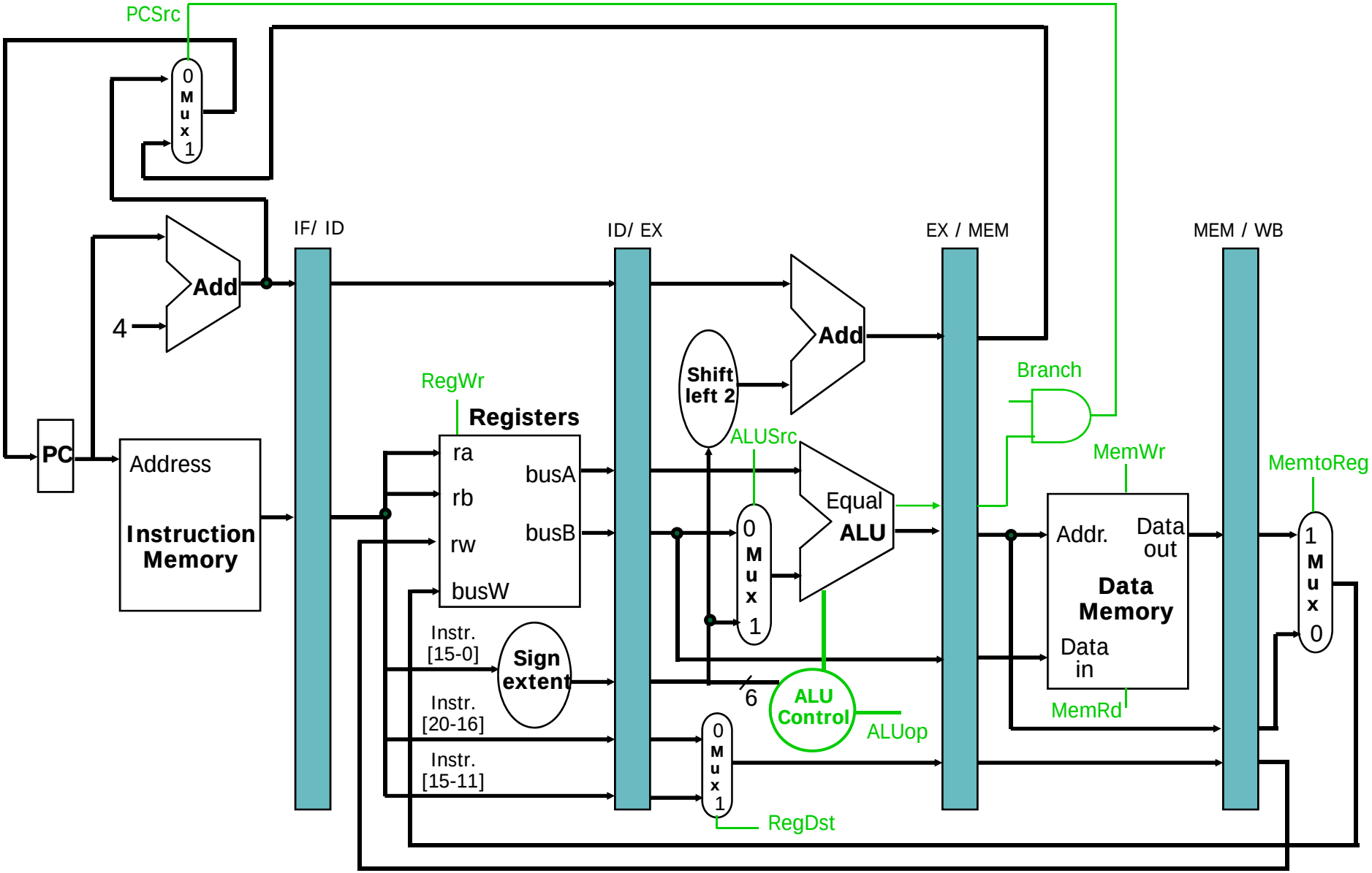
Comp 303
Computer Architecture
A Pipelined Datapath Control

