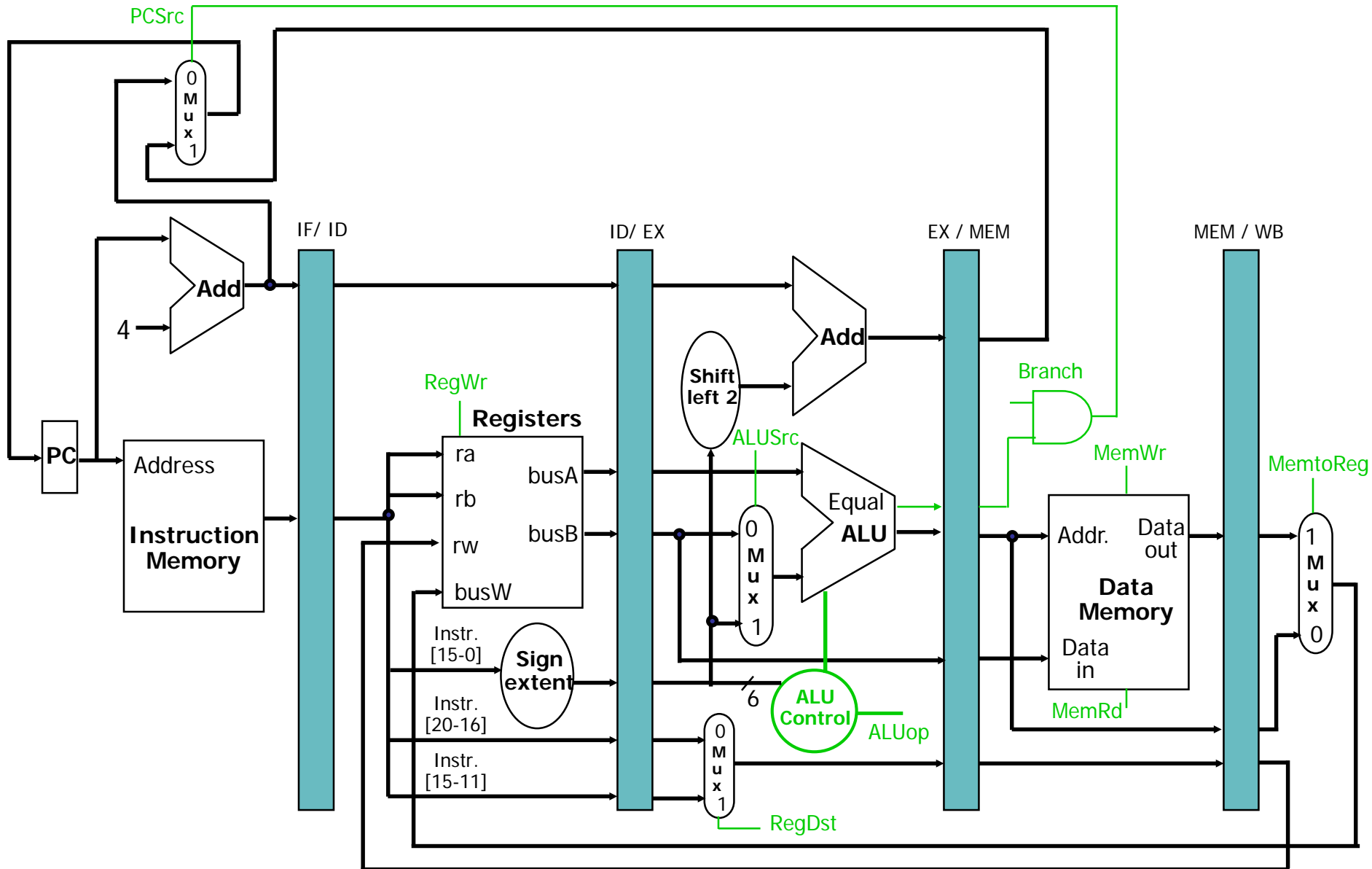


CSE331 – Computer Organization

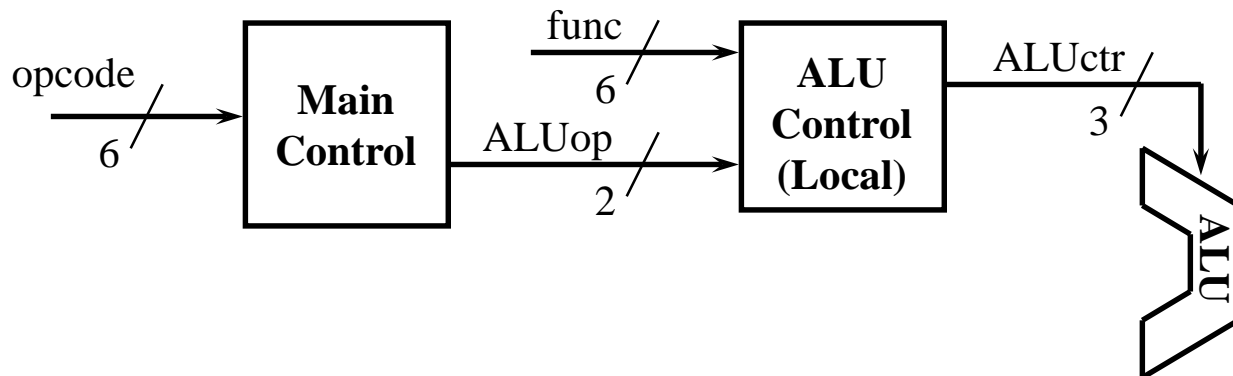
Lecture 11: A Pipelined Datapath Control

Pipelined Datapath with Control Signals



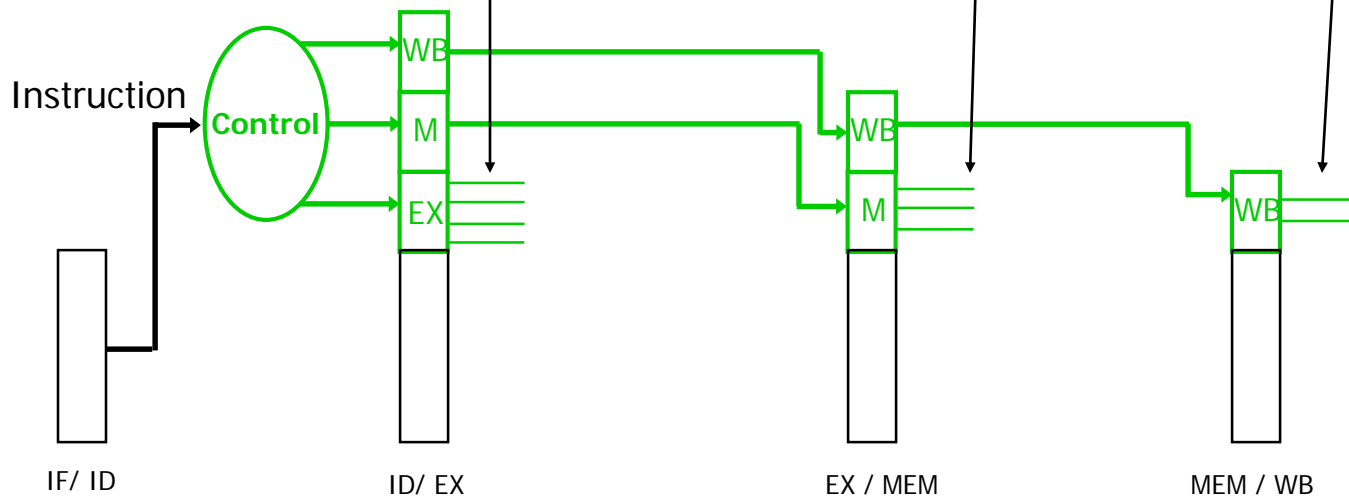
Recall: ALU Control Bits

Instruction opcode	ALUop	Instruction operation	Function field	Desired ALU action	ALU control
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
beq	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less than	101010	set on less than	111

