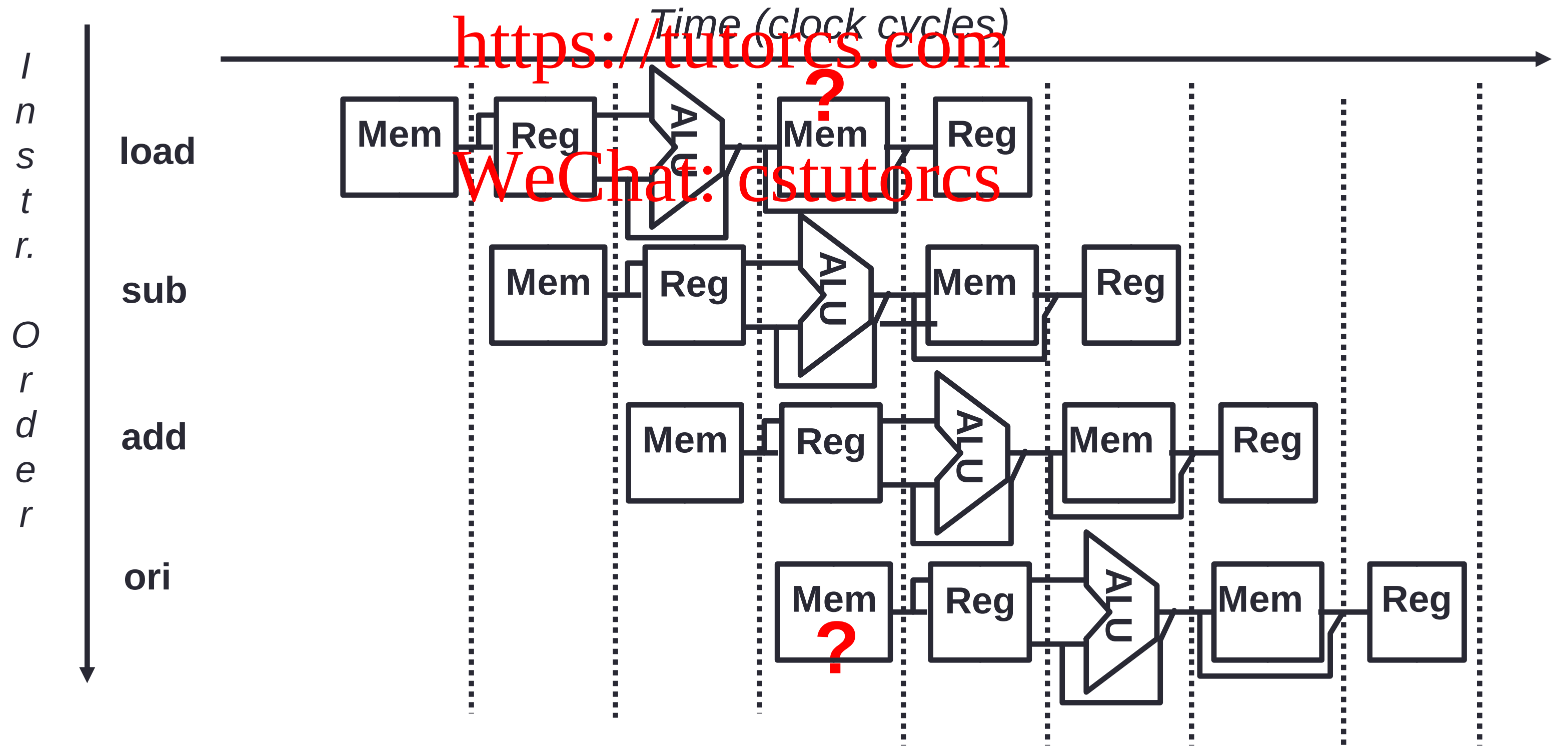
- Suggested reading
  - H&P Chapter 4.7

Hazards

- **Pipeline hazards are the situations where instruction execution cannot proceed in the pipeline.**
  - **Structural hazard**
    - **Required resource is busy.**
  - **Data hazard**
    - **Need to wait for previous instruction to update its data.**
  - **Control hazard**
    - **The control decision cannot be made till the condition check by the previous instruction is completed.**

Structural hazard

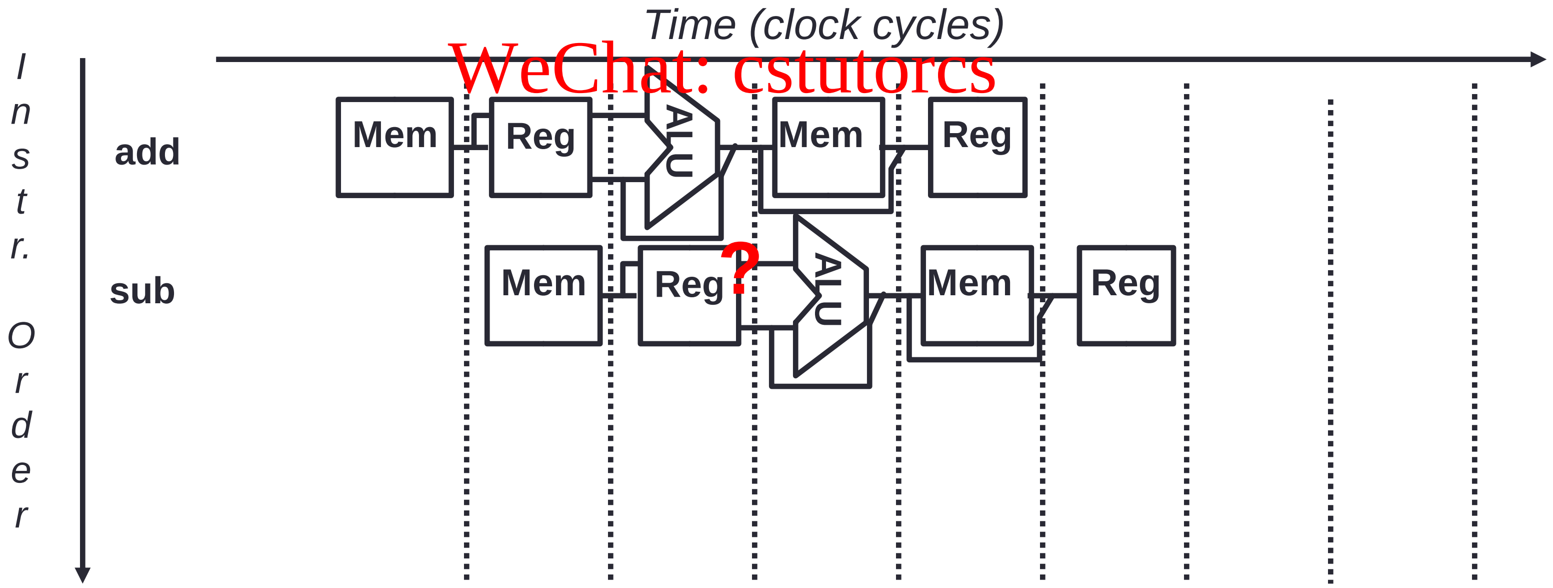
- Resource is not available
  - Example
    - In MIPS pipeline if a single memory is used for both instruction and data, both Instruction Fetch and Load/Store will compete for memory access in the same clock cycle