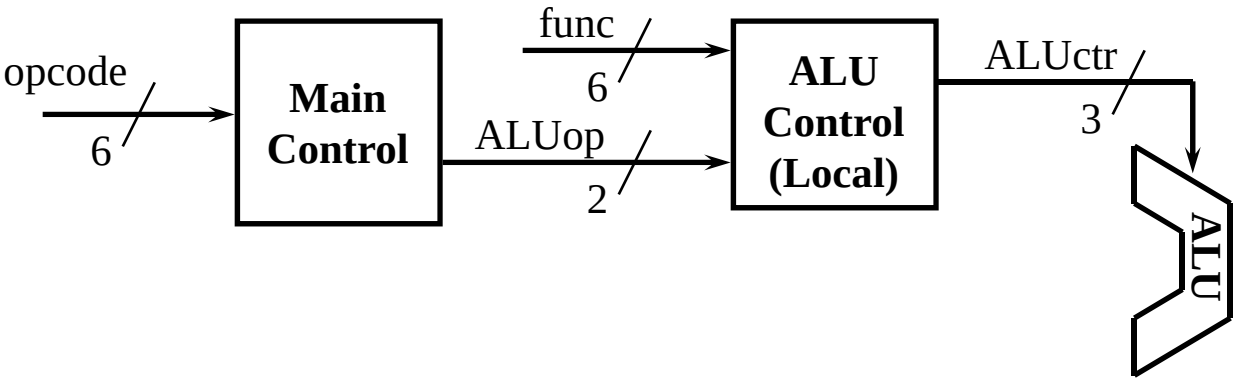
Lecture 13

Pipelined Datapath with Control Signals



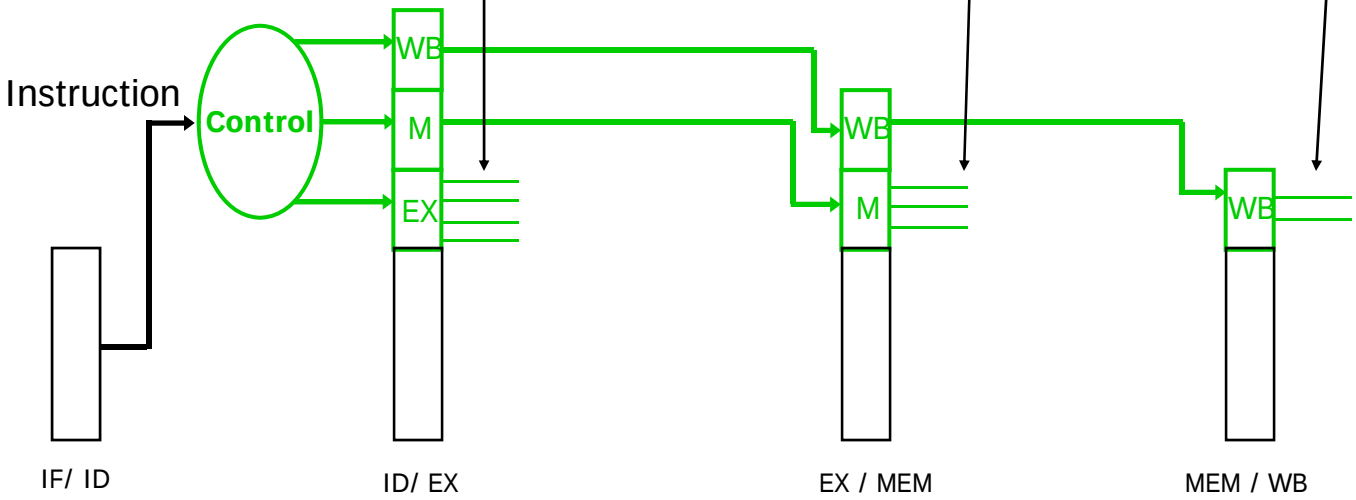
Recall: ALU Control Bits

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**

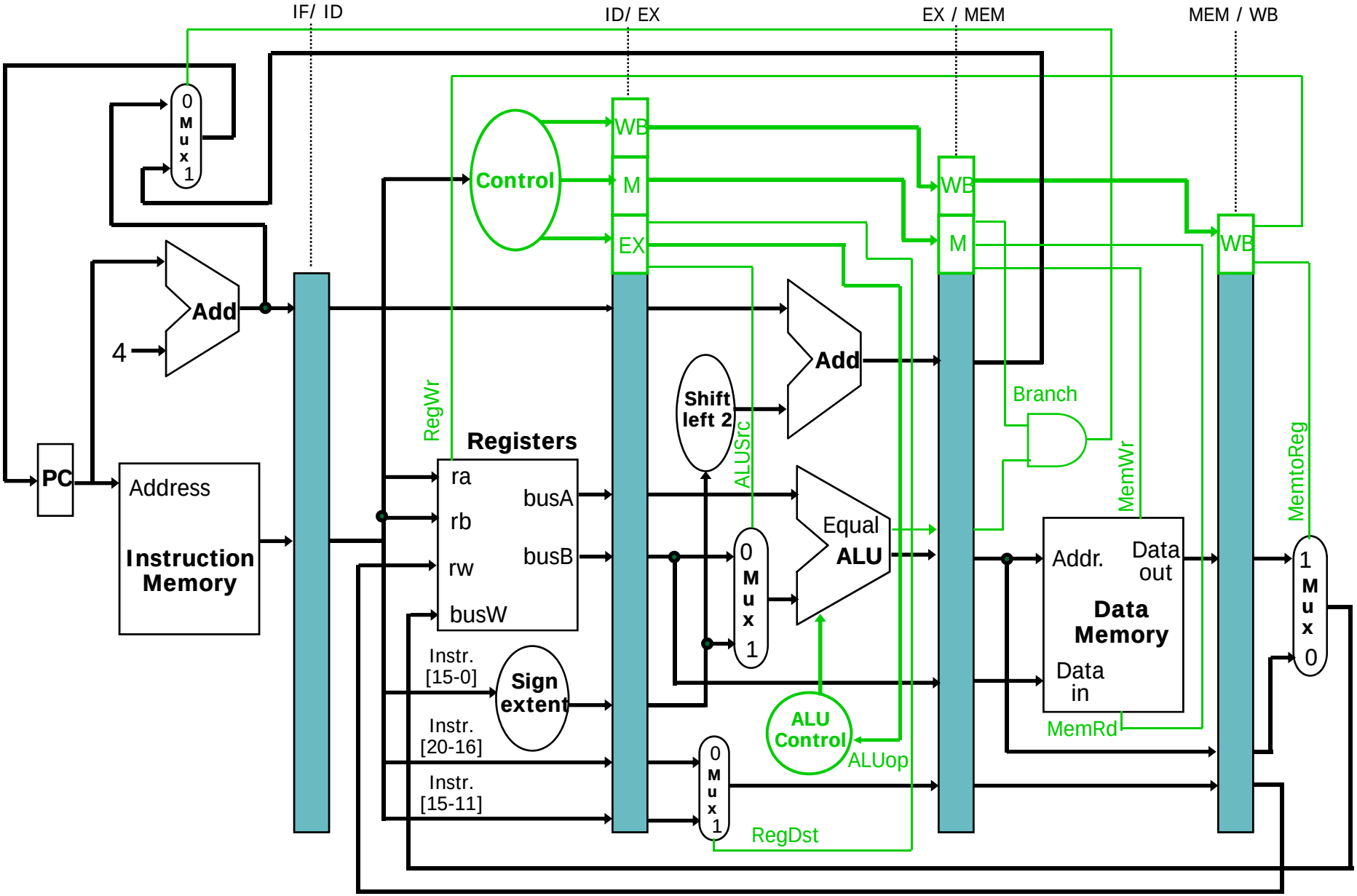


The Values of Control Lines for The Last Three Pipeline Stages

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