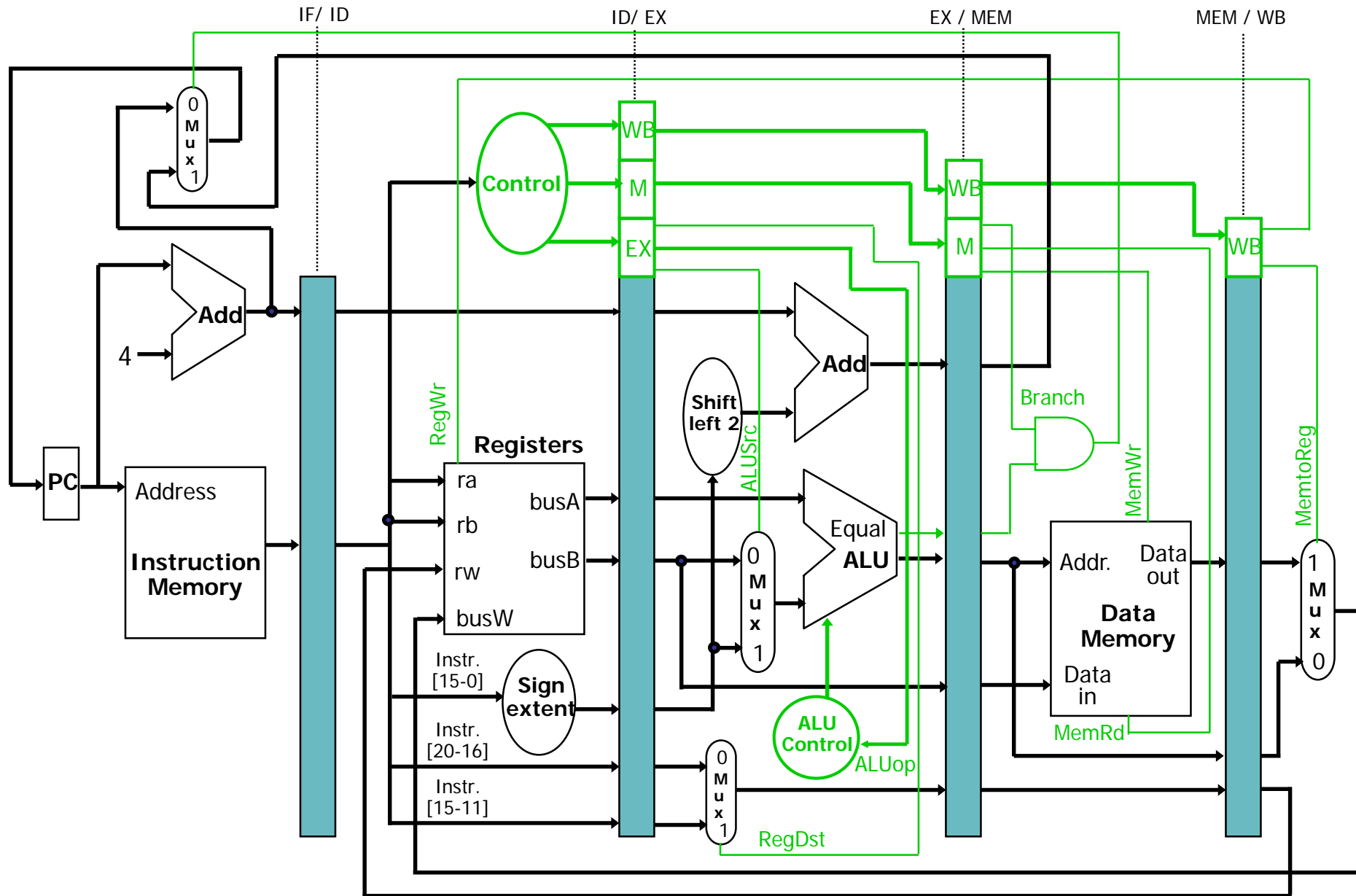


The Values of Control Lines for The Last Three Pipeline Stages

Instructions	Execution / Address Calculation stage control lines				Memory Access stage control lines			Write Back stage control lines	
	Reg Dst	ALU Op0	ALU Op1	ALU Src	Branch	Mem Rd	Mem Wr	Reg Wr	Mem to Reg
R-type	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
Beq	X	0	1	0	1	0	0	0	X



The Pipelined Datapath with Control Signals



An Example to Clarify Pipelined Control

- Let's look at what's happening in the pipeline for the following program.

lw \$10, 20(\$1)

sub \$11, \$2, \$3

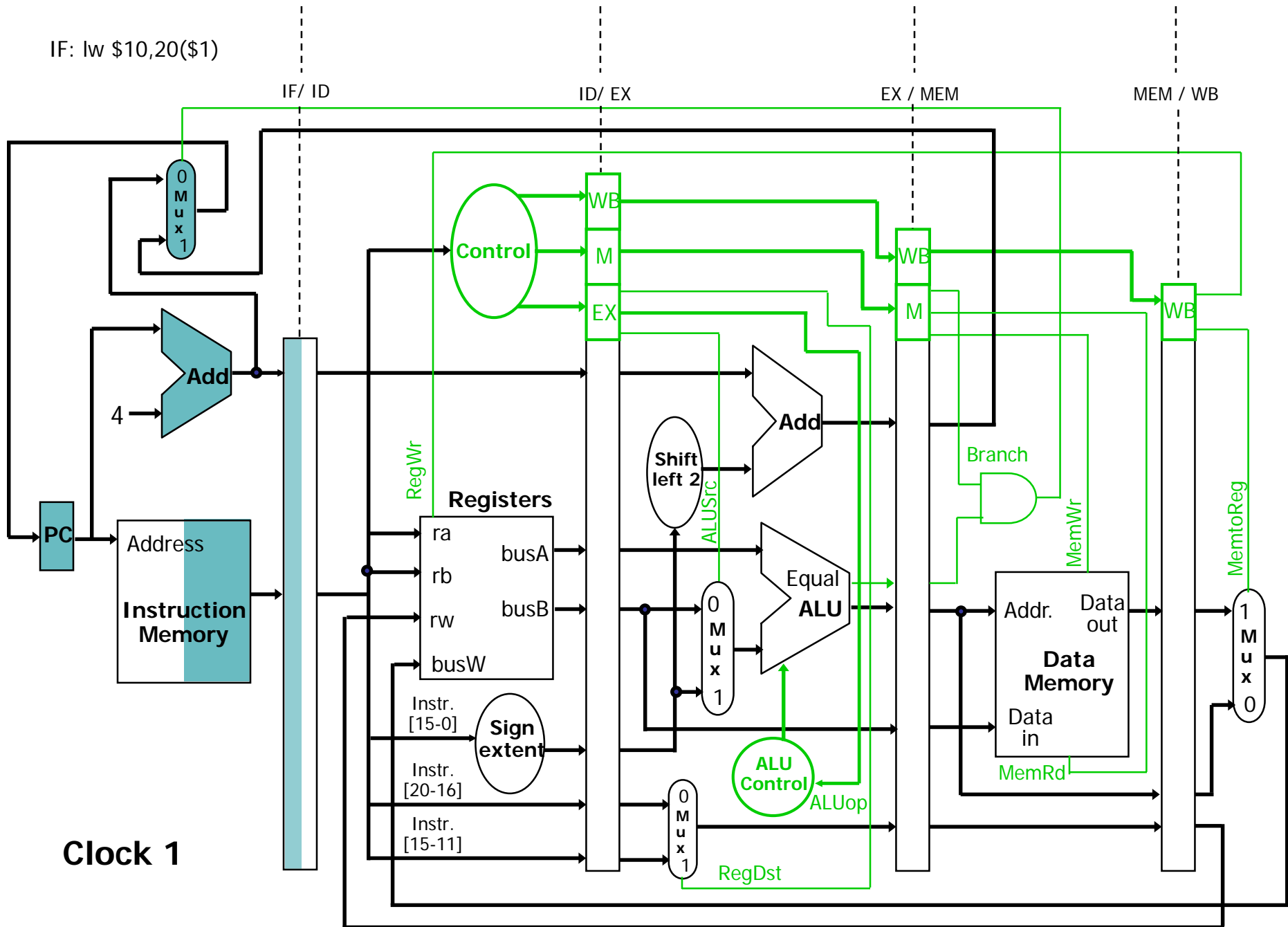
and \$12, \$4, \$5

or \$13, \$6, \$7

add \$14, \$8, \$9

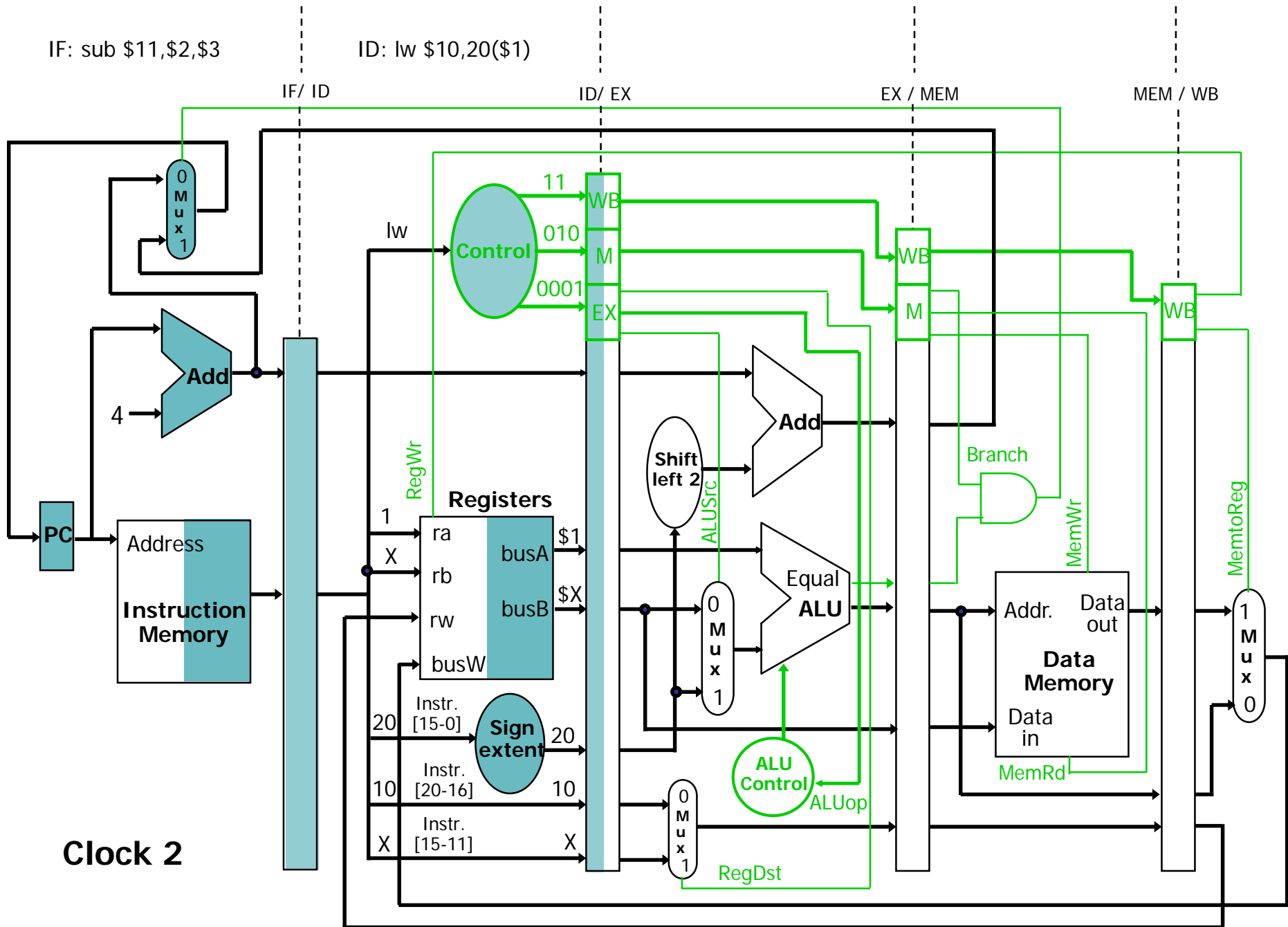
- Code does not have any data, control, or structural hazard.