Data hazard

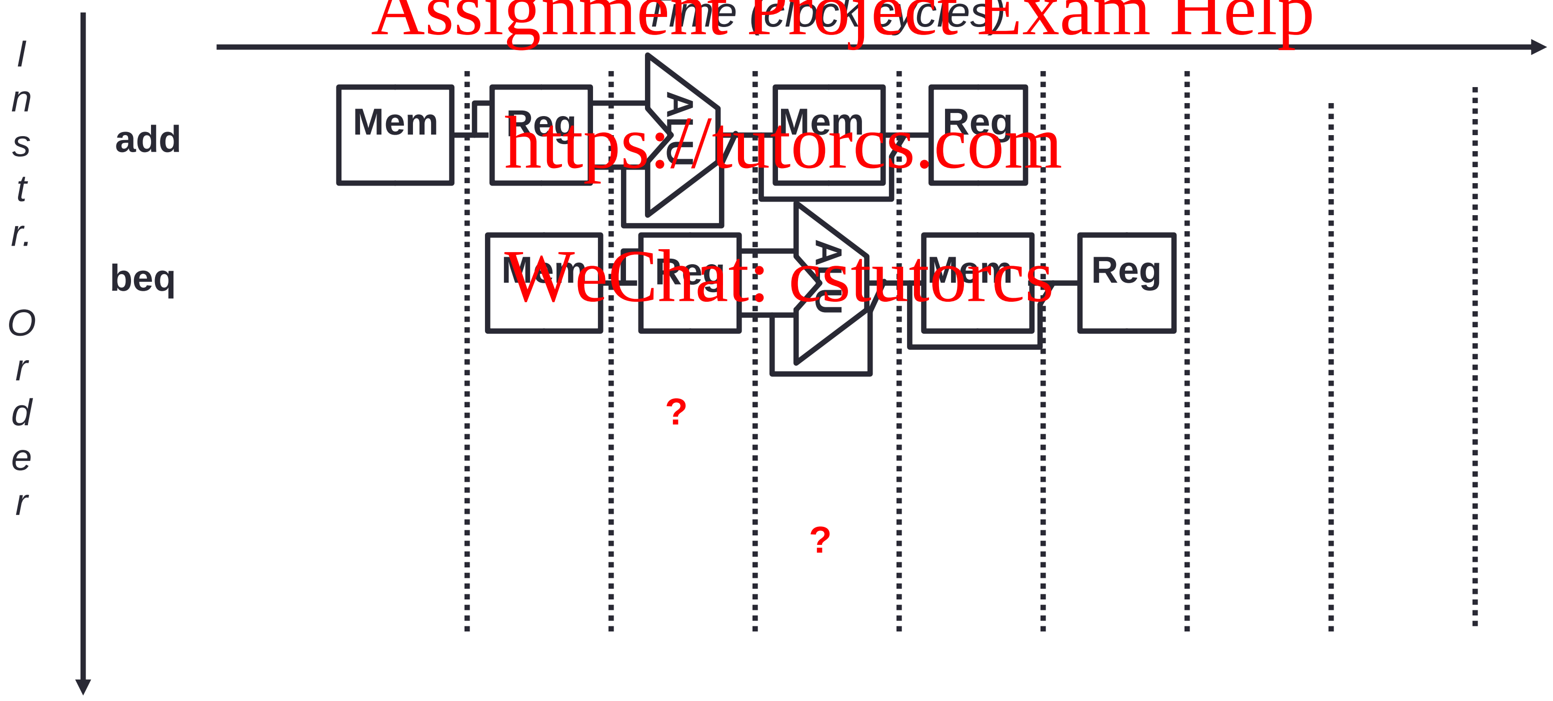
- **Data is not available**
  - **Example**

```
add     $s0, $t0, $t1
sub     $t2, $s0, $t3
```



Control hazard

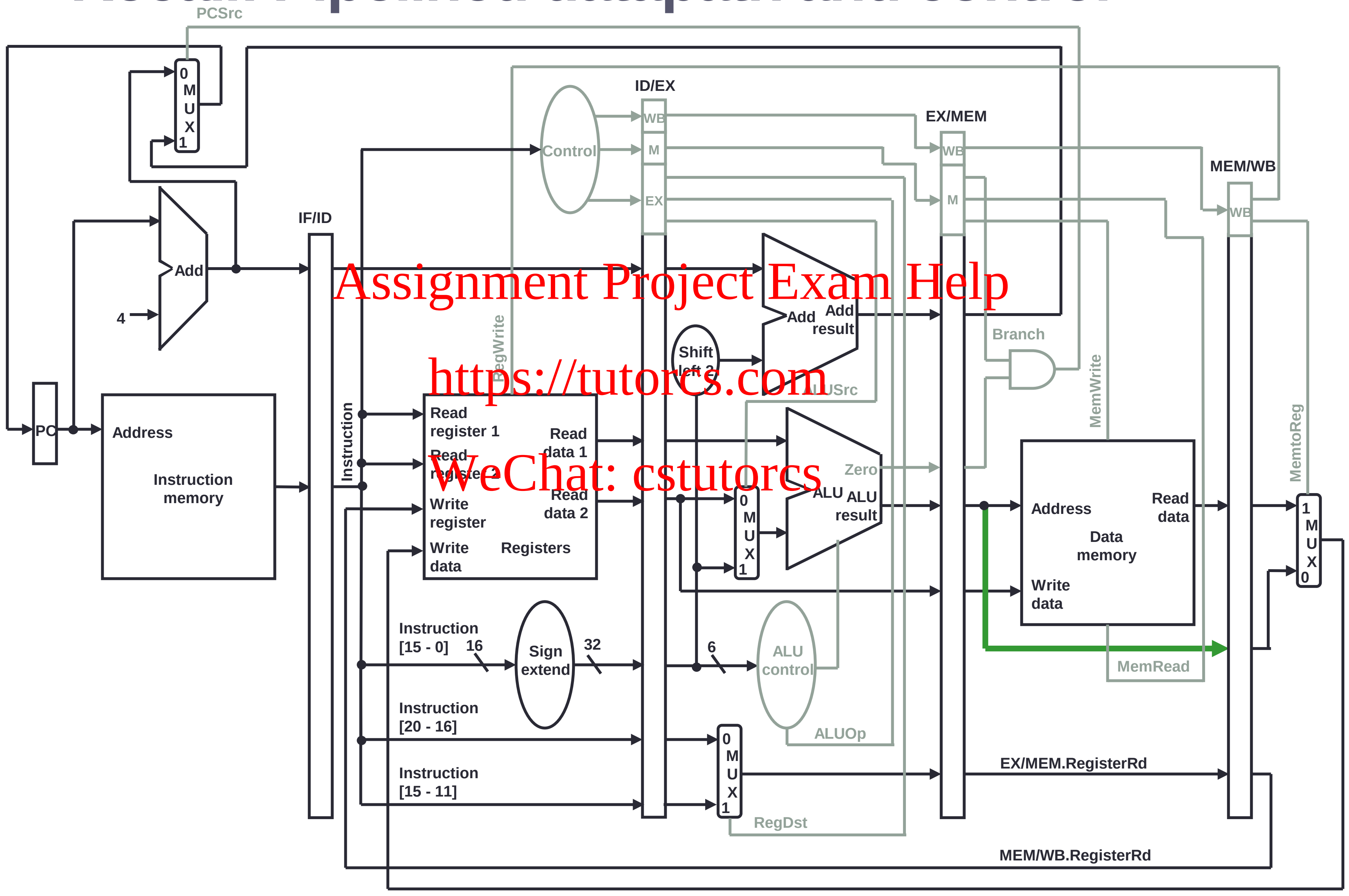
- **Next instruction is not available**
  - **Fetching next instruction depends on branch condition**



Design solutions to structural hazard

- **Structural hazards occur when one resource is to be used by more than one instruction in one cycle**
- **Solutions:** Never access a resource more than once per cycle
  - **Allocate each resource to a single pipeline stage**
    - **Duplicate resources if necessary e.g. IMEM/DMEM**
  - **Every instruction must follow the same sequence of cycles/stages**
    - **Skipping a stage can introduce structural hazards**
      - **e.g R-type instructions cannot skip MEM stage**
    - **Unnecessary trailing cycles/stages can be dropped**
      - **e.g. BRA/JMP don’t need to complete WB (or even MEM)**

Recall: Pipelined datapath and control



In-class exercises (1)

- **If the connection line in green in the previous slide is removed and the R-type instruction is allowed to skip the Mem stage, how are the following two instructions executed in the pipeline?**

LW $1, $2, $3

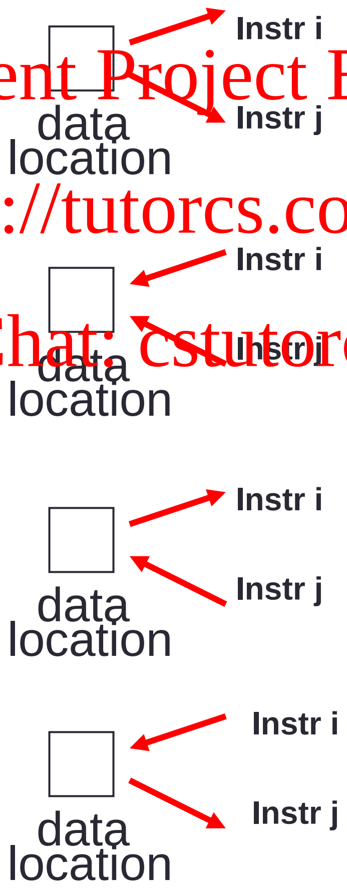
ADDU $4, $5, $6

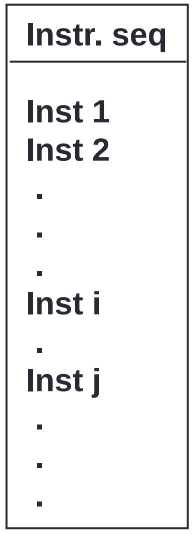
In-class exercises (2)

- **Assume there is only one memory for both instruction and data in the pipelined processor. Assume there are 20% load/store instructions. What is the average CPI due to this type of hazards?**

Data hazard

- **Data hazards are caused by data dependency**
- **There are four types of data dependency between instructions**
  - **RAR**
    - **Read after Read**
  - **WAW**
    - **Write after Write**
  - **WAR**
    - **Write after Read**
  - **RAW**
    - **Read after Write**





In-class exercises (3)

Identify all data dependencies in the following code segment.

```
add   $2,   $5,   $4
add   $4,   $2,   $5
sw    $5,   100($2)
add   $3,   $2,   $4
```