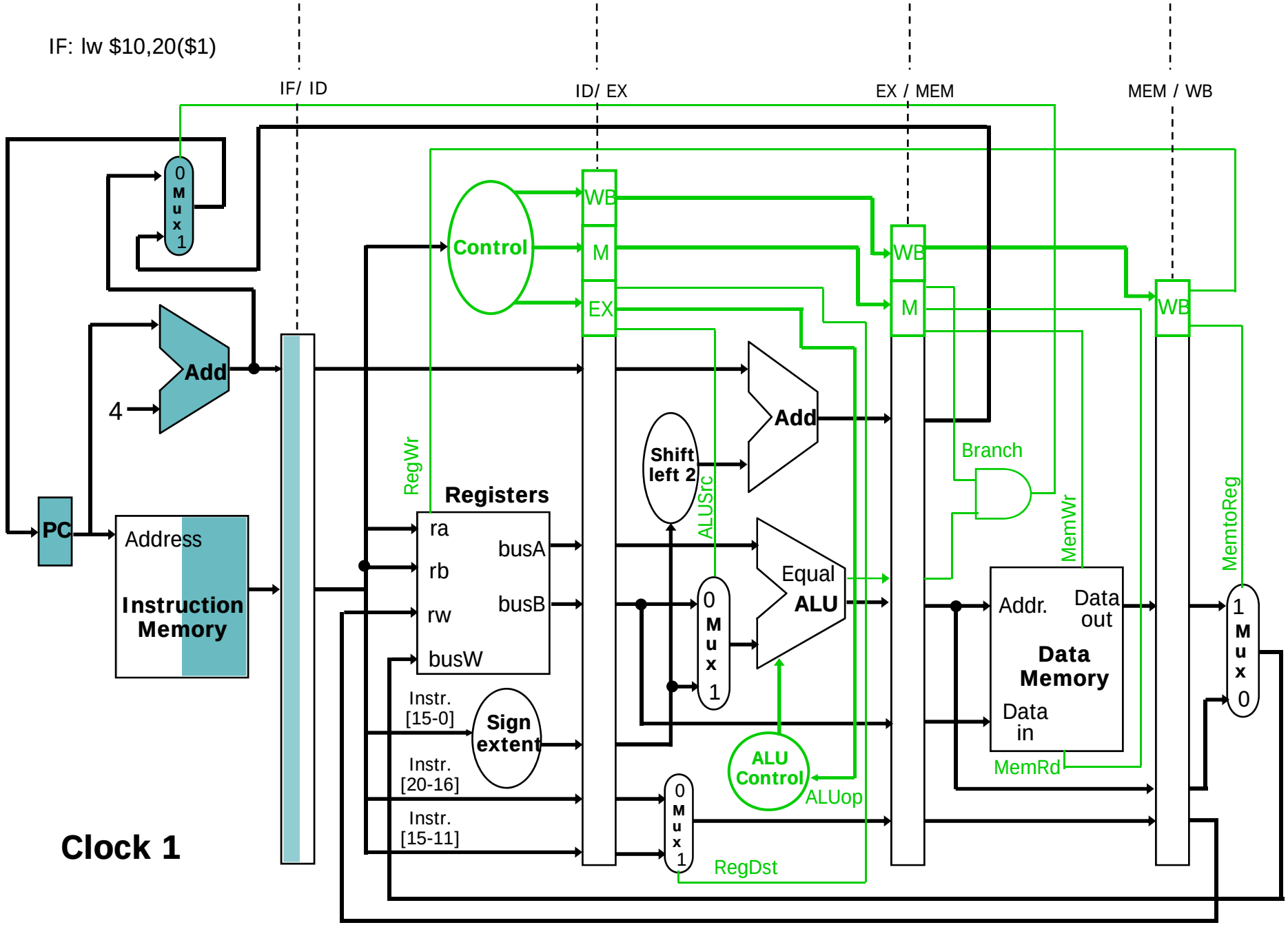


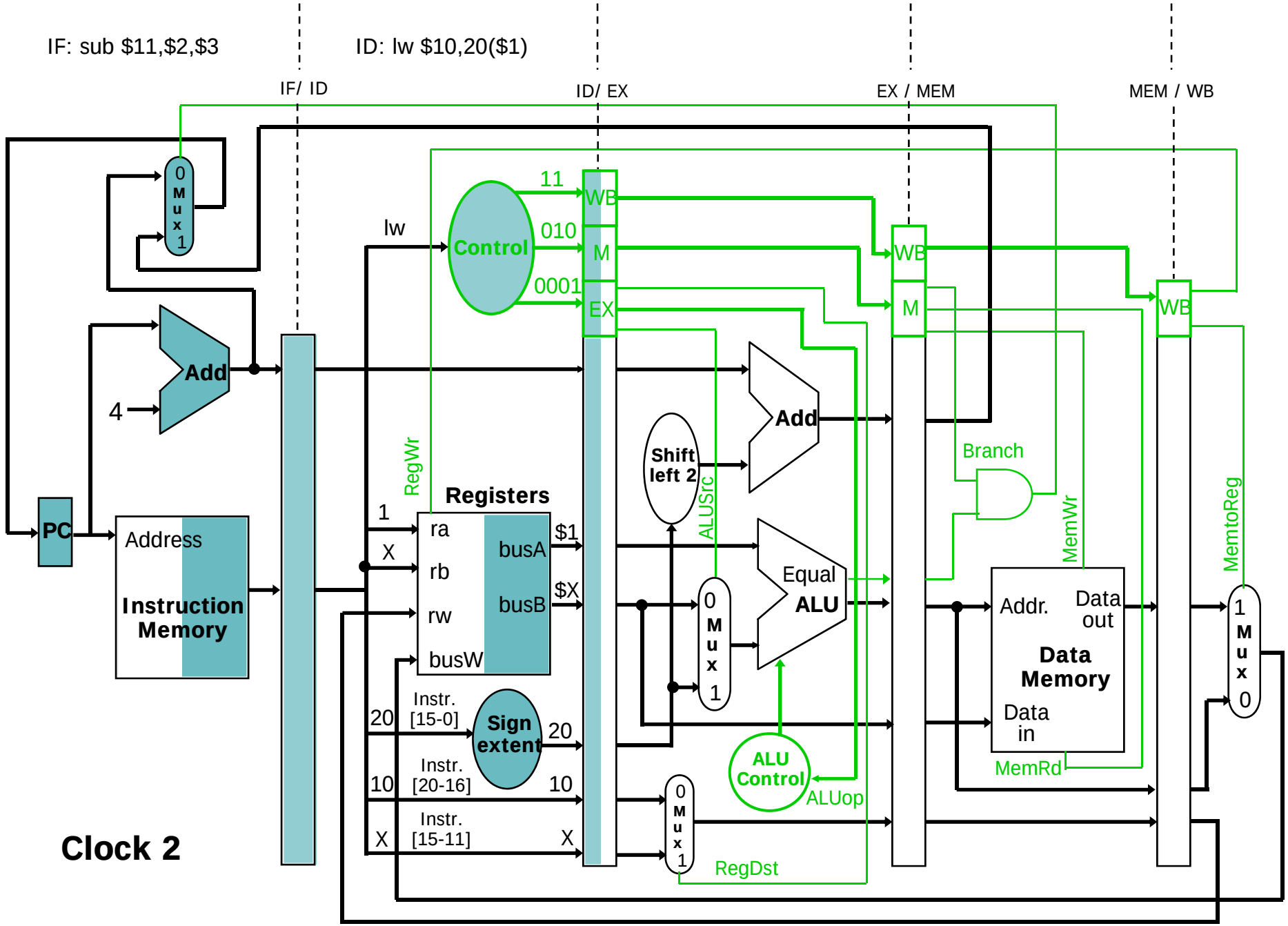
An Example to Clarify Pipelined Control

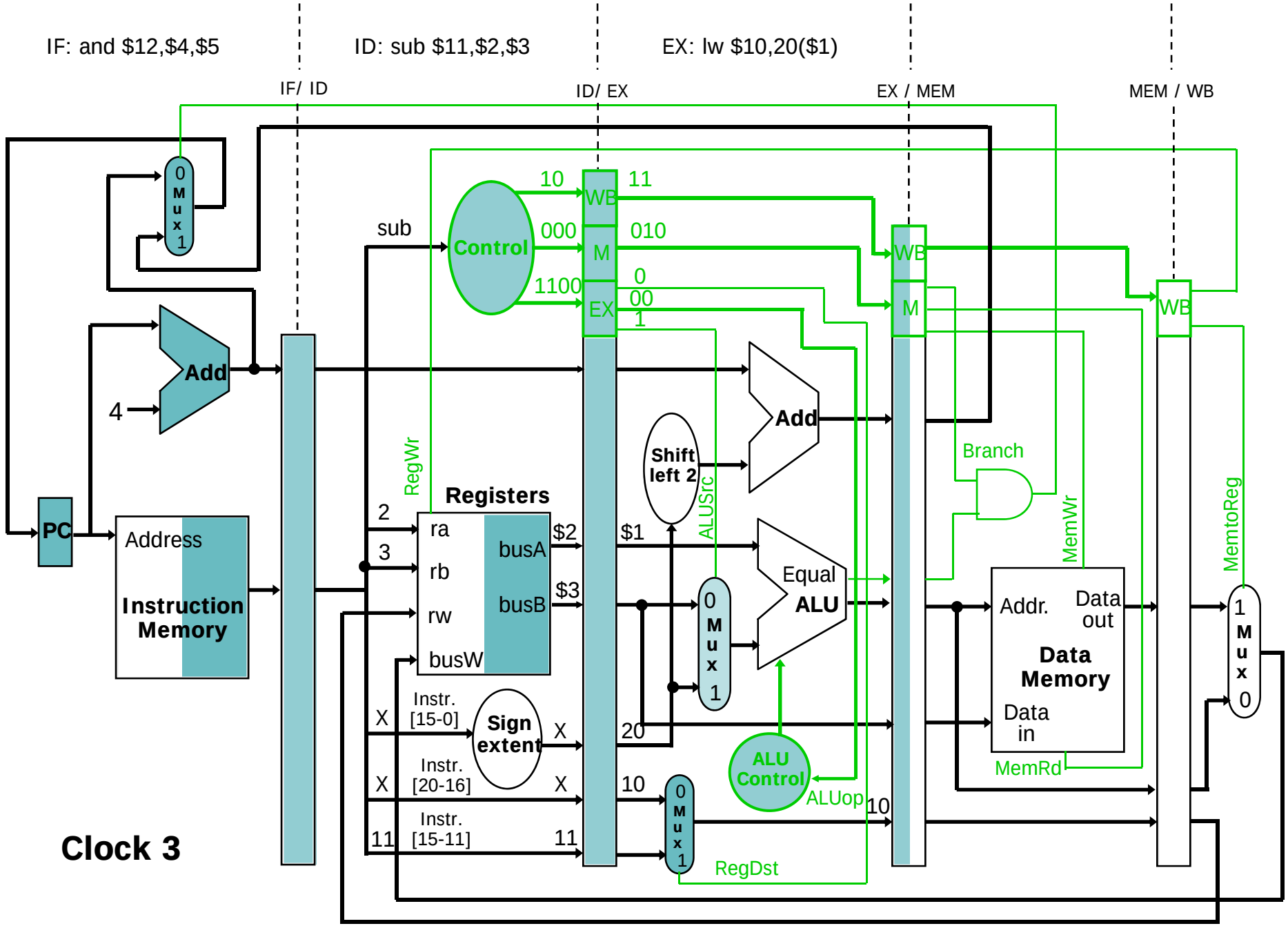
- Let's look at what's happing in the pipeline for the following program.

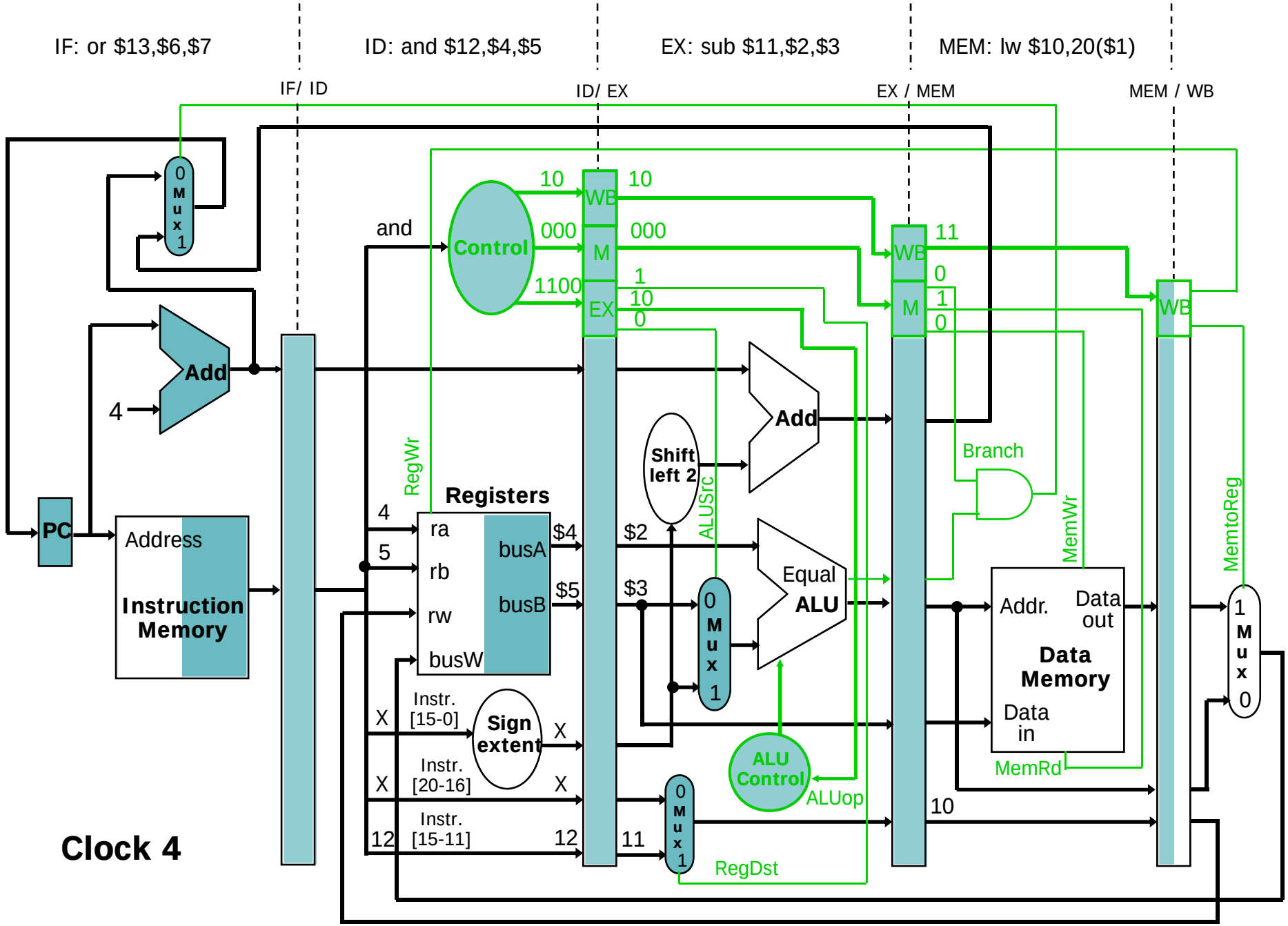
```
lw   $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or   $13, $6, $7
add $14, $8, $9
```