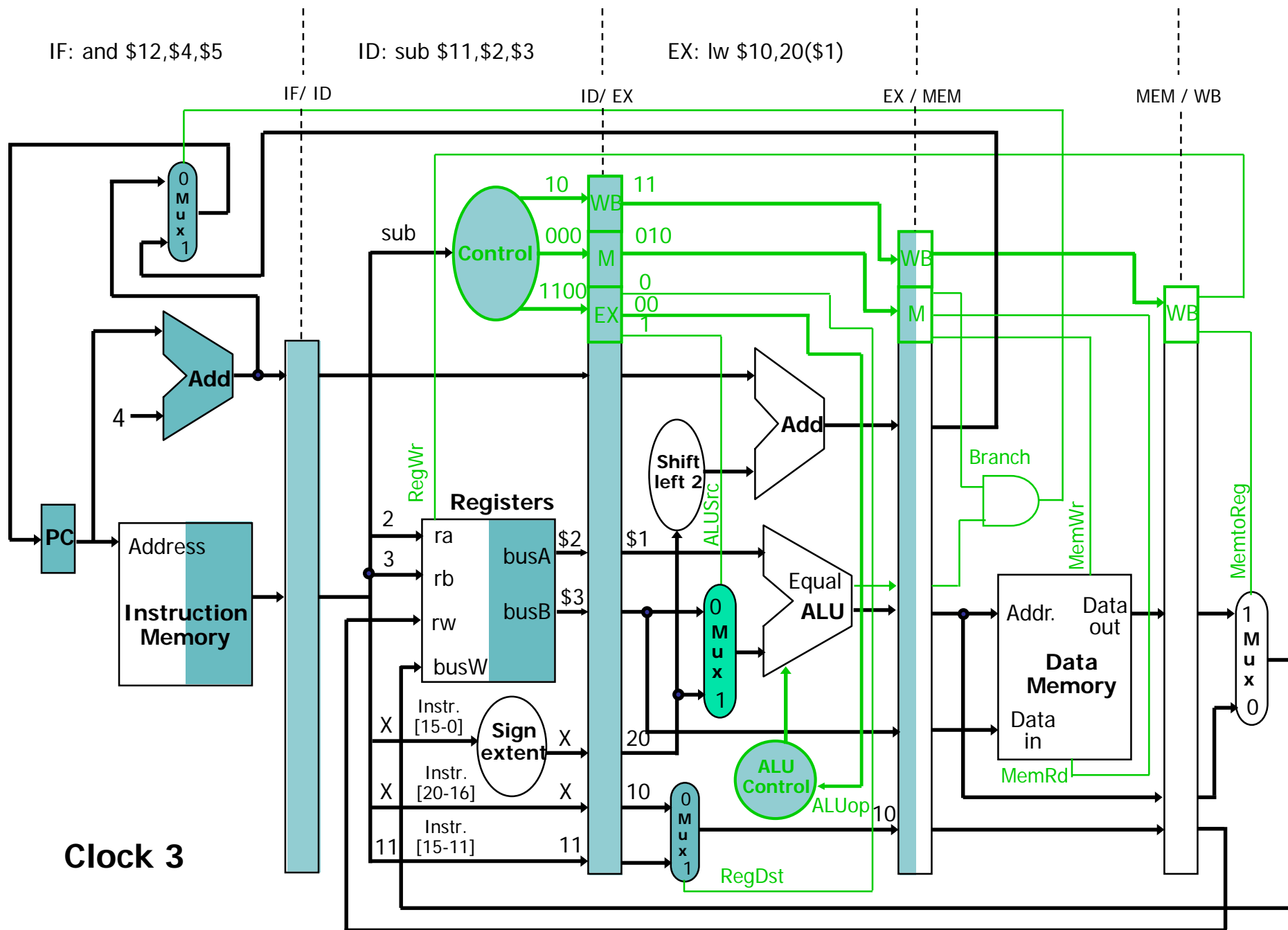
IF: lw \$t0,20(\$t1)

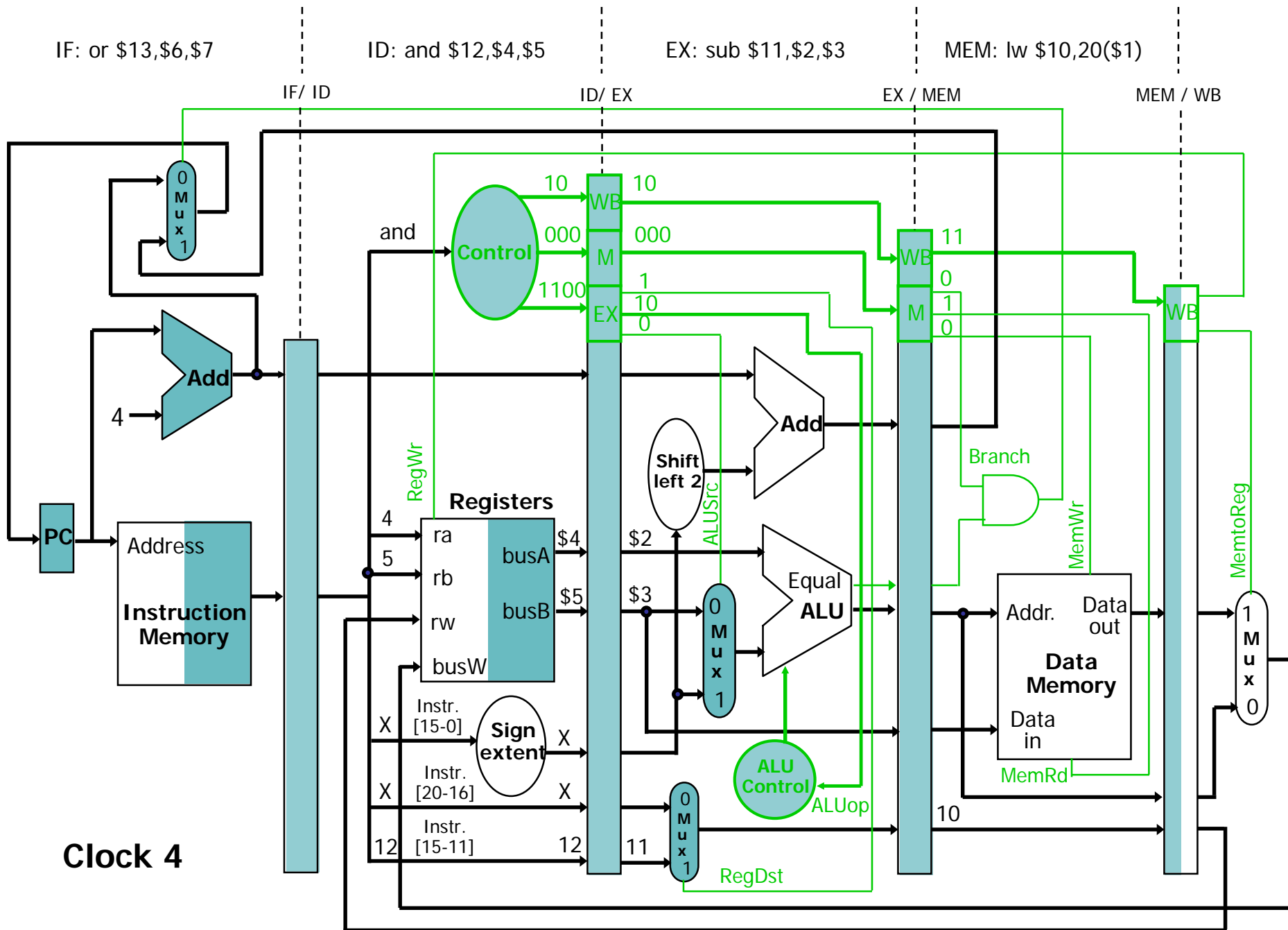


IF: sub \$11,\$2,\$3

ID: lw \$10,20(\$1)







IF: add \$14,\$8,\$9

ID: or \$13,\$6,\$7

EX: and \$12,\$4,\$5

MEM: sub \$11,\$2,\$3

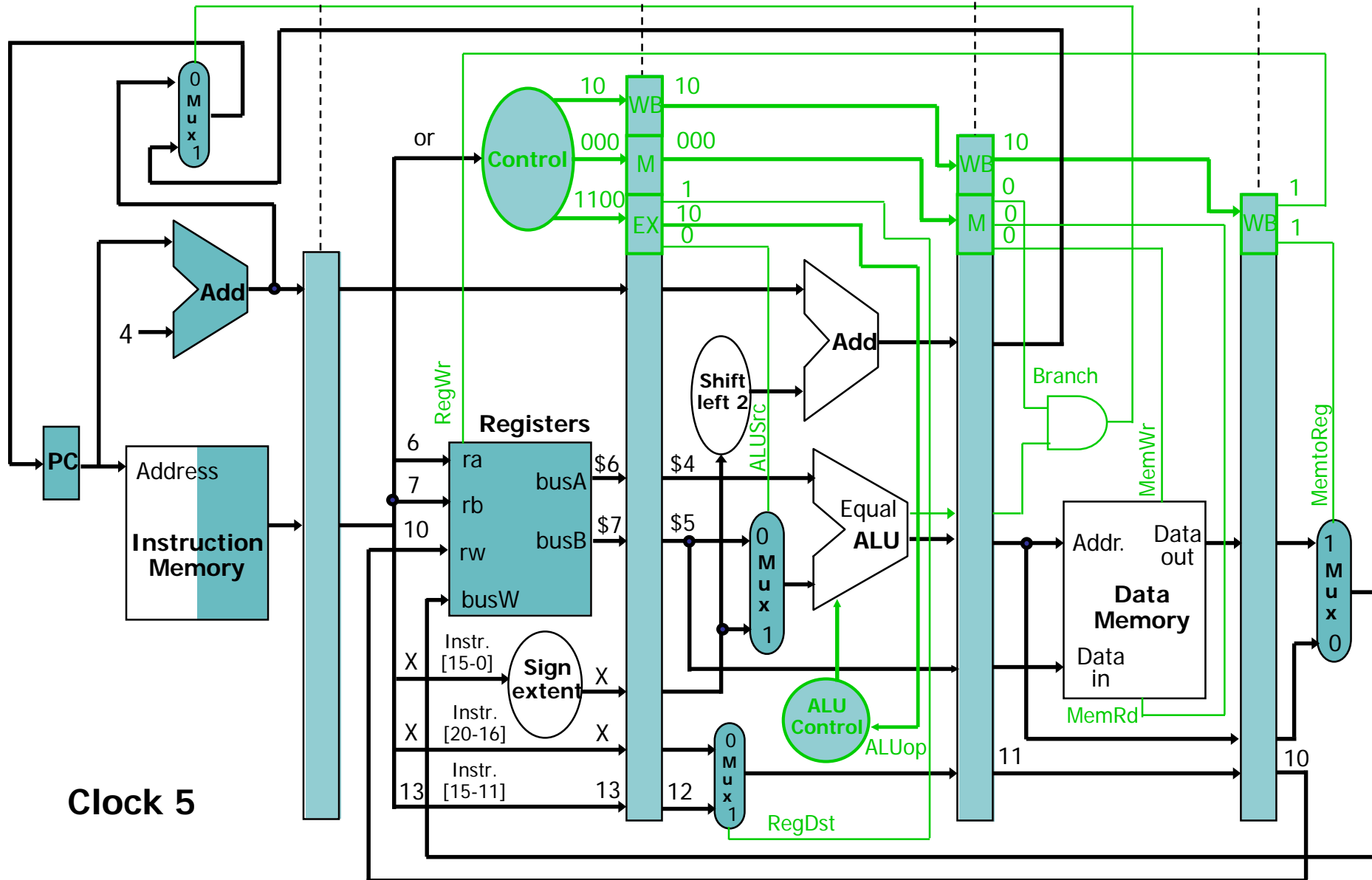
WB: lw \$10.