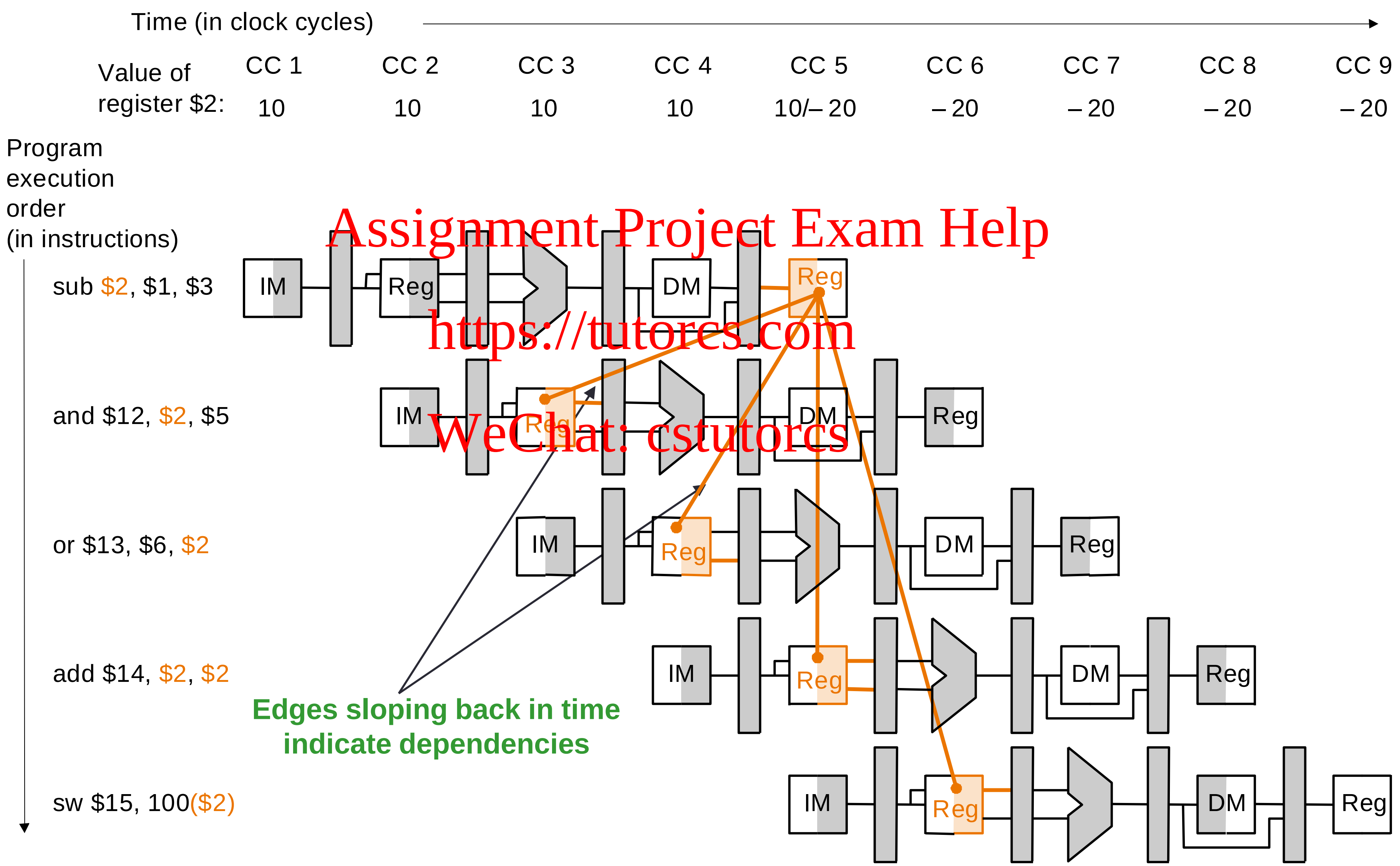
Design solutions to data hazard

- **Avoid some hazards by properly partitioning/scheduling “tasks” in the pipeline. For example,**
  - **eliminating WAR by always fetching operands early in the pipeline**
    - **in ID stage**
  - **eliminating WAW by doing all WBs in order**
    - **single write stage**
- **Detect and resolve remaining ones (RAW)**
  - **stall**
  - **forward (if possible)**

Data hazards in our pipeline

- **Happen when there are RAW data dependencies between instructions executing in the pipeline**
  - **Read the old (wrong) data**
  - **See example in the next slide**

Data hazard example



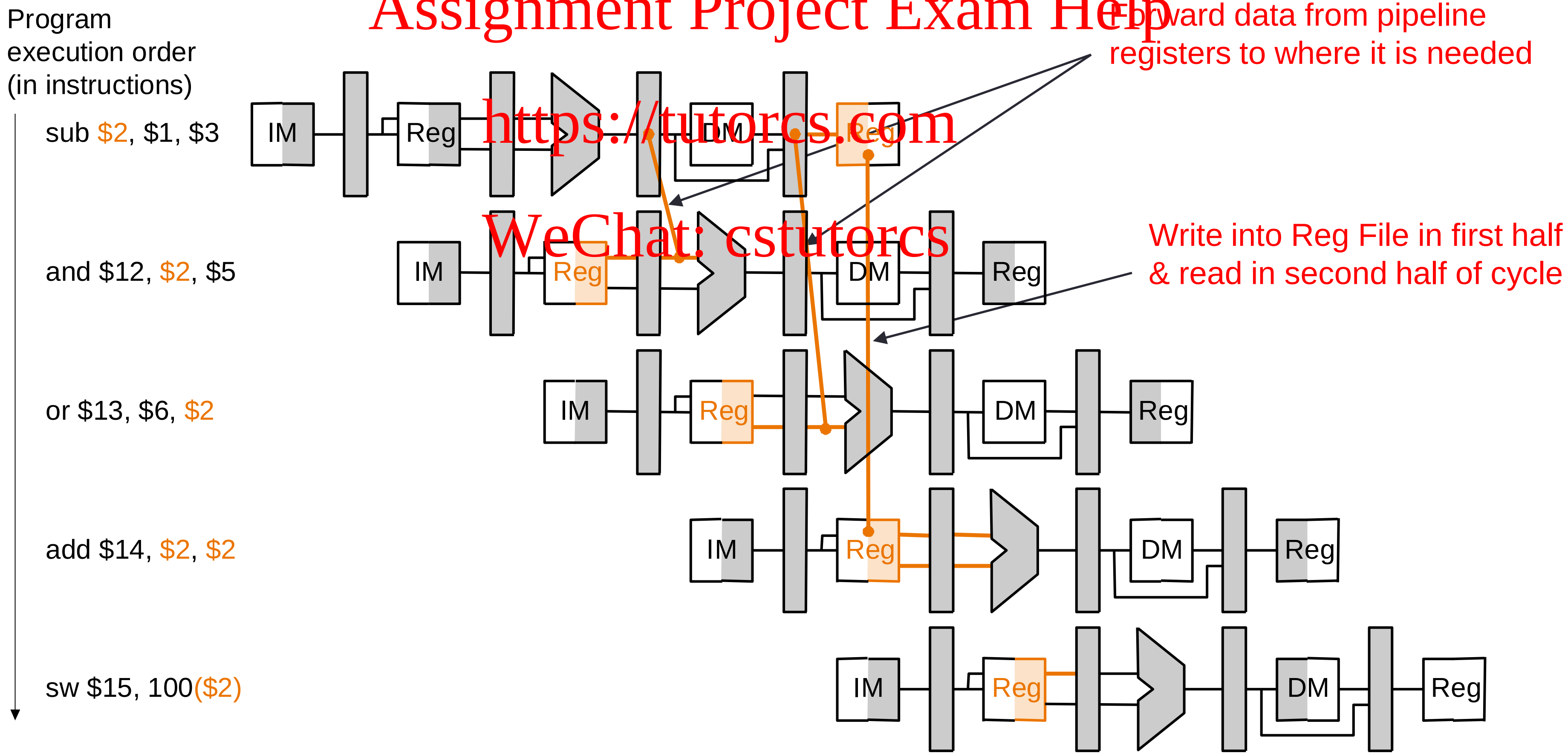
Handling data hazard

- **Stalling pipeline**
  - **A basic solution**
    - **If the correct data is not available in the storage location, stall the instruction execution in the pipeline.**
- **Forwarding**
  - **A very effective approach**
    - **If the correct data is available in the pipeline, forward the data to where it is required in the pipeline.**
  - **See example in the next slide**