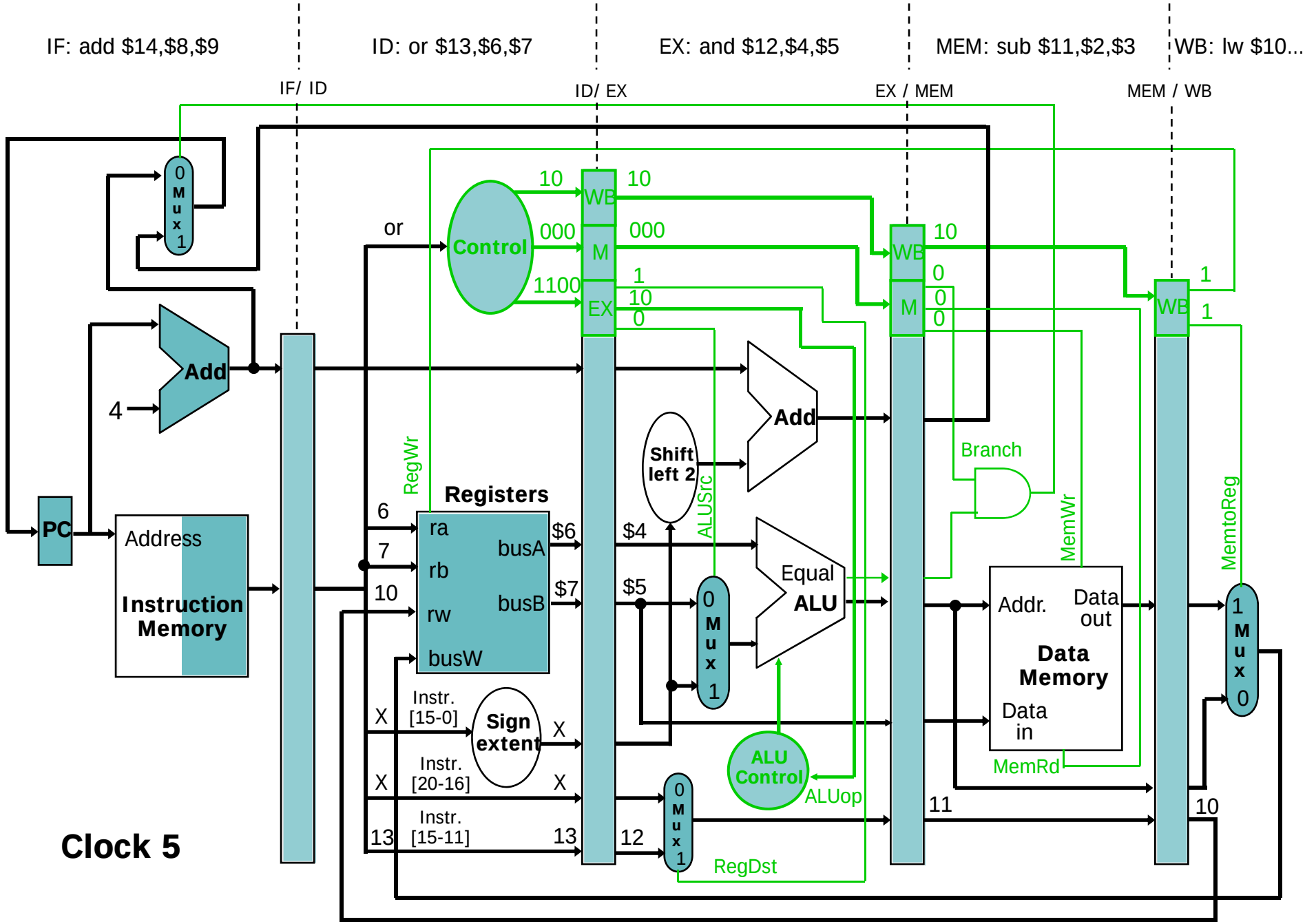
- Code does not have any data, control, or structural hazard.


IF: lw $10,20($1)
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0
M
u
x
1
Add
4
PC
Address
Instruction
Memory
Control
WB
M
EX
RegWr
Registers
ra
rb
rw
busW
busA
busB
Instr.
[15-0]
Sign
extent
Instr.
[20-16]
Instr.
[15-11]
Shift
left 2
Add
ALUSrc
Equal
ALU
ALU
Control
ALUop
RegDst
Branch
MemWr
MemRd
Addr.
Data
out
Data
Memory
Data
in
MemtoReg
Clock 1


IF: sub $11,$2,$3
ID: lw $10,20($1)
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0 M u x 1
Add
4
PC
Address
Instruction Memory
lw
Control
11
010
0001
WB
M
EX
WB
M
WB
Add
Shift left 2
ALUSrc
Branch
RegWr
Registers
1
X
ra
rb
rw
busW
busA
busB
$1
$X
Equal
ALU
0 M u x 1
MemWr
MemtoReg
Addr.
Data out
Data Memory
Data in
1 M u x 0
MemRd
20
Instr. [15-0]
Sign extent
20
10
Instr. [20-16]
10
X
Instr. [15-11]
X
ALU Control
ALUop
0 M u x 1
RegDst
Clock 2


IF: and $12,$4,$5
ID: sub $11,$2,$3
EX: lw $10,20($1)
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0 M u x 1
Add
4
PC
Address
Instruction Memory
sub
Control
10
000
1100
WB
M
EX
11
010
0
00
1
WB
M
WB
Add
Shift left 2
RegWr
Registers
2
3
ra
rb
rw
busW
busA
busB
$2
$3
$1
ALUSrc
Equal
ALU
0 M u x 1
Branch
MemWr
MemtoReg
Addr.
Data out
Data Memory
Data in
MemRd
1 M u x 0
Instr. [15-0]
Sign extent
X
20
Instr. [20-16]
10
Instr. [15-11]
11
ALU Control
ALUop
10
RegDst
Clock 3


IF: or $13,$6,$7
ID: and $12,$4,$5
EX: sub $11,$2,$3
MEM: lw $10,20($1)
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0 M u x 1
Add
4
PC
Address
Instruction Memory
and
Control
10
000
1100
WB
M
EX
10
000
1
10
0
11
0
1
0
Add
Shift left 2
RegWr
Registers
4
5
ra
rb
rw
busW
busA
busB
$4
$5
$2
$3
ALUSrc
0 M u x 1
Equal
ALU
Branch
MemWr
MemtoReg
Addr.
Data out
Data Memory
Data in
MemRd
1 M u x 0
X
Instr. [15-0]
Sign extent
X
X
Instr. [20-16]
X
12
Instr. [15-11]
12
11
0 M u x 1
ALU Control
ALUop
RegDst
10
Clock 4


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
Clock 5
PC
Add
4
Address
Instruction Memory
0 M u x 1
or
Control
10
000
1100
WB
M
EX
Registers
ra
rb
rw
busW
busA
busB
6
7
10
$6
$7
RegWr
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent
X
13
Shift left 2
Add
$4
$5
12
ALUSrc
Equal
ALU
ALU Control
ALUop
RegDst
Branch
MemWr
MemRd
Addr.
Data out
Data Memory
Data in
11
MemtoReg
1 M u x 0