IF/ ID

ID/ EX

EX / MEM

MEM / WB



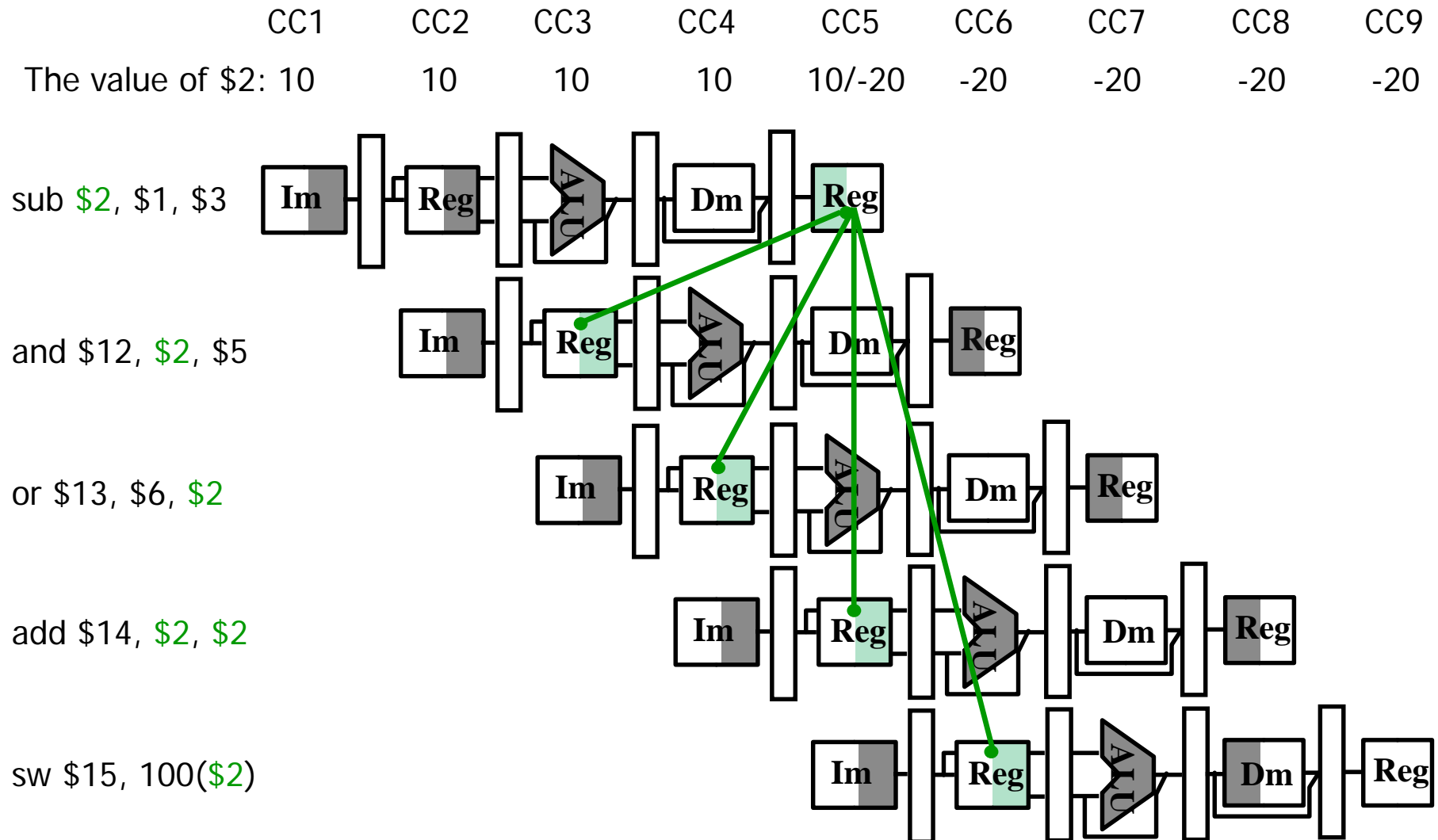
Data Hazards

- Previous example shows us how independent instructions that do not use the results calculated by prior instructions are executed.
- This is not the case with real programs.
- Let's look at the following code sequence.

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

- The last four instructions are all dependent on the register \$2 of the first instruction.
- Assume that register \$2 had the value of 10 before the subtract instruction and -20 afterwards.

Data Hazards (Cont')



Data Hazards (Cont')

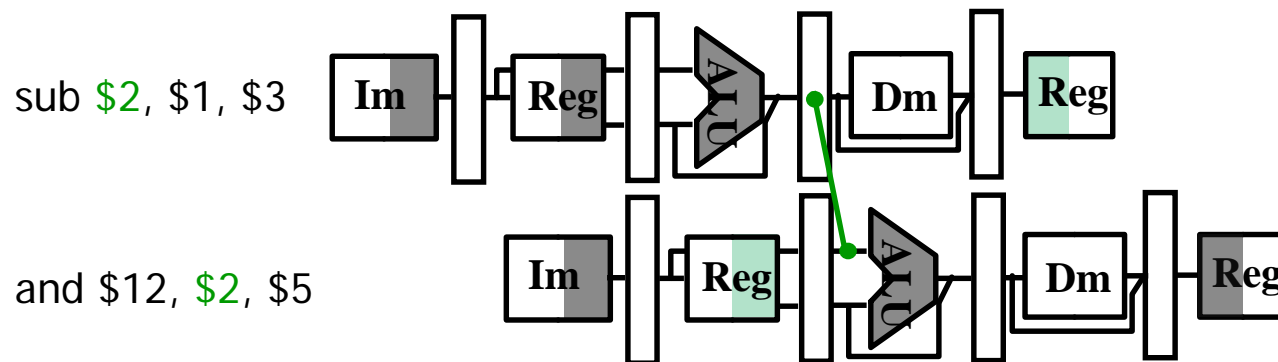
- **Simple solution:** Compiler inserts `nop` instructions between the `sub` and instructions.
 - `nop` (no operation) instruction neither modifies data nor writes a result.

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

- **Result:** It works but 2 clock cycles will be wasted.
 - ➔ Performance decrease

Data Hazard Detection & Forwarding

- It is possible to detect data hazard and then forward the proper value to resolve the hazard.
- When an instruction tries to read a register in its EX stage that an earlier instruction intends to write in its WB stage.
- This is the case between sub-and instruction below:

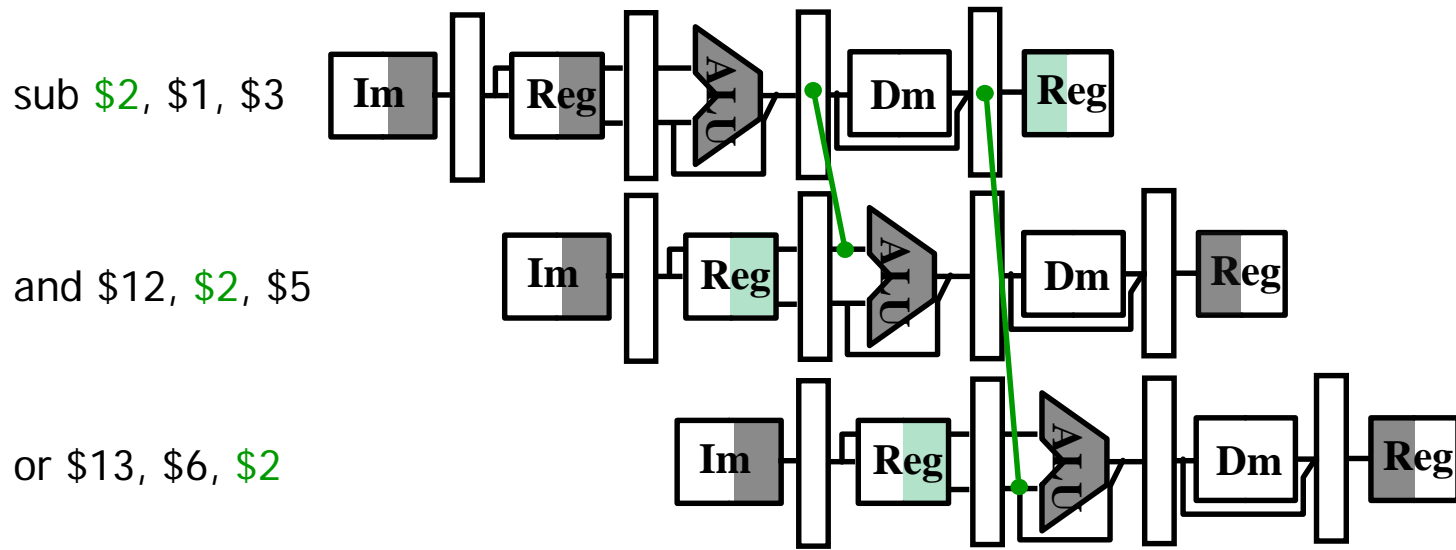


- This hazard can be detected by simply checking:

$$\text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRs} = \$2$$

Data Hazard Detection & Forwarding (Cont')

- Another hazard is between sub-or instructions:



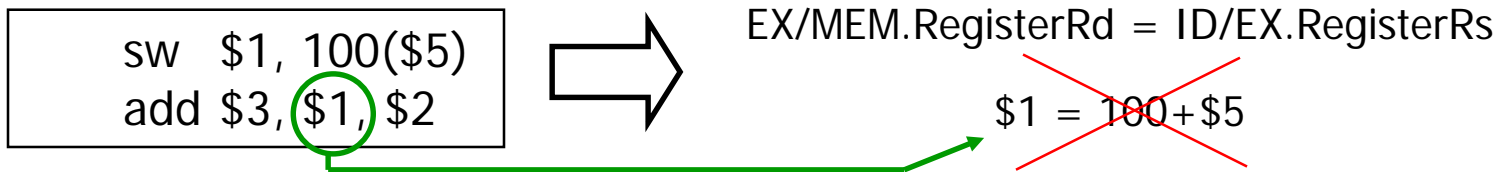
- This hazard can be detected by simply checking:
 $\text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRt} = \2
- There is no data hazard between sub-add and sub-sw instructions.