Forwarding

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

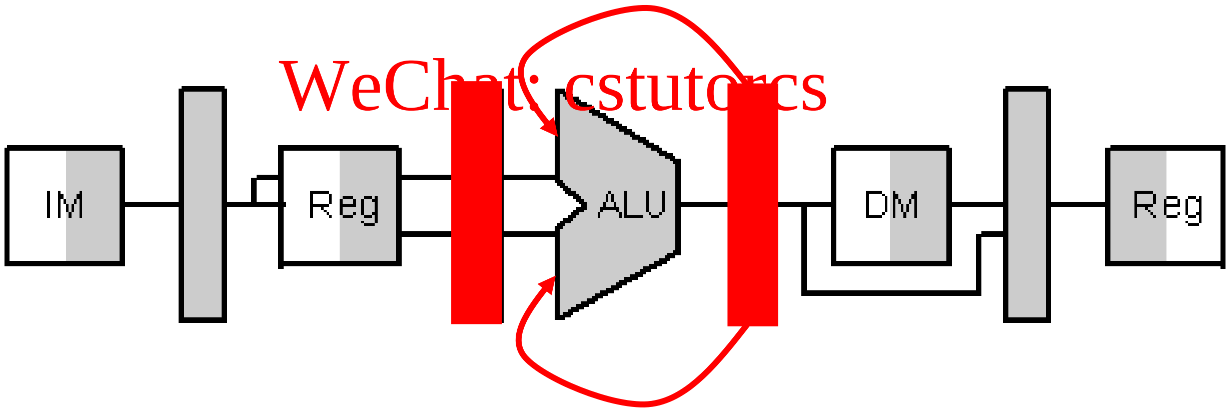

Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM
Forward data from pipeline registers to where it is needed
Write into Reg File in first half & read in second half of cycle

Forwarding implementation

- **Detect data hazards**
  - **Check for data dependencies between each instruction and its preceding instructions in the pipeline**
- **Forward the correct data to where it is required. For example**
  - **Ex stage**

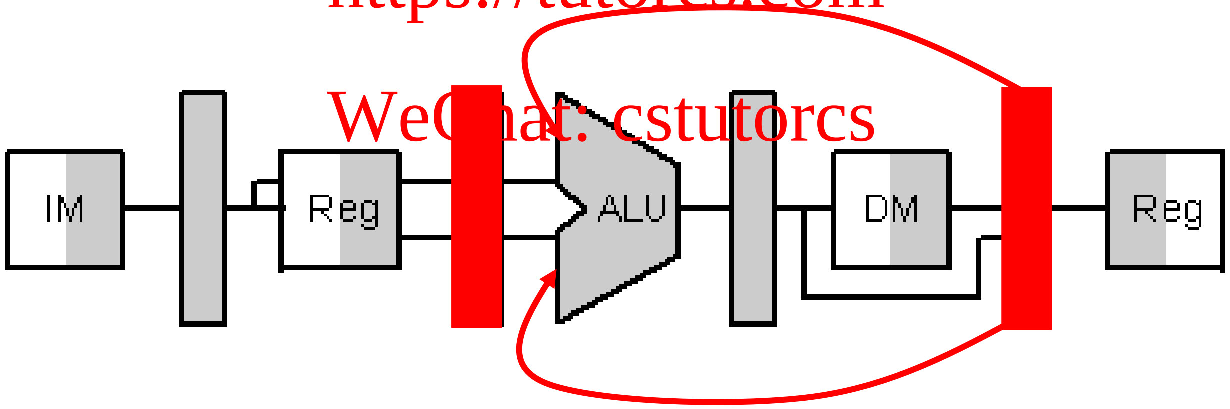
Detect data hazard for Ex stage (1)

- Based on the pipeline register field contents
  - Source register dependent on destination register of its first preceding instruction
    - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
    - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt



Detect data hazard for Ex stage (2)

- **Based on the pipeline register field contents**
  - **Source register dependent on destination register of the second preceding instruction**
    - **2a. MEM/WB.RegisterRd = ID/EX.RegisterRs**
    - **2b. MEM/WB.RegisterRd = ID/EX.RegisterRt**



Example

sub	$2, $1, $3	**IF**	**ID**	**EX**	**ME**	**WB**				
and	$12, $2, $5		**IF**	**ID**	**EX**	**ME**	**WB**			
or	$13, $6, $2			**IF**	**ID**	**EX**	**ME**	**WB**		
add	$14, $2, $2				**IF**	**ID**	**EX**	**ME**	**WB**	
sw	$15, 100($2)					**IF**	**ID**	**EX**	**ME**	**WB**

Dependent instructions: sub-and

Data hazard type: 1a

Example

```
sub    $2, $1, $3      IF  ID  EX  ME| WB
and    $12, $2, $5         IF  ID  EX  ME  WB
or     $13, $6, $2             IF  ID| EX  ME  WB
add    $14, $2, $2                 IF  ID  EX  ME  WB
sw     $15, 100($2)                    IF  ID  EX  ME  WB
```

Dependent instructions: sub-and, sub-or

Data hazard type: 1a, 2b

Detect data hazard for Ex stage (3)

- More considerations: preceding instructions must
  1. Write back a result
  2. If they write to $0 (or $zero), the result is never stored, and hence needs not to be forwarded
     - Recall: $0 is a special register in MIPS that always has value 0

Forwarding for Ex hazard (1) (cont.)

- **Data forwarded from Mem stage:**

```
IF     EX/MEM.RegWrite
AND    EX/MEM.RegisterRd != 0
AND    EX/MEM.RegisterRd = ID/EX.RegisterRs
       forward ALU result to first ALU op

IF     EX/MEM.RegWrite
AND    EX/MEM.RegisterRd != 0
AND    EX/MEM.RegisterRd = ID/EX.RegisterRt
       forward ALU result to second ALU op
```

Forwarding for EX hazard (2)

- Data forwarded from WB stage:

```
IF      MEM/WB.RegWrite
AND     MEM/WB.RegisterRd != 0
AND     MEM/WB.RegisterRd = ID/EX.RegisterRs
        forward MEM data to first ALU op

IF      MEM/WB.RegWrite
AND     MEM/WB.RegisterRd != 0
AND     MEM/WB.RegisterRd = ID/EX.RegisterRt
        forward MEM data to second ALU op
```

Multiple data dependencies

- **What if a data is dependent on both the Mem and WB data?**

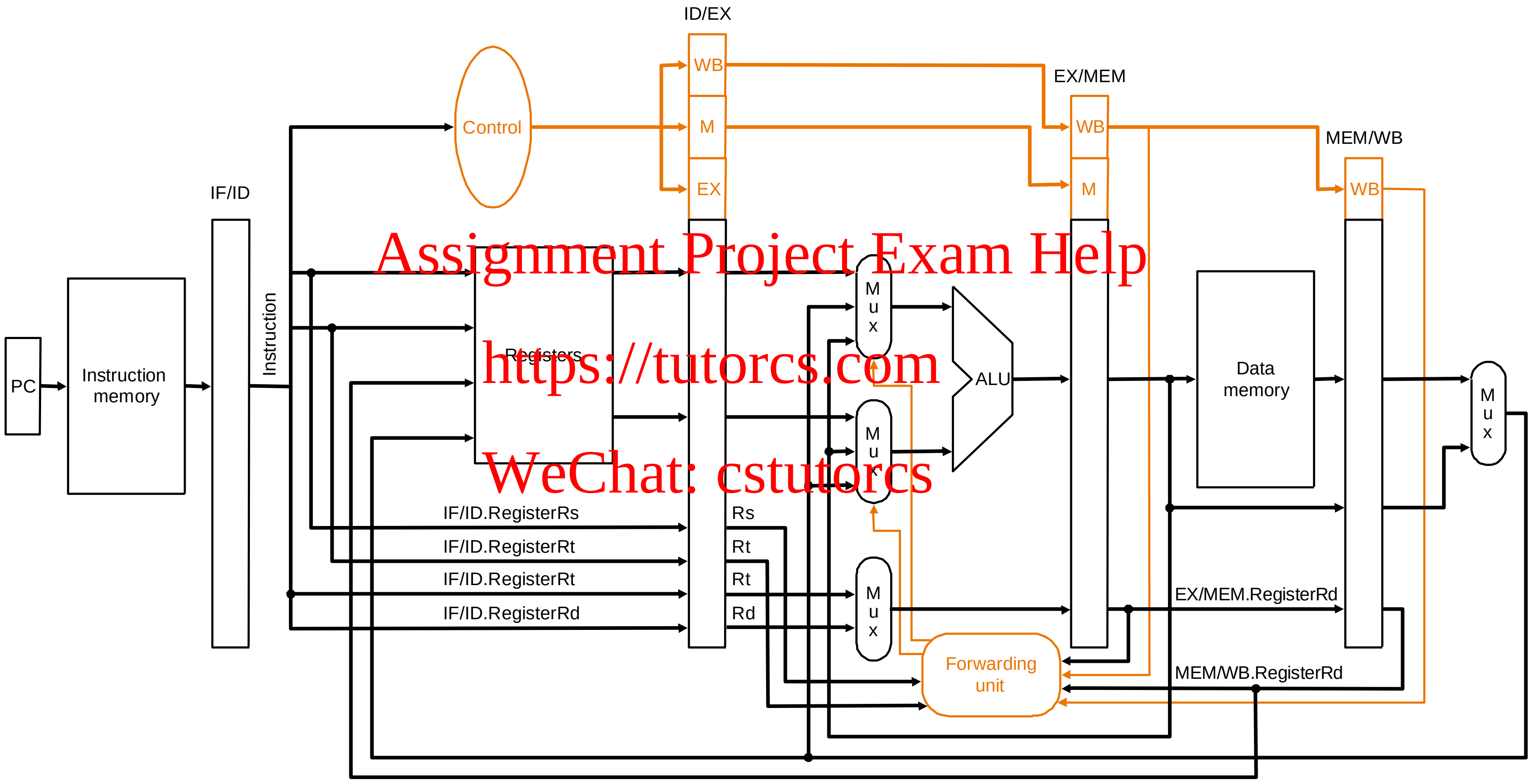
```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
```