Data Hazards

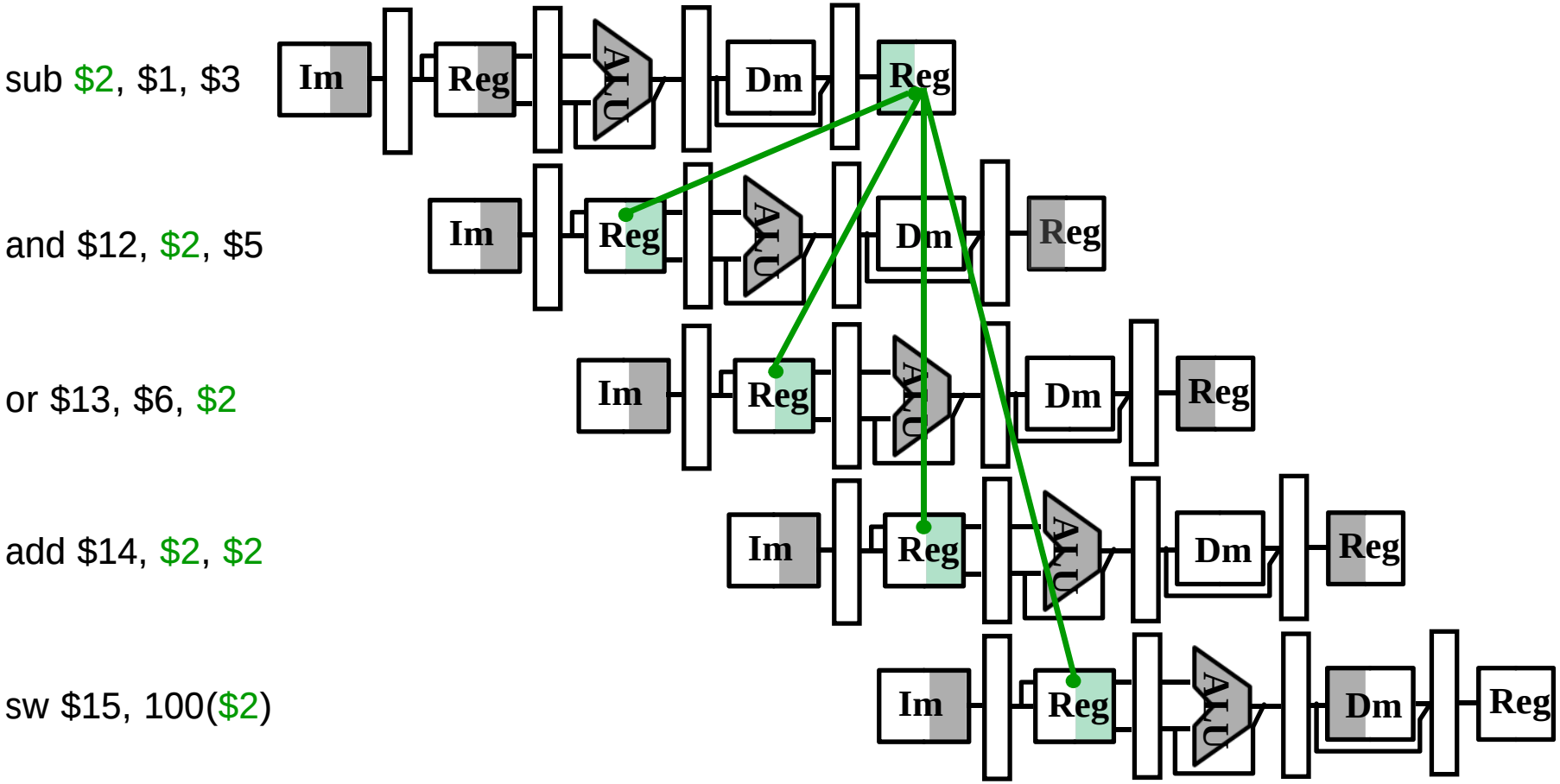
- Previous example shows us how independent instructions that do not use the results calculated by prior instructions are executed.
- This is not the case with real programs.
- Let's look at the following code sequence.

```
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')

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
The value of $2:	10	10	10	10	10/-20	-20	-20	-20	-20



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.

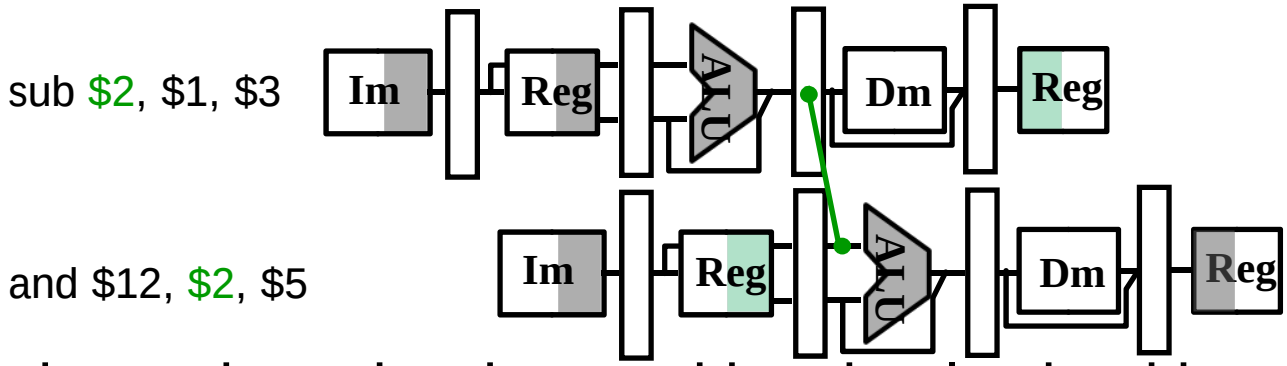
```
sub     $2, $1, $3
nop
nop
and     $12, $2, $5
or      $13, $6, $2
add     $14, $2, $2
sw      $15, 100($2)
```

- **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:

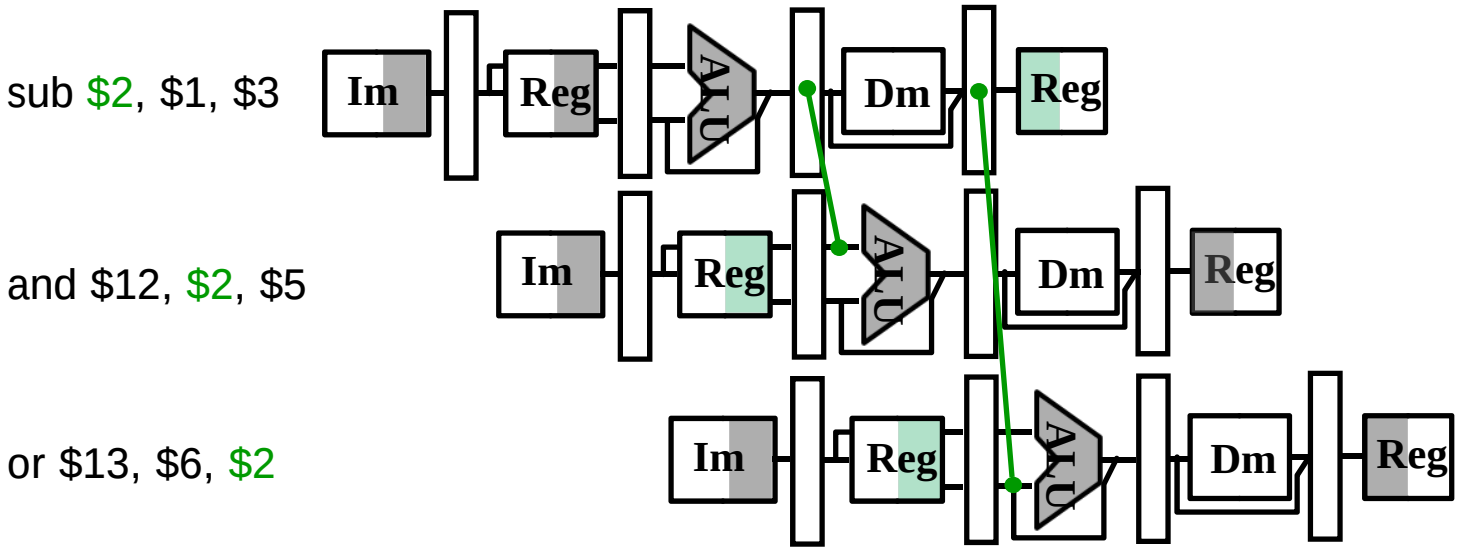


- This hazard can be detected by simply checking:

EX/MEM.RegisterRd = ID/EX.RegisterRs = $2

Data Hazard Detection & Forwarding (Cont')

- Another hazard is between `sub-or` instructions:



- This hazard can be detected by simply checking:

  MEM/WB.RegisterRd = 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:

```
sw   $1, 100($5)
add  $3, $1, $2
```

EX/MEM.RegisterRd = ID/EX.RegisterRs

~~$1 = 100+$5~~

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

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 ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA=10

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

MEM hazard:

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

if (MEM/WB.RegWr
and (MEM/WB.RegisterRd ≠ 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:

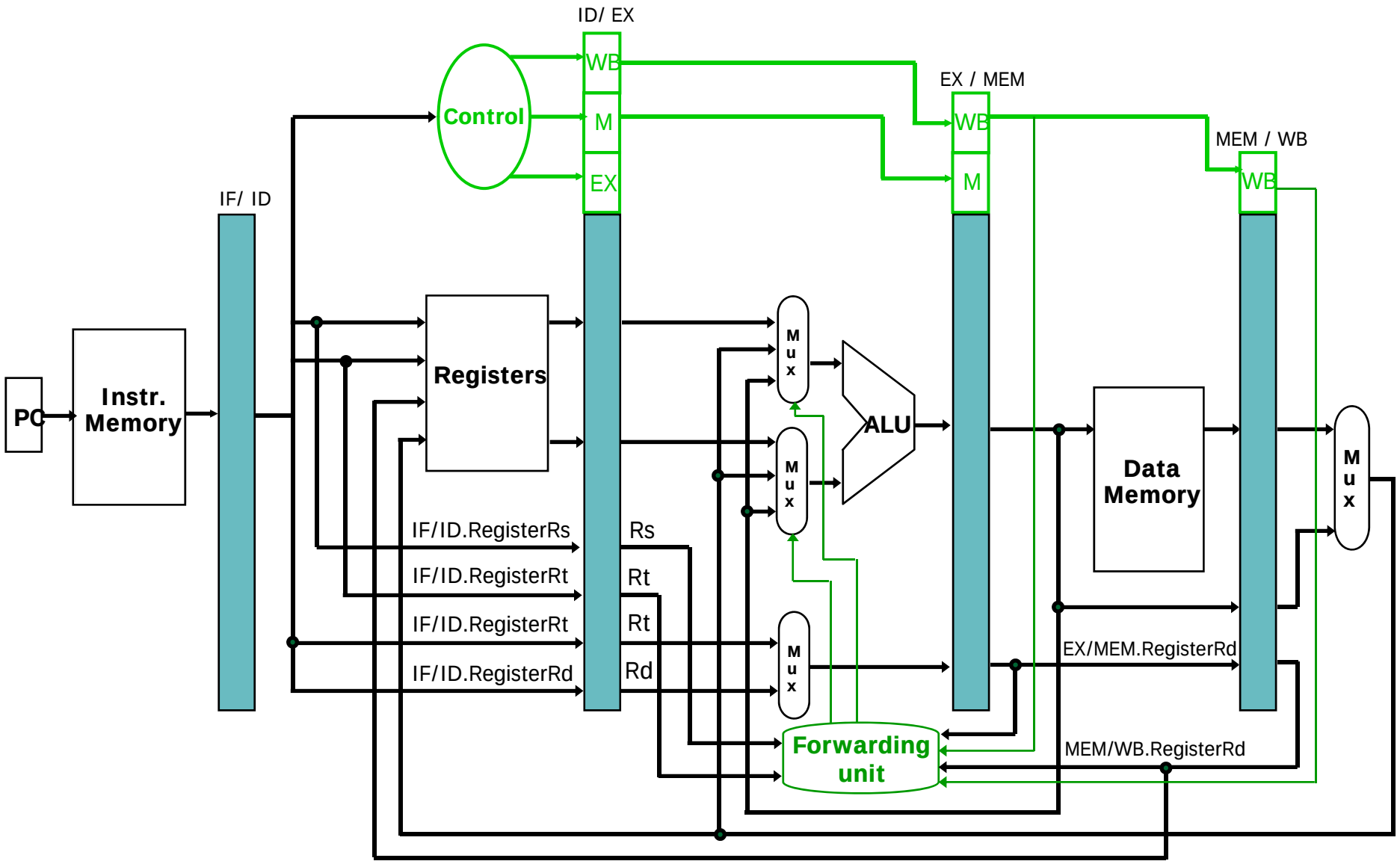
```
if (MEM/WB.RegWr
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA=01

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

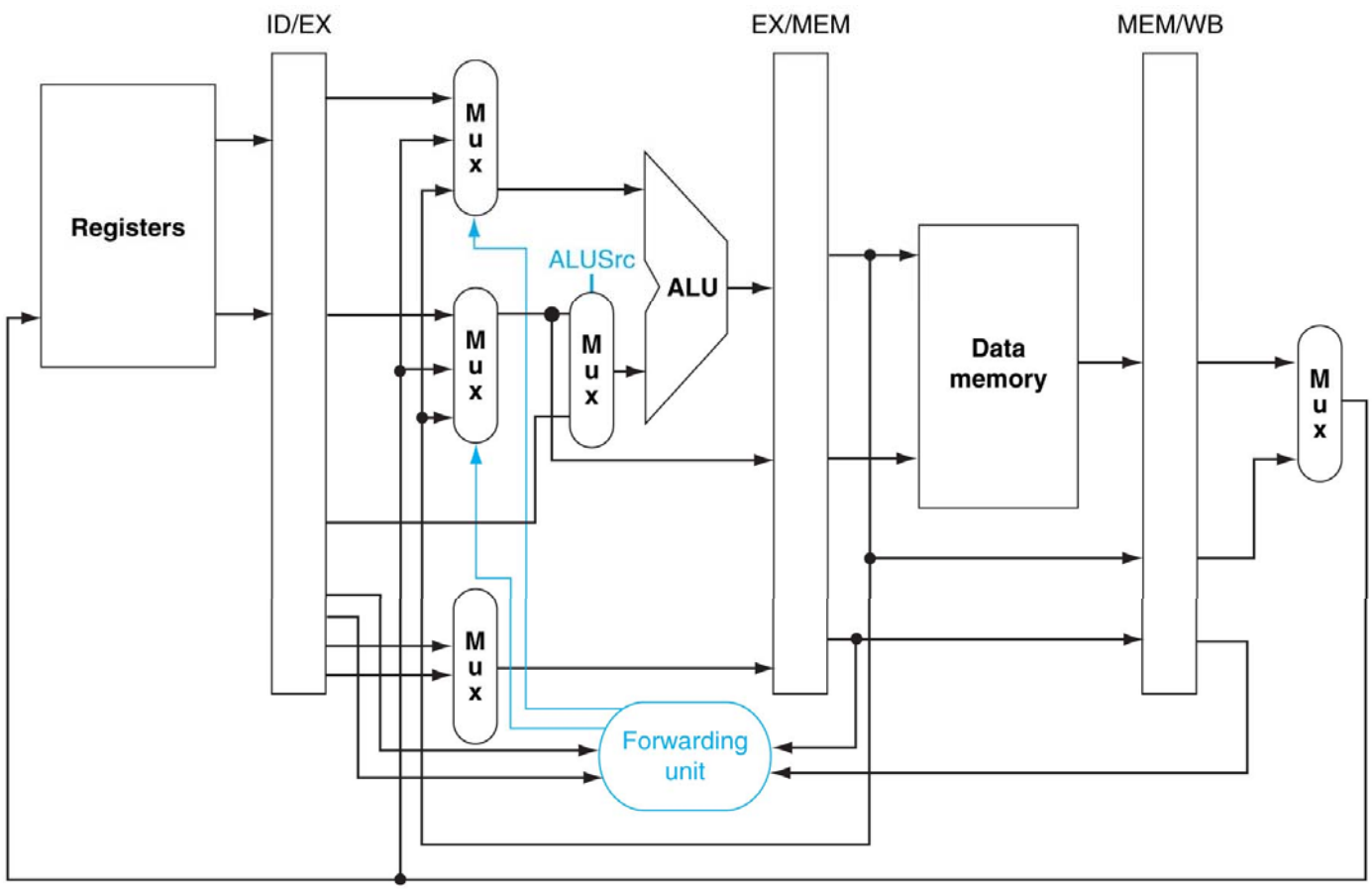
Summary of Data Hazard Conditions (Cont')

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.