Summary of Data Hazard Conditions

- 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

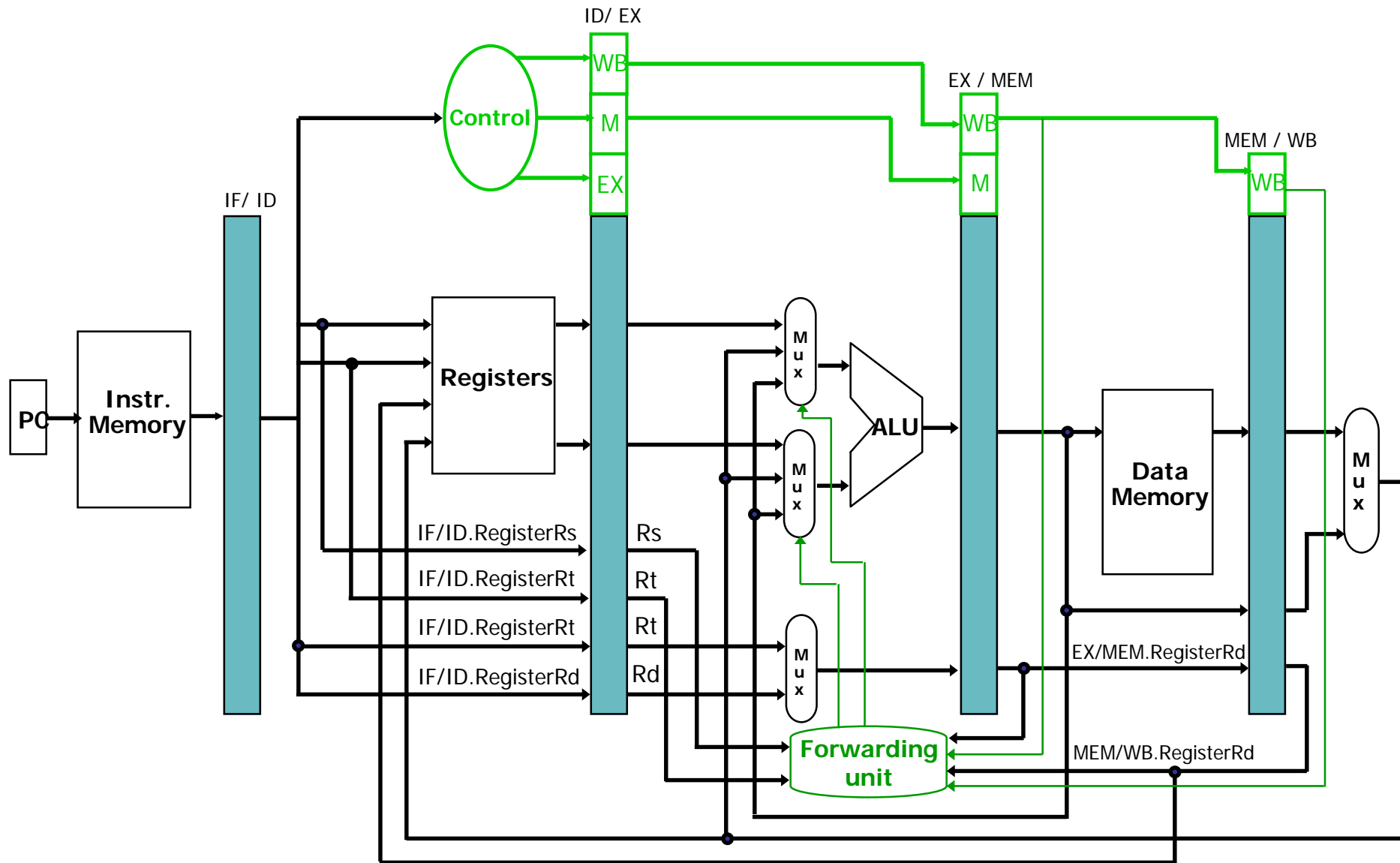
This actually refers to destination field of an instruction. It is **rd** field in **R-type** instructions and **rt** field in **I-type** instructions. Mux in the **EX** stage chooses the correct one, therefore, **EX/MEM** and **MEM/WB** pipeline registers store this information as a **rd** field (EX/MEM.Register**Rd** and MEM/WB.Register**Rd**).

- Since some of the instructions (i.e. `sw`, `beq`) do not write to register file, the above policy is inaccurate. Consider the following code sequence:



- This problem can be solved simply by checking RegWr signal.

The Pipelined Datapath with Forwarding



Summary of Data Hazard Conditions (Cont')

- Another problem: What happens if \$0 is used as a destination register?
 - A non-zero value would not be forwarded.
- Therefore, hazard detection should be the following:

EX hazard:

if (EX/MEM.RegWr
and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA=10

if (EX/MEM.RegWr
and (EX/MEM.RegisterRd \neq 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB=10

MEM hazard:

if (MEM/WB.RegWr
and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA=01

if (MEM/WB.RegWr
and (MEM/WB.RegisterRd \neq 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB=01

Summary of Data Hazard Conditions (Cont')

- Consider the following sequence:

add \$1, \$1, \$2

add \$1, \$1, \$3

add \$1, \$1, \$4

. . .

- In this case, the result is forwarded from MEM stage because the result in the MEM stage is the more recent result than the result in WB stage. Thus, the control for the MEM hazard:

MEM hazard:

if (MEM/WB.RegWr

and (MEM/WB.RegisterRd \neq 0)

and (EX/MEM.RegisterRd \neq ID/EX.RegisterRs)

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

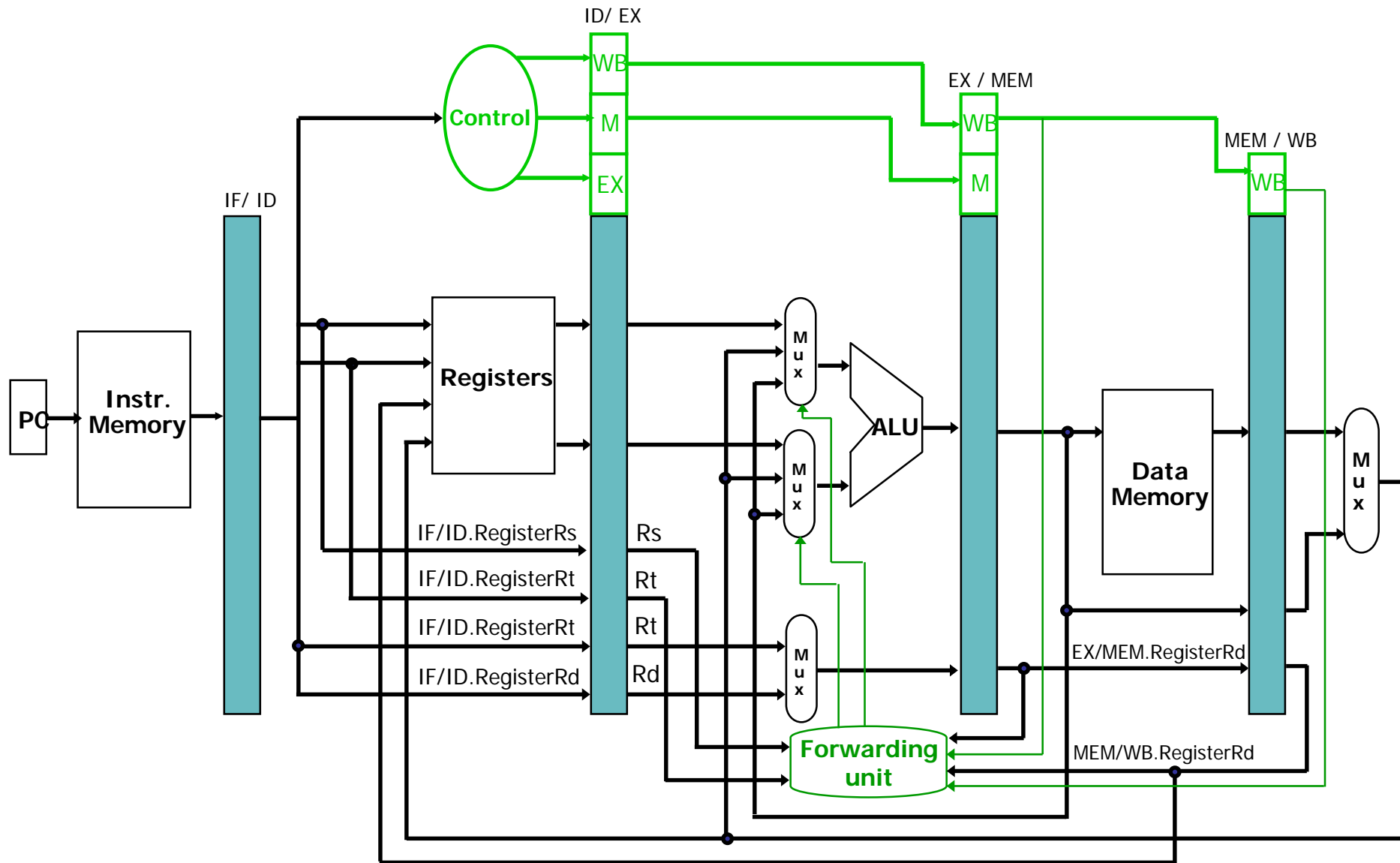
if (MEM/WB.RegWr

and (MEM/WB.RegisterRd \neq 0)