- **Forward the more recent results – the data from Mem**

```
IF      MEM/WB.RegWrite
AND     MEM/WB.RegisterRd != 0
AND     EX/MEM.RegisterRd != ID/EX.RegisterRs
AND     MEM/WB.RegisterRd = ID/EX.RegisterRs
        forward WB data to first ALU op

IF      MEM/WB.RegWrite
AND     MEM/WB.RegisterRd != 0
AND     EX/MEM.RegisterRd != ID/EX.RegisterRt
AND     MEM/WB.RegisterRd = ID/EX.RegisterRt
        forward WB data to second  ALU op
```

Modified datapath with forwarding unit



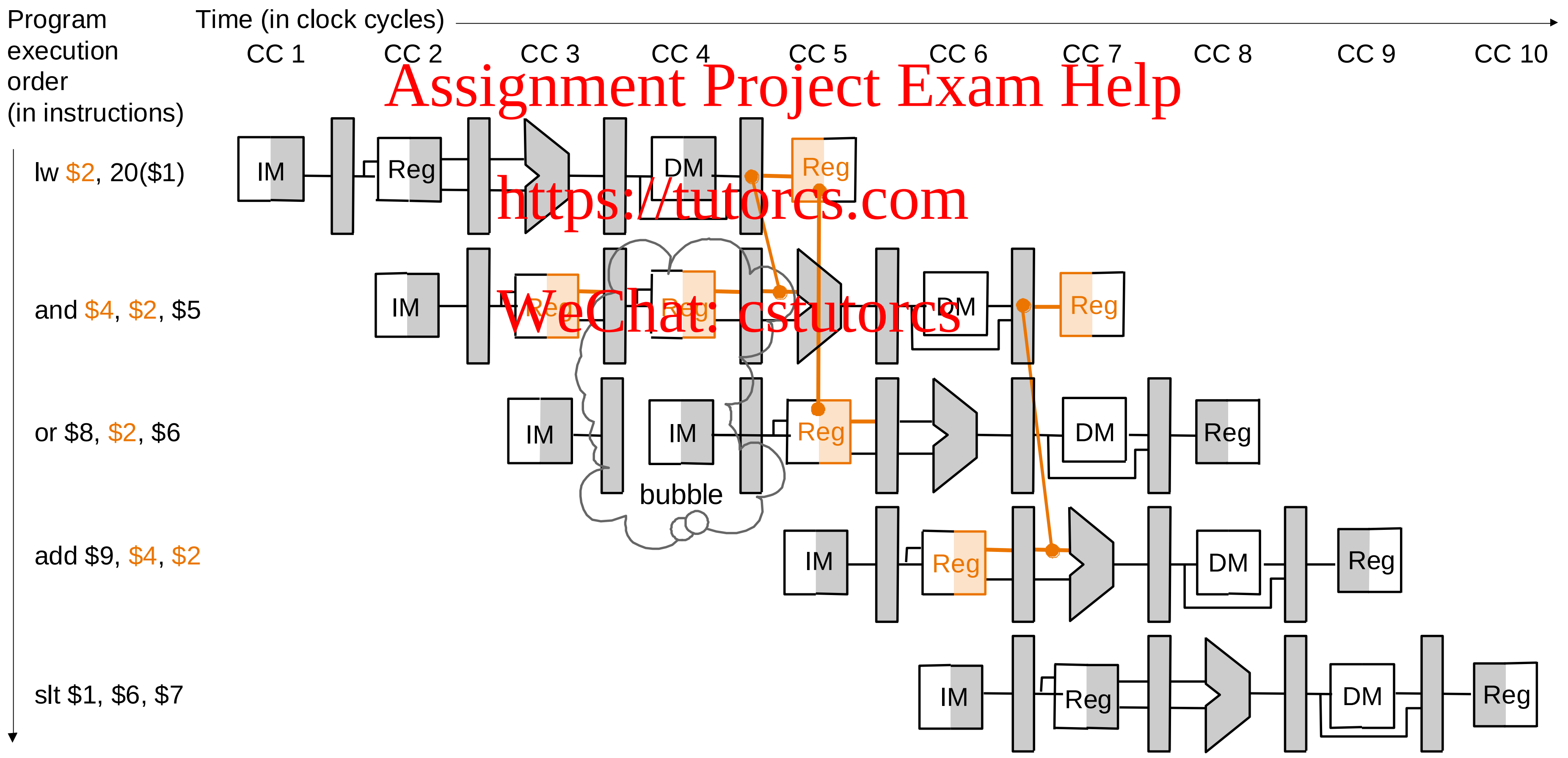
In-class exercise (4)

Identify all of the data dependencies in the following code. Which dependencies are data hazards that can be resolved by forwarding?

```
add   $2,   $5,   $4
add   $4,   $2,   $5
sw    $5,   100($2)
add   $3,   $2,   $4
```

When forwarding does not work → STALL

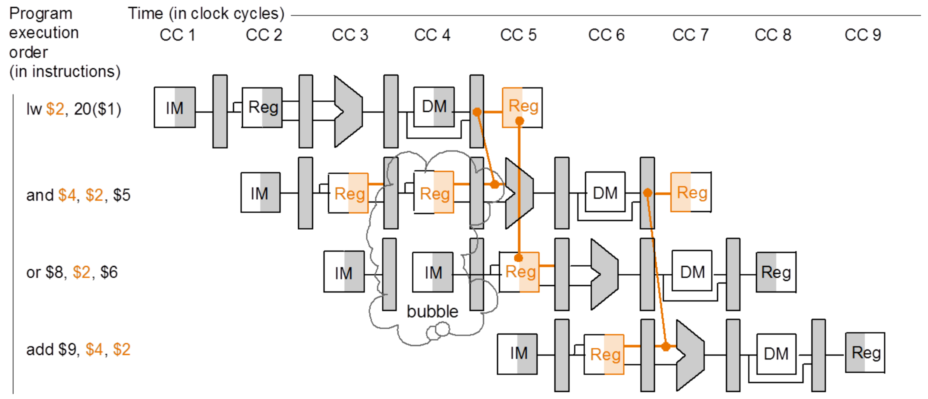
- **Detect such a hazard**
- **Stall the pipeline (insert bubble)**



Detect such a hazard

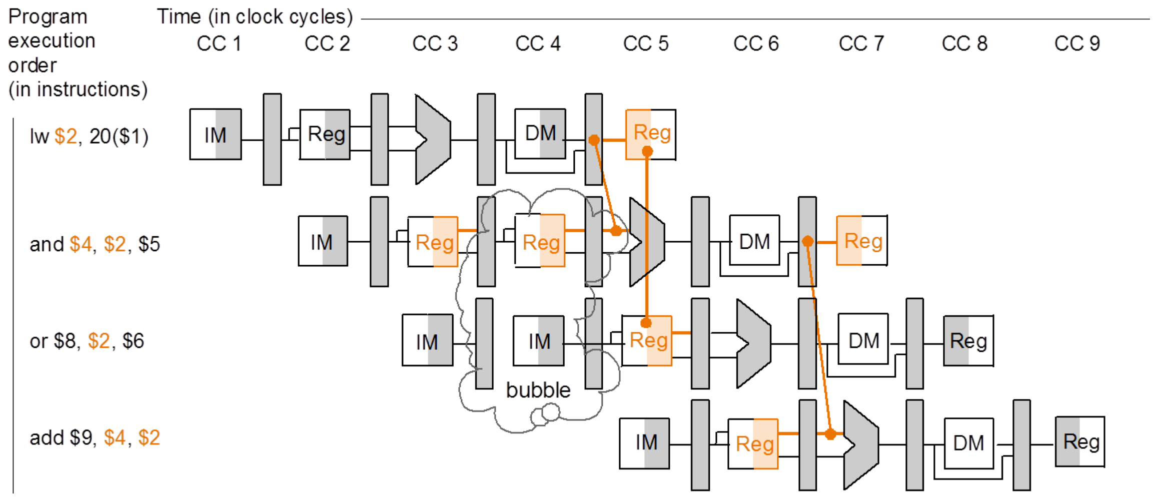
- **Such a hazard happens when a load instruction is followed by an instruction dependent on the memory data (load-use hazard, or *LUH*)**
  - **A hazard detection unit is needed to check the following condition:**

IF	ID/EX.MemRead
AND	(ID/EX.RegisterRt = IF/ID.RegisterRs
OR	ID/EX.RegisterRt = IF/ID.RegisterRt)
	stall the pipeline