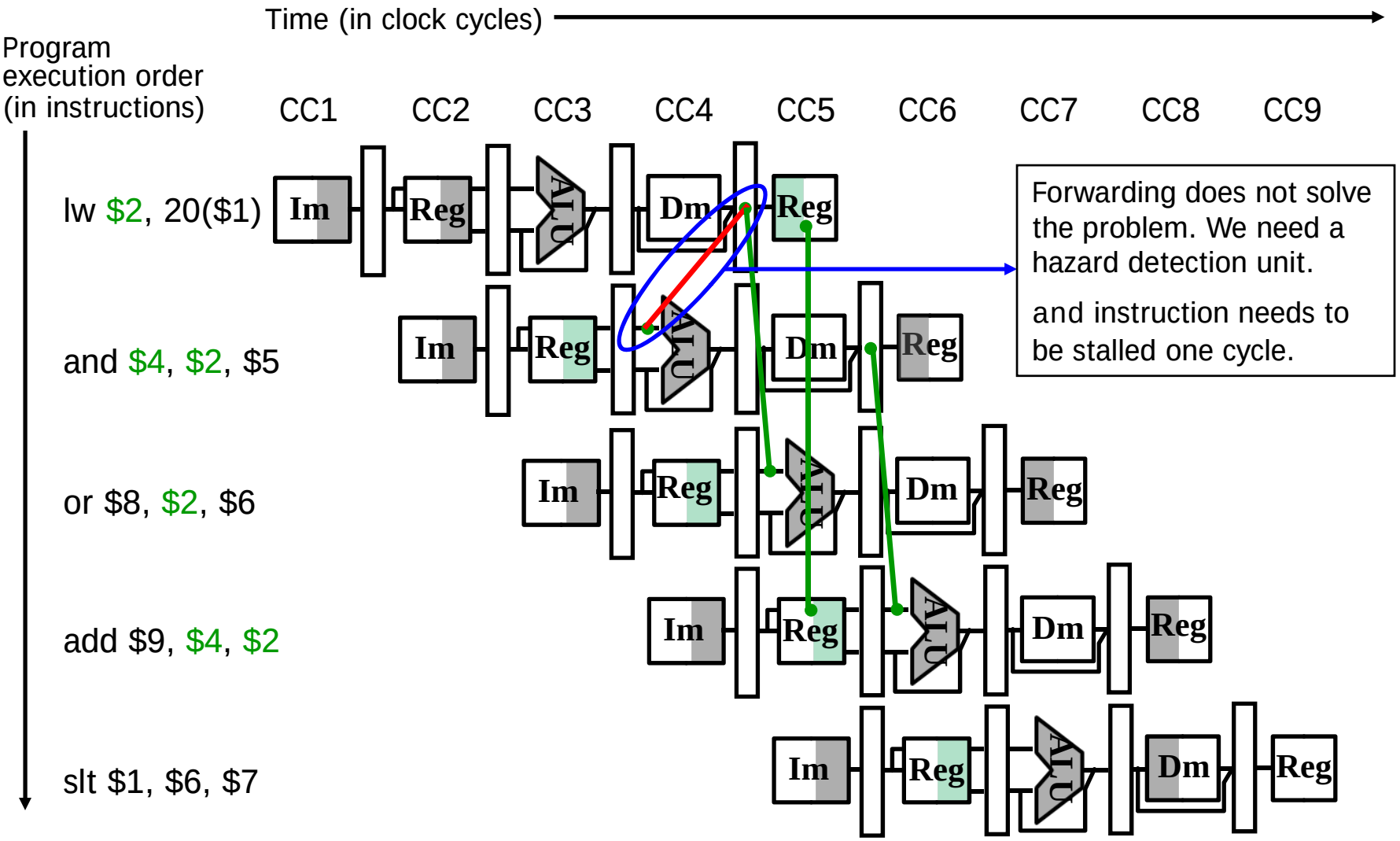
The Pipelined Datapath with Forwarding



Signed Immediate Enhancement



Data Hazards and Stalls



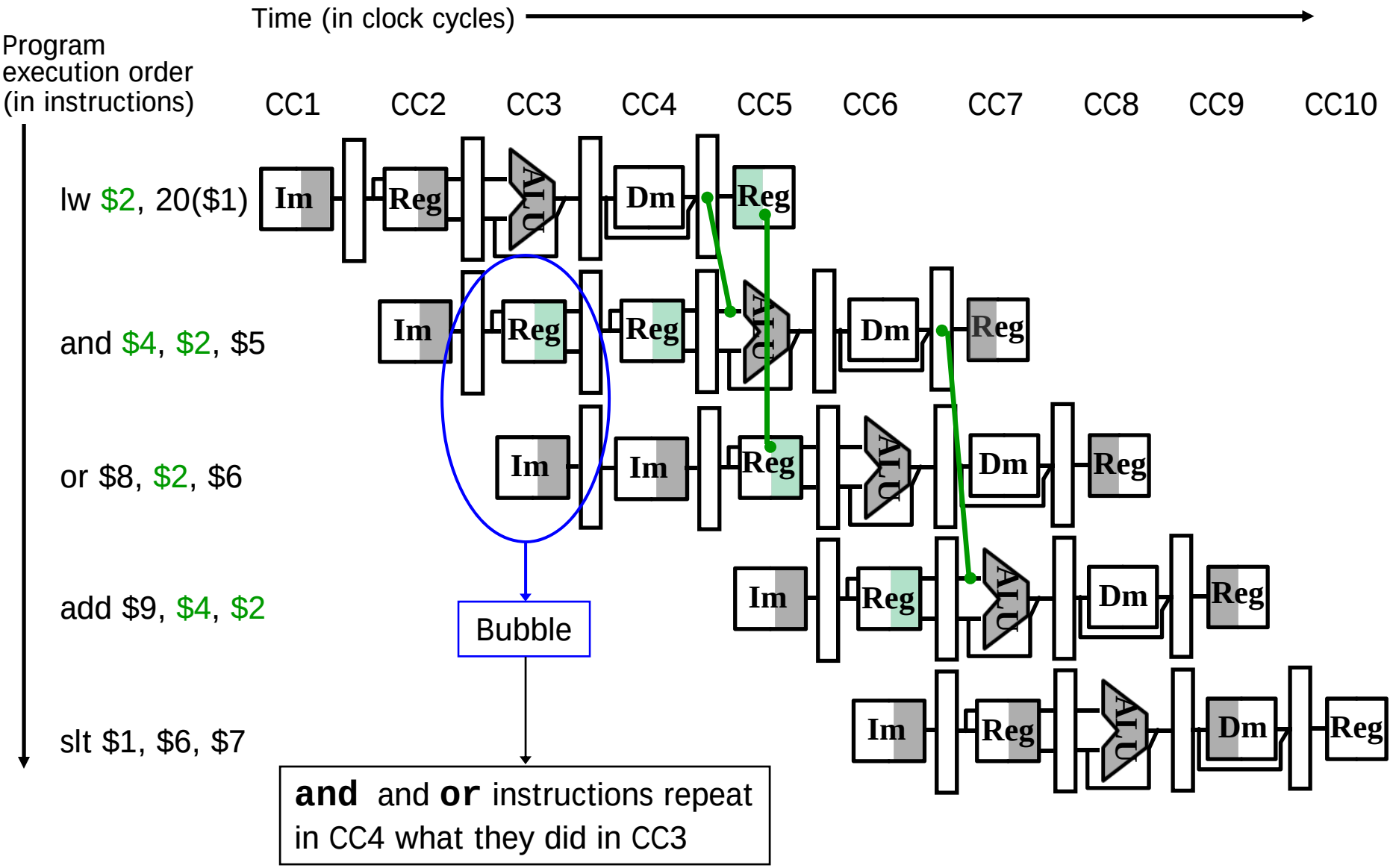
Hazard Detection

- The control for the hazard detection unit is:

```
if (ID/EX.MemRd
  and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
```

Checks if the instruction is a load

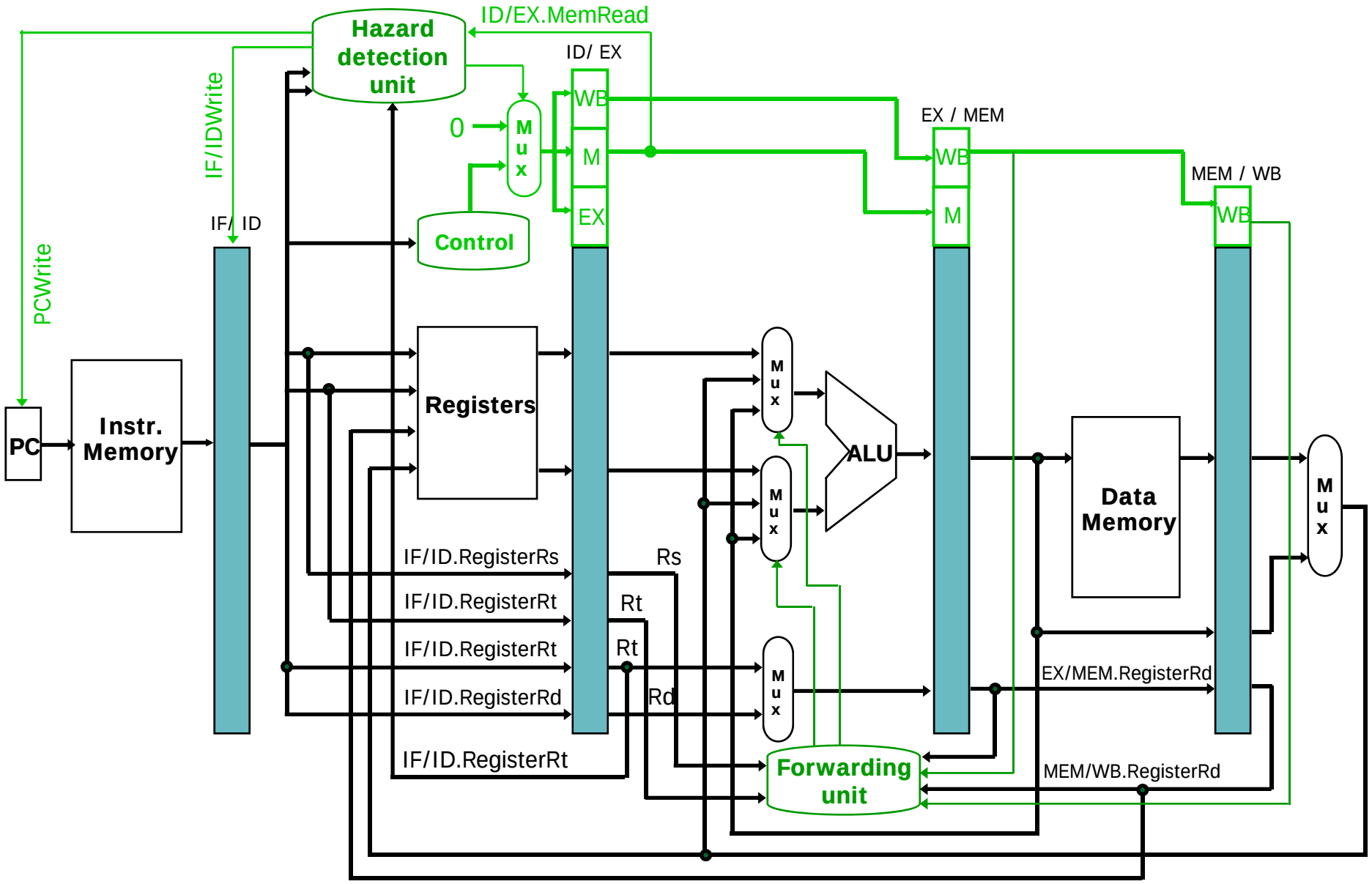
Data Hazards and Stalls (Cont')