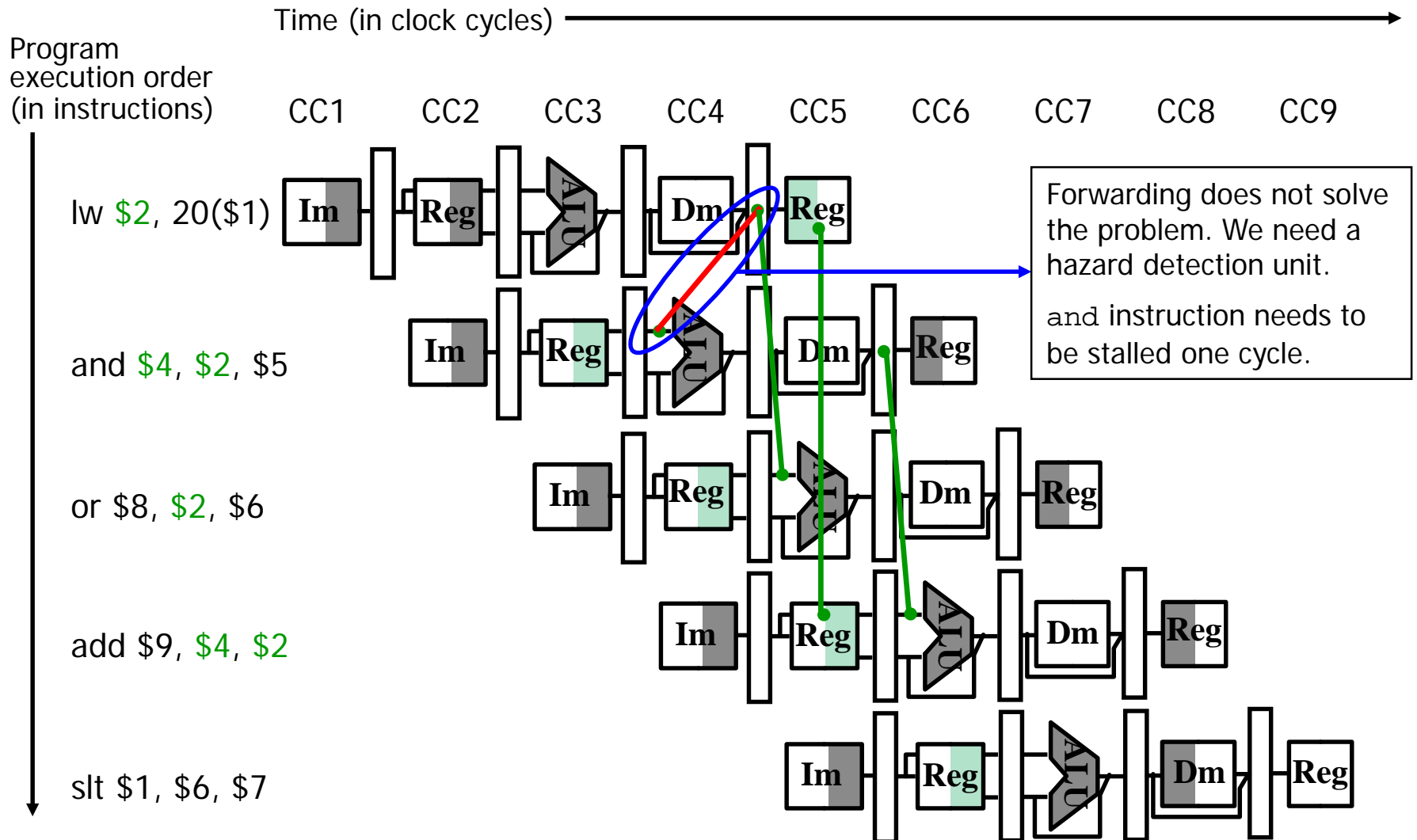
and (EX/MEM.RegisterRd \neq ID/EX.RegisterRt)

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

The Pipelined Datapath with Forwarding




Data Hazards and Stalls

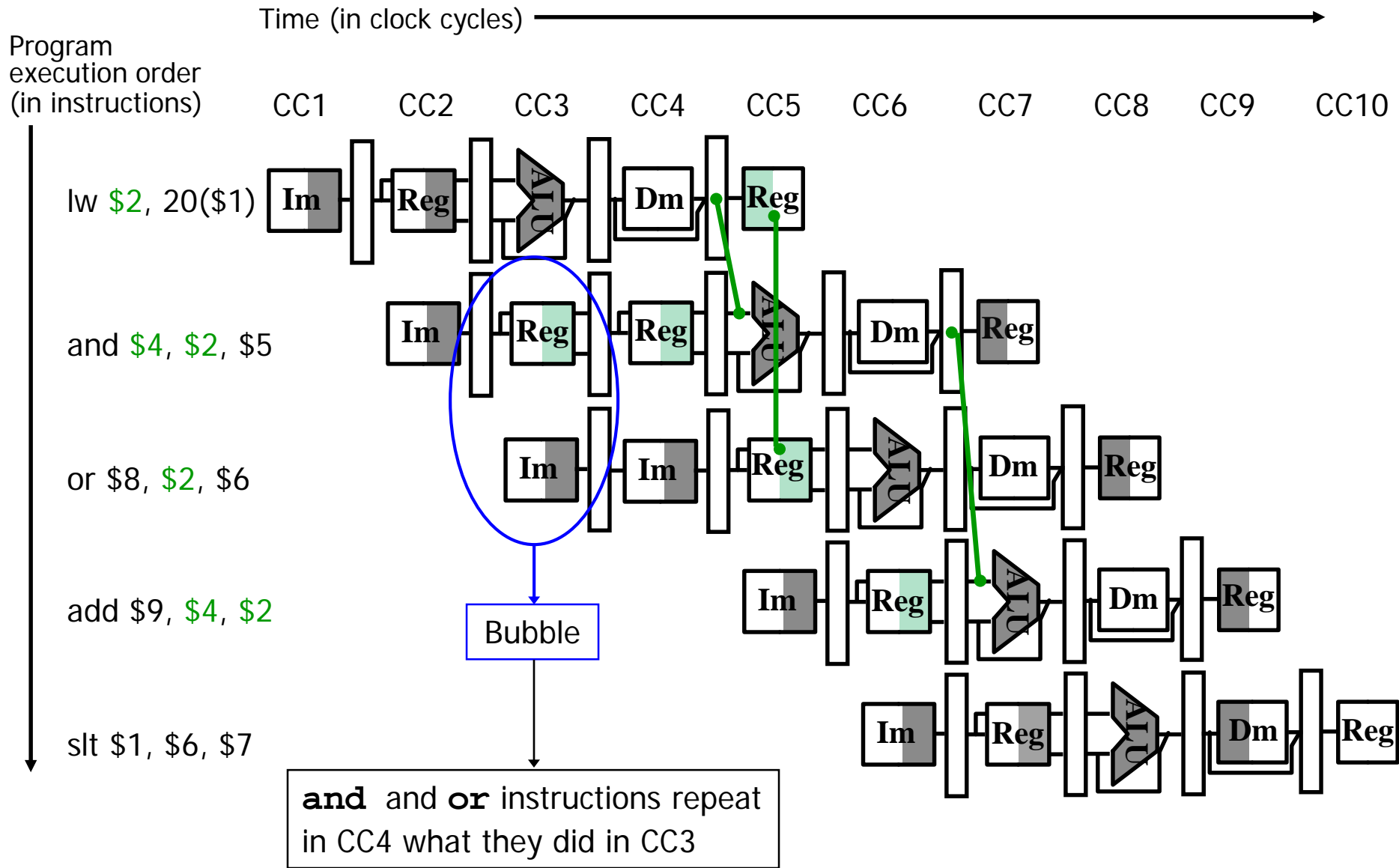


Hazard Detection

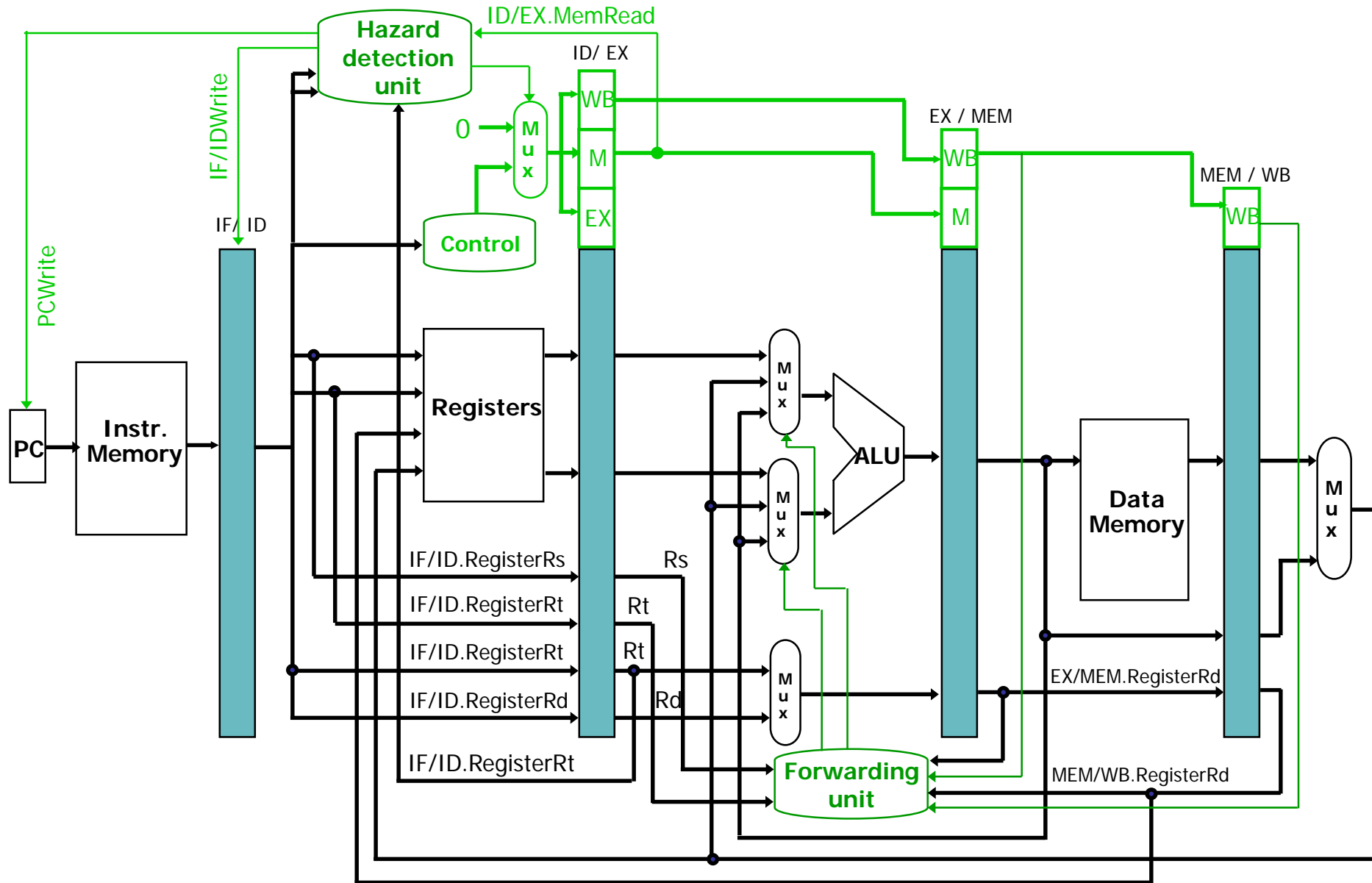
- The control for the hazard detection unit is:

if (ID/EX.MemRd  Checks if the instruction is a load
and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline

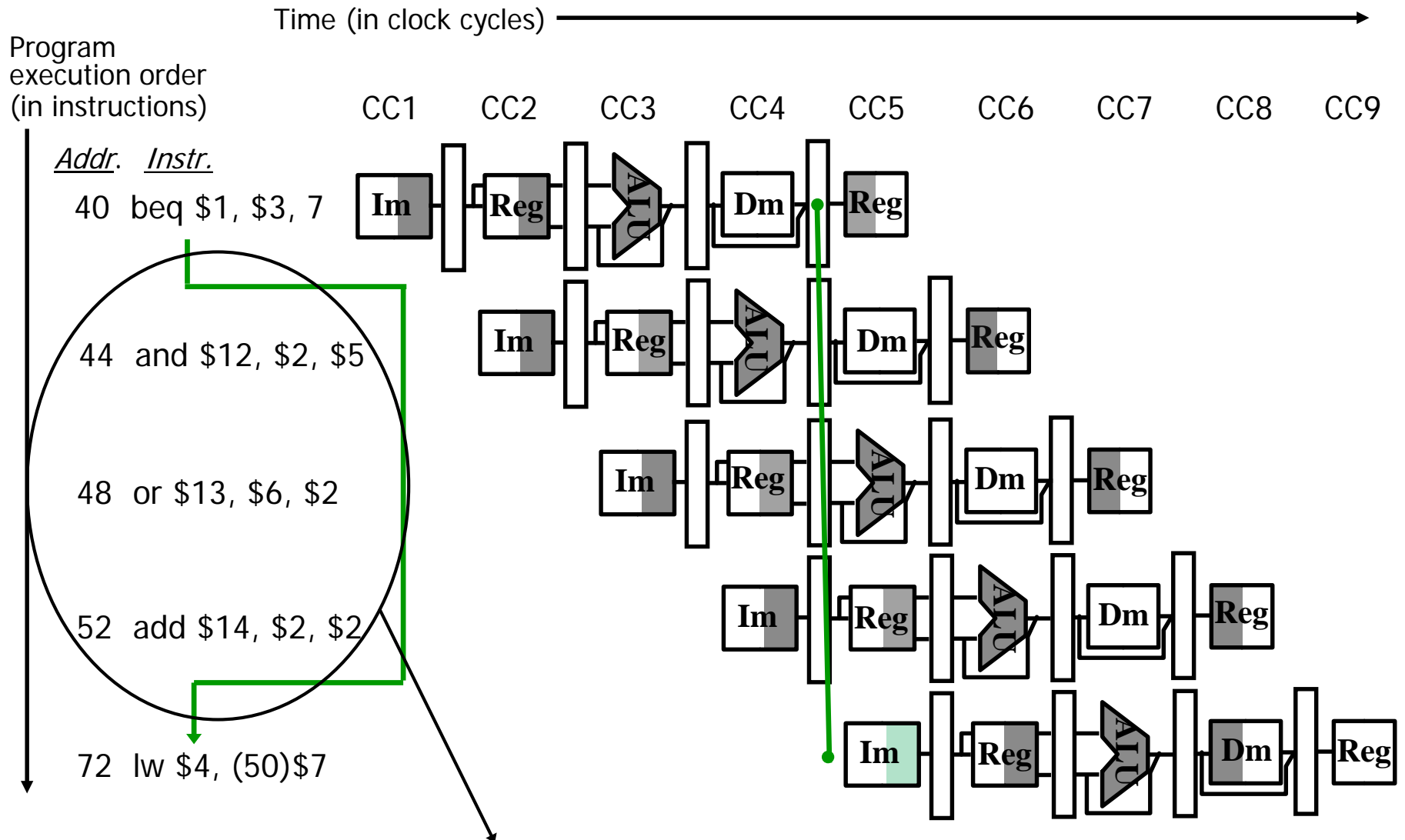
Data Hazards and Stalls (Cont')



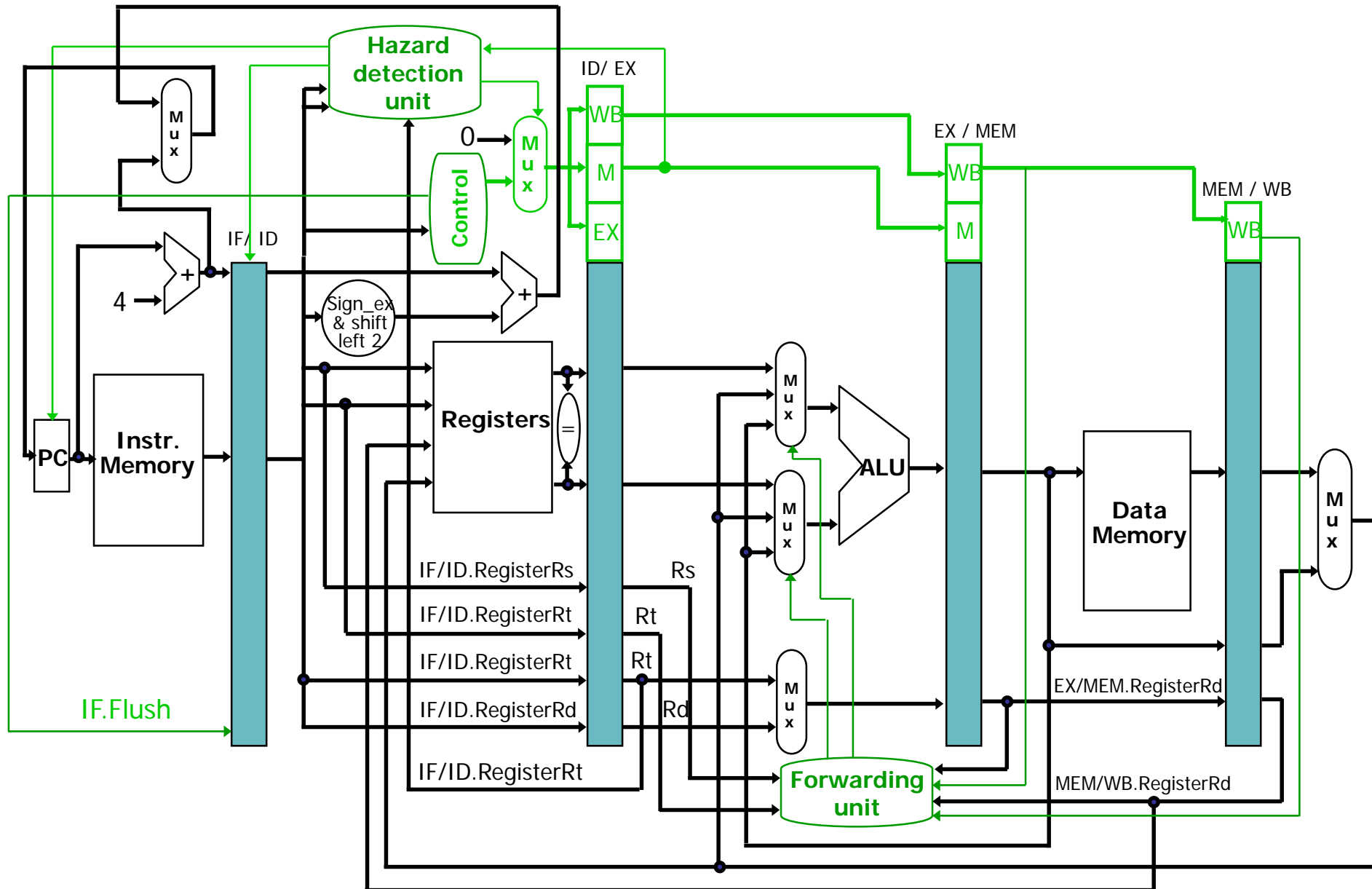
The Pipelined Datapath with Forwarding and Hazard Detection Unit