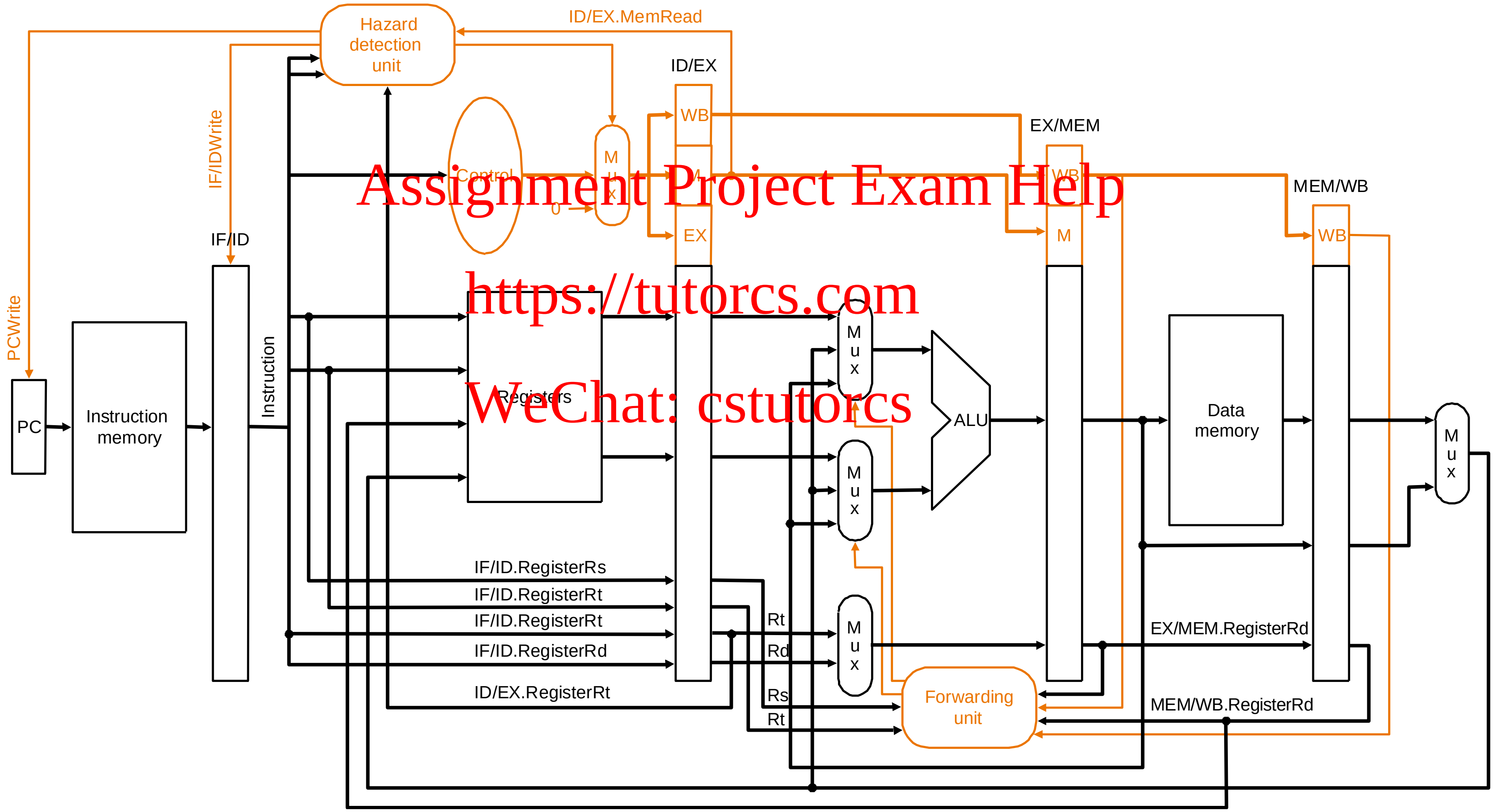


Stall pipeline

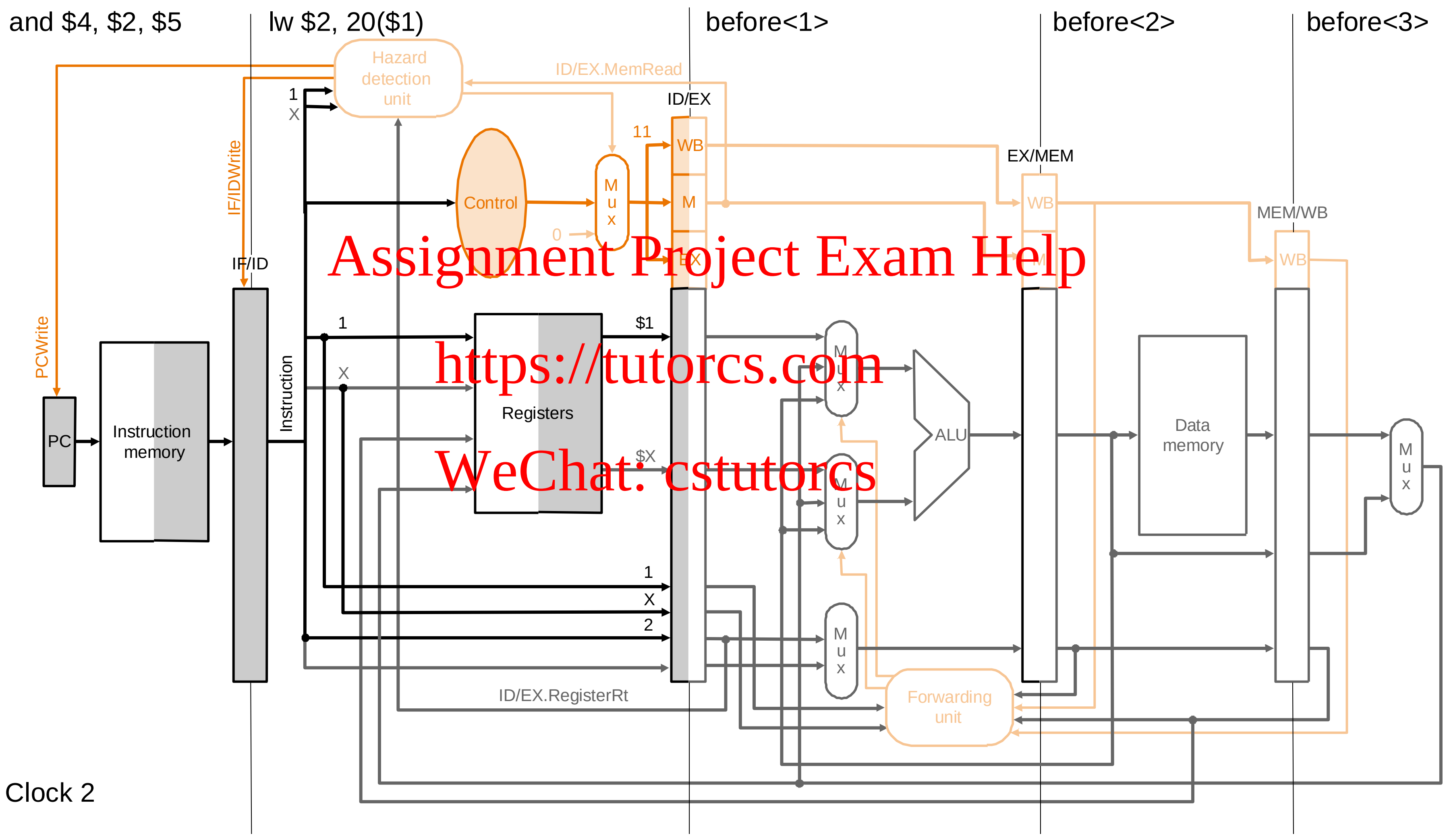
- Prevent the PC and IF/ID registers from changing
  - by not loading new values
- Set the EX, MEM and WB control fields of the ID/EX register to 0 to perform a NO-OP
  - since no state updates will occur
- See next slide