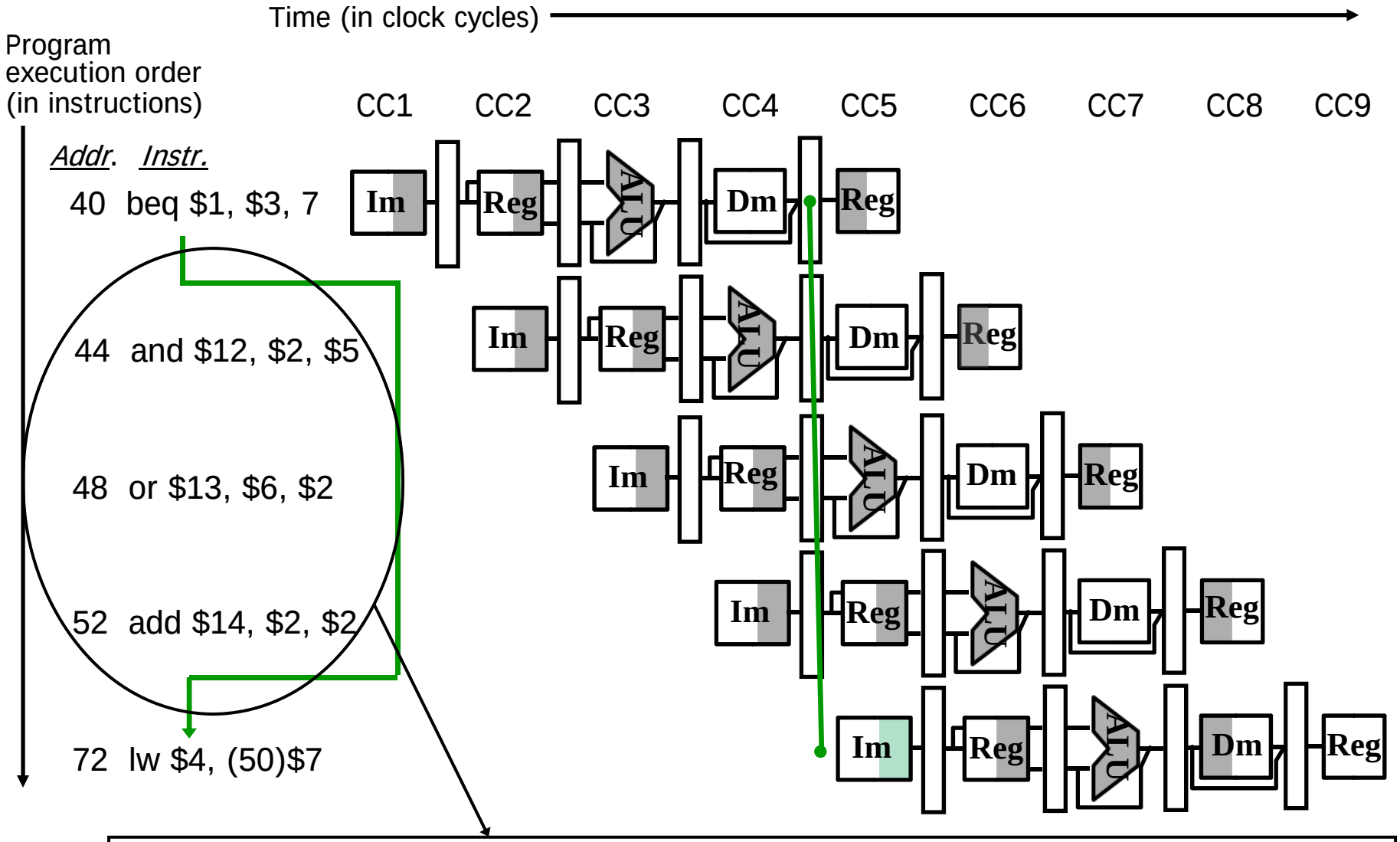
Data Hazards and Stalls (Cont')

Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The Pipelined Datapath with Forwarding and Hazard Detection Unit

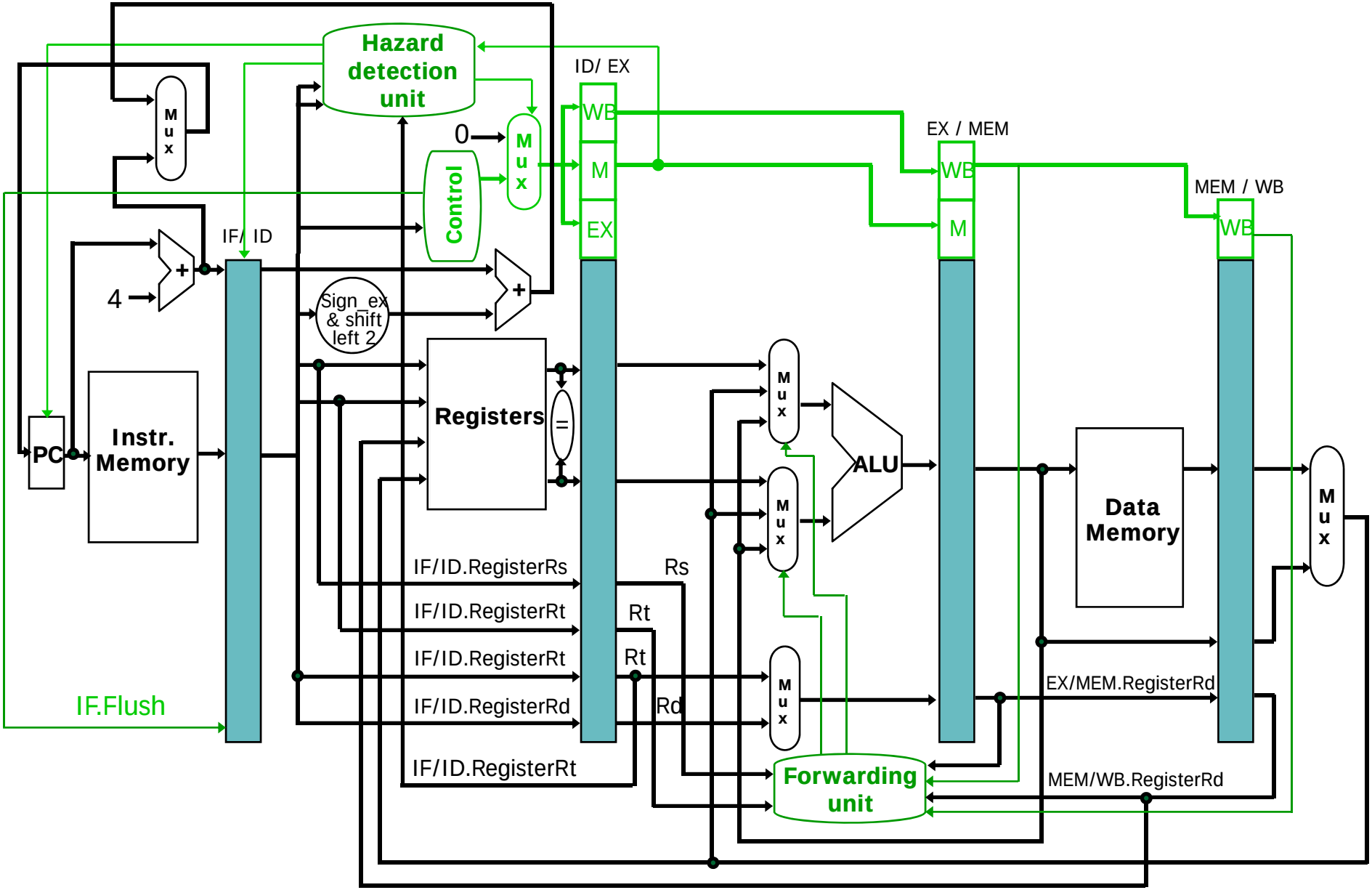


Branch Hazards



Actually, the number of instructions needs to be flushed can be reduced from 3 to 1 instruction (shown in the following slide) when the direction of branch is mispredicted.

Datapath for Branch (including HW to flush the pipeline)



Reading Assignment

- Read Chapter 4
- Find the error in one of the Hazard Detection Functions in the chapter.