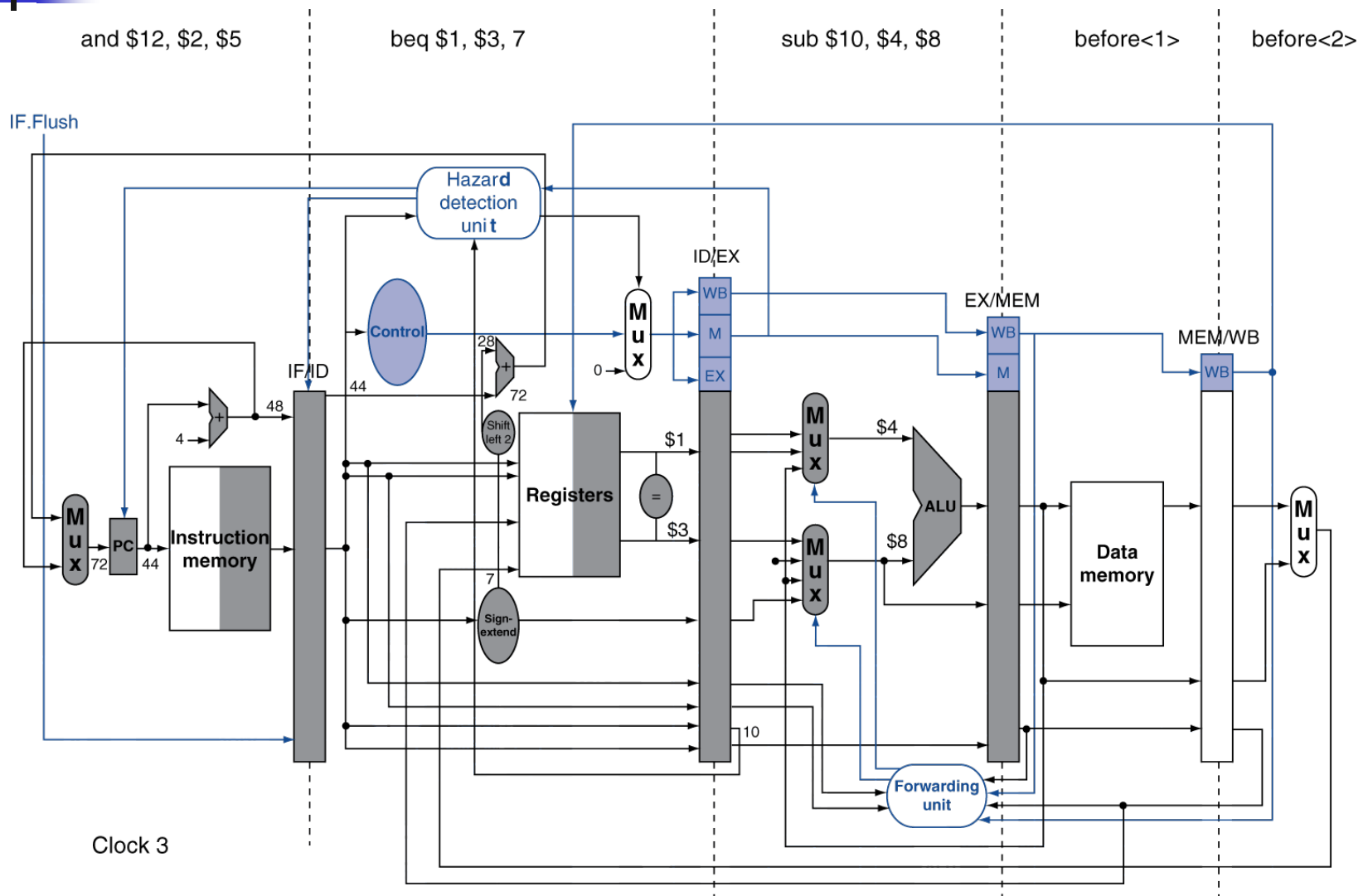
Branch Hazards



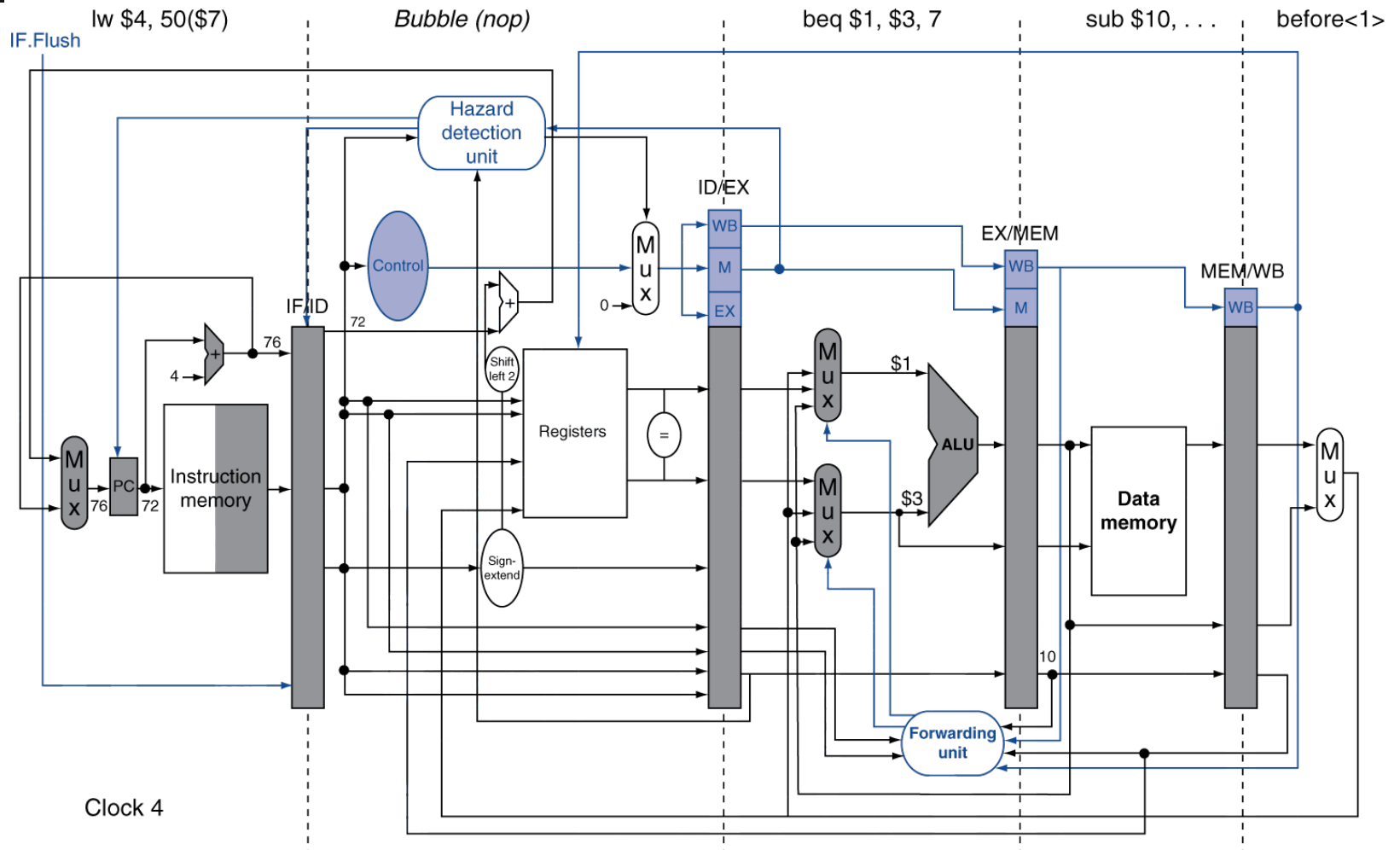
Datapath for Branch (including HW to flush the pipeline)



Example: Branch Taken

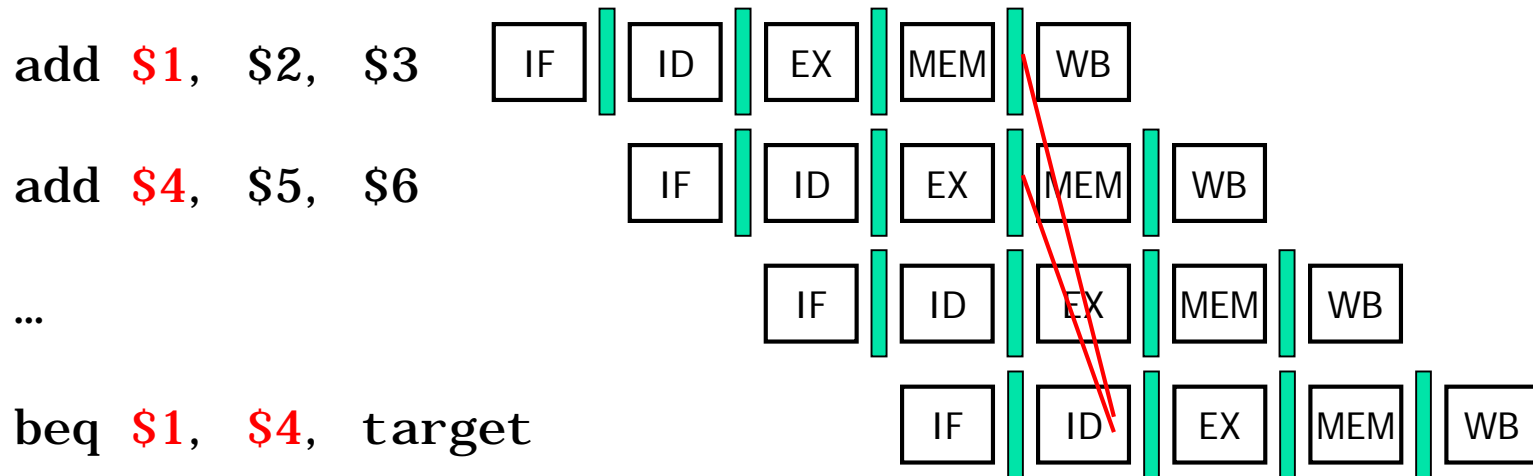


Example: Branch Taken



Data Hazards for Branches

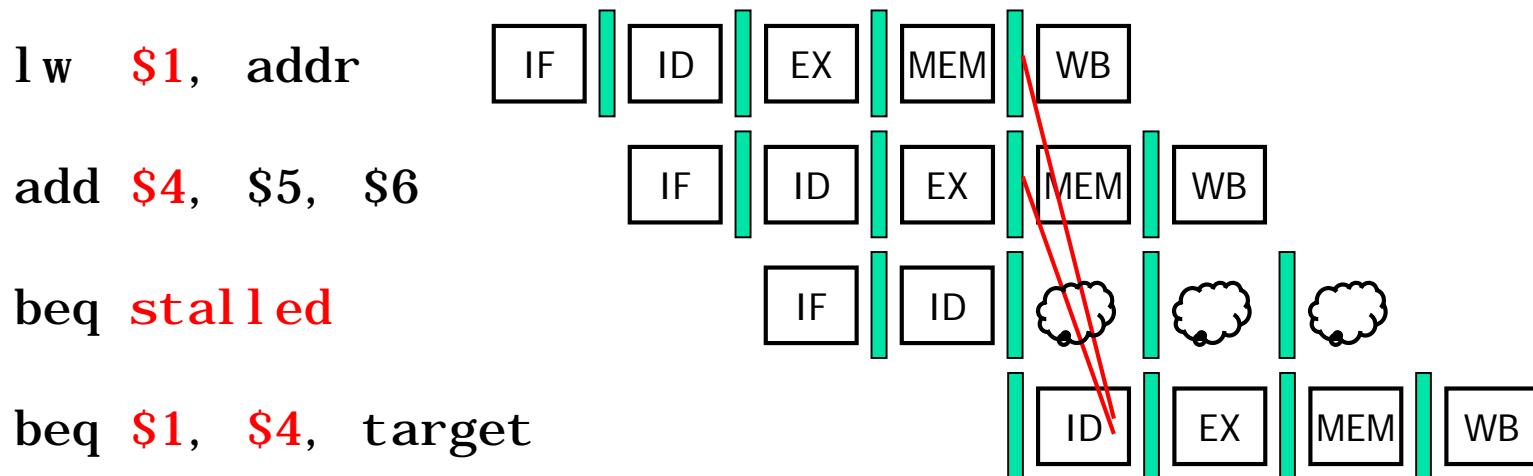
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction



- Can resolve using forwarding

Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
 - Need 1 stall cycle



Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
 - Need 2 stall cycles