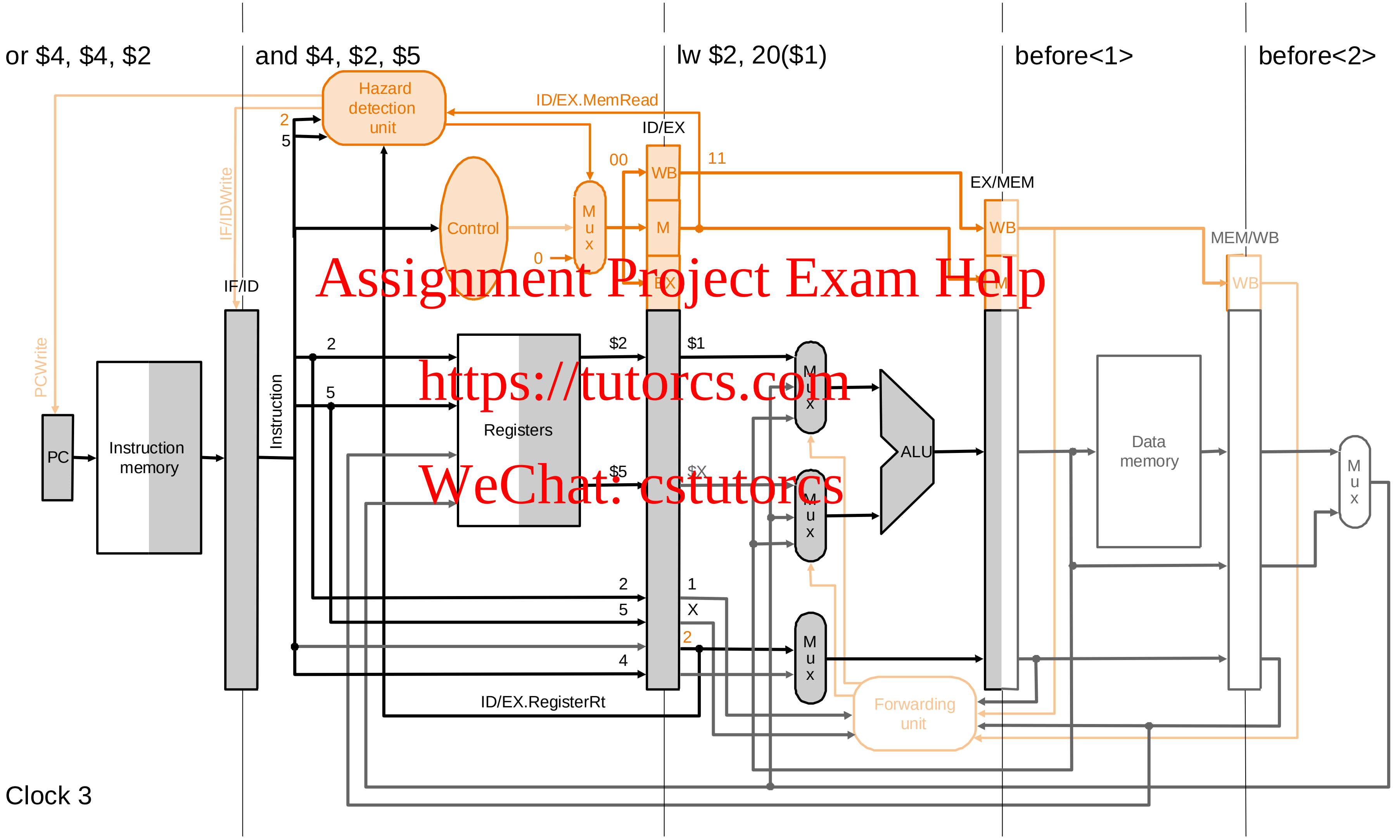
Stall pipeline (cont.)



Execution example of LUH

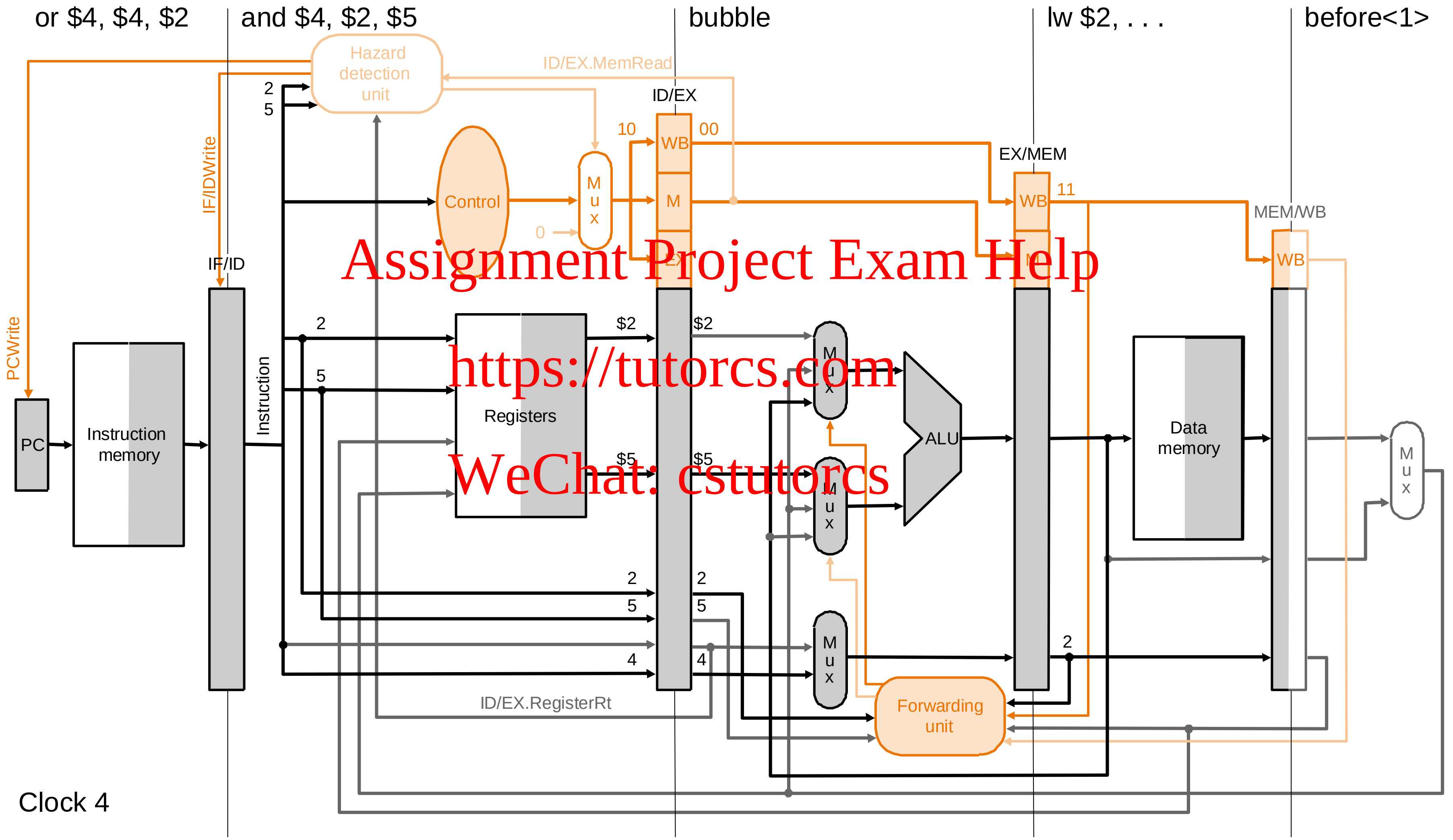


Execution example of LUH (cont.)

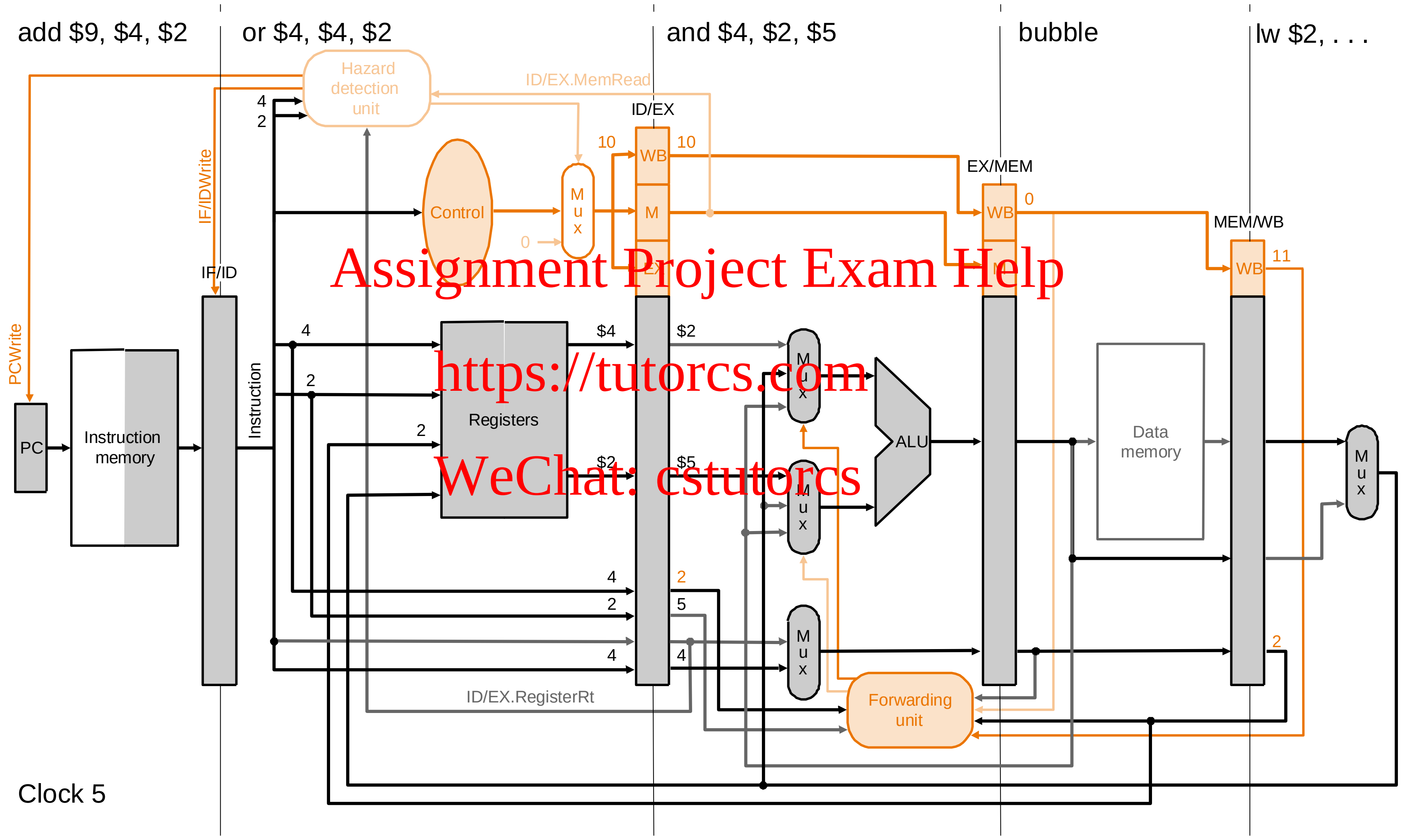


Clock 3

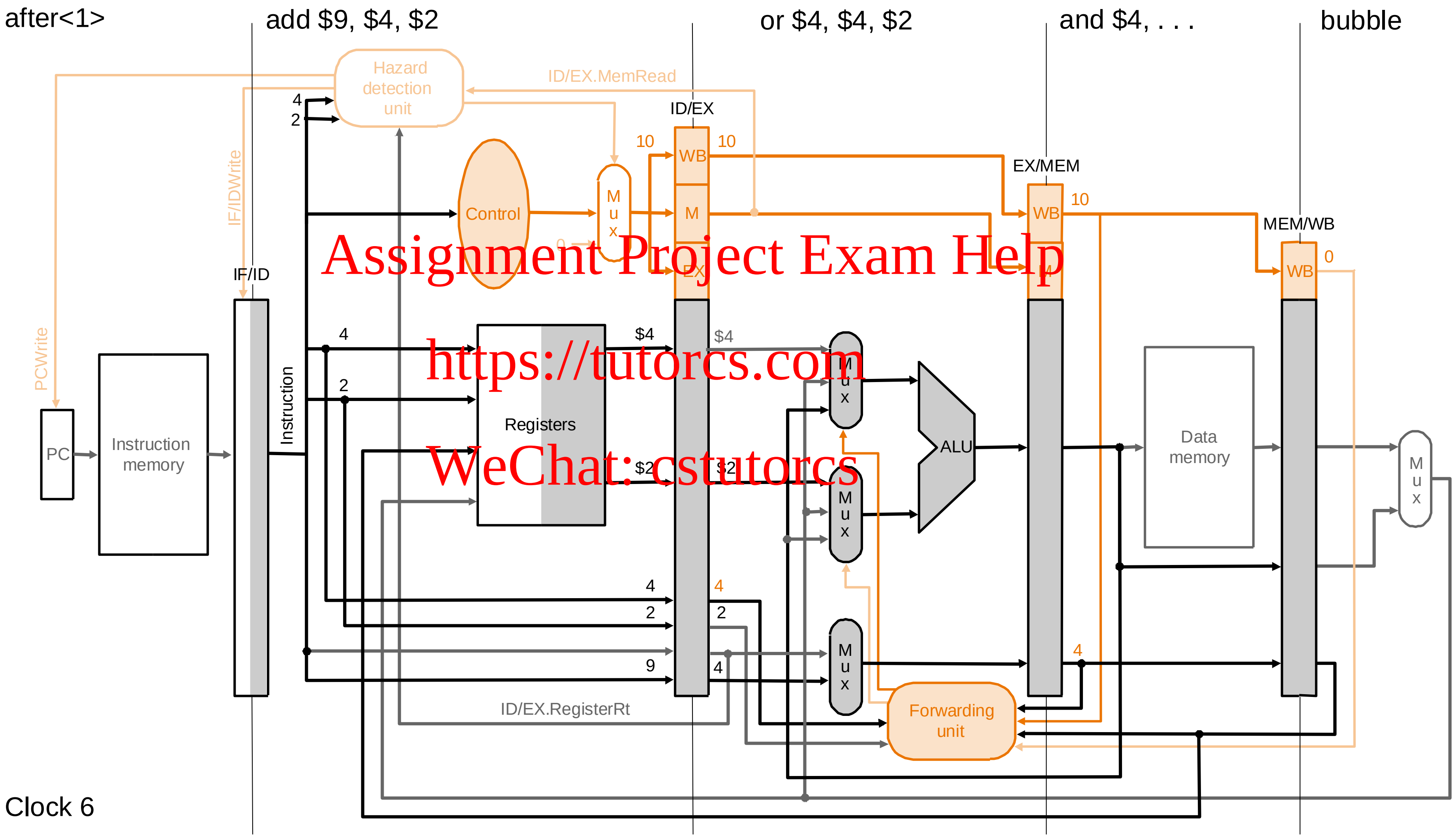
Execution example of LUH (cont.)



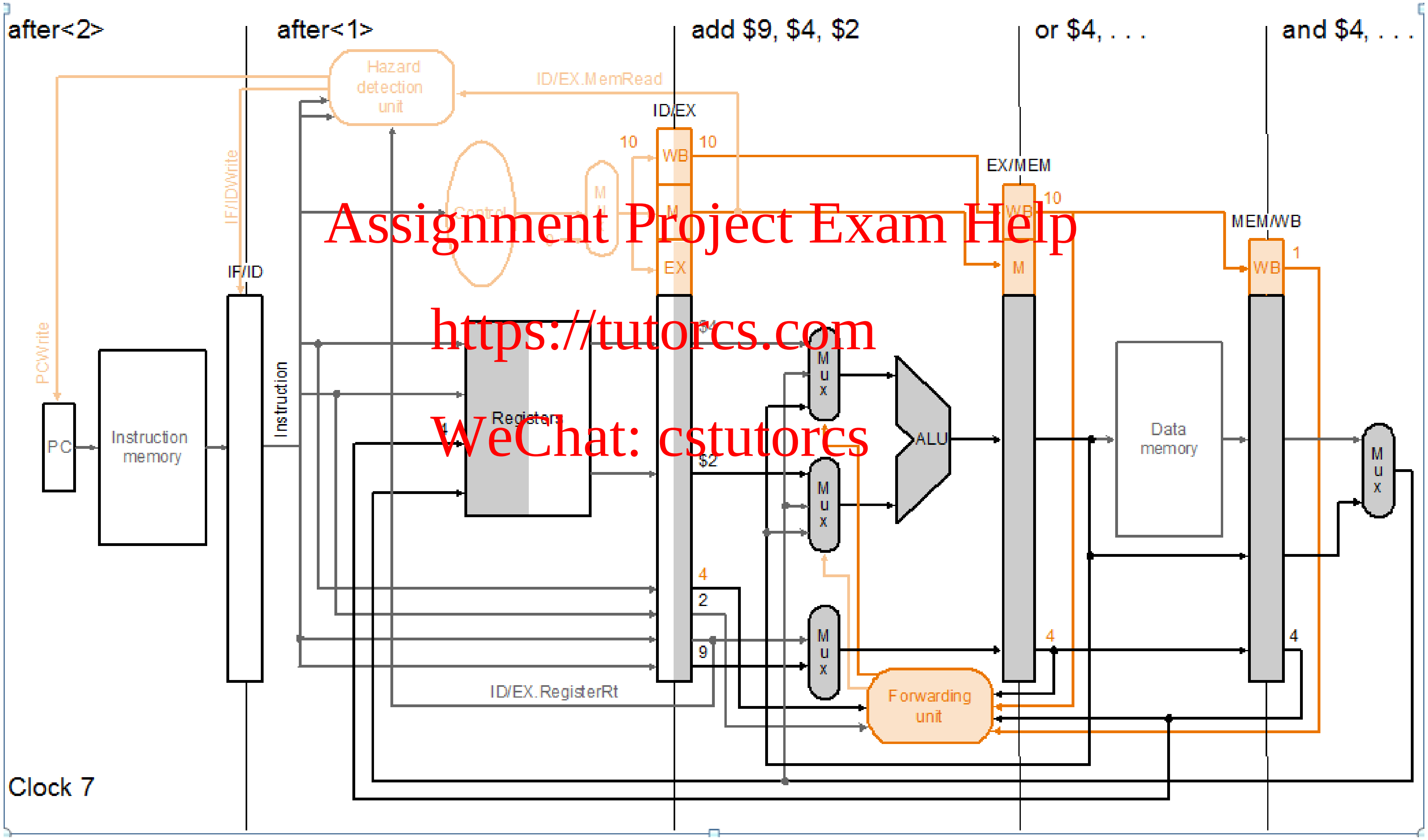
Execution example of LUH (cont.)



Execution example of LUH (cont.)



Execution example of LUH (cont.)


after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/ID
IF/IDWrite
PCWrite
Control
WB
M
EX
10
1
PC
Instruction memory
Instruction
Registers
ALU
Data memory
M u x
$4
$2
4
2
9
ID/EX.RegisterRt
Forwarding unit
Clock 7