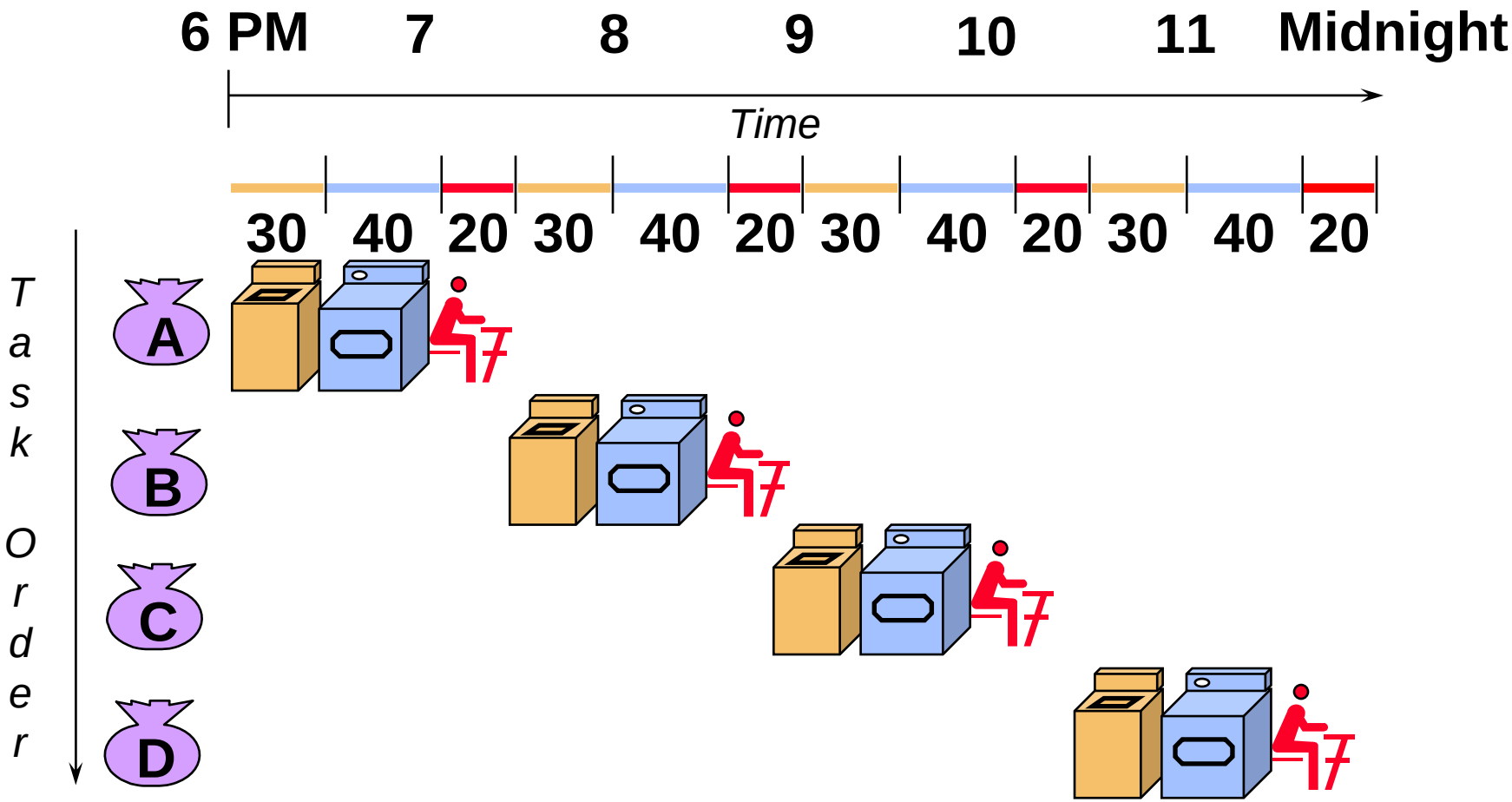
PS Midterm 2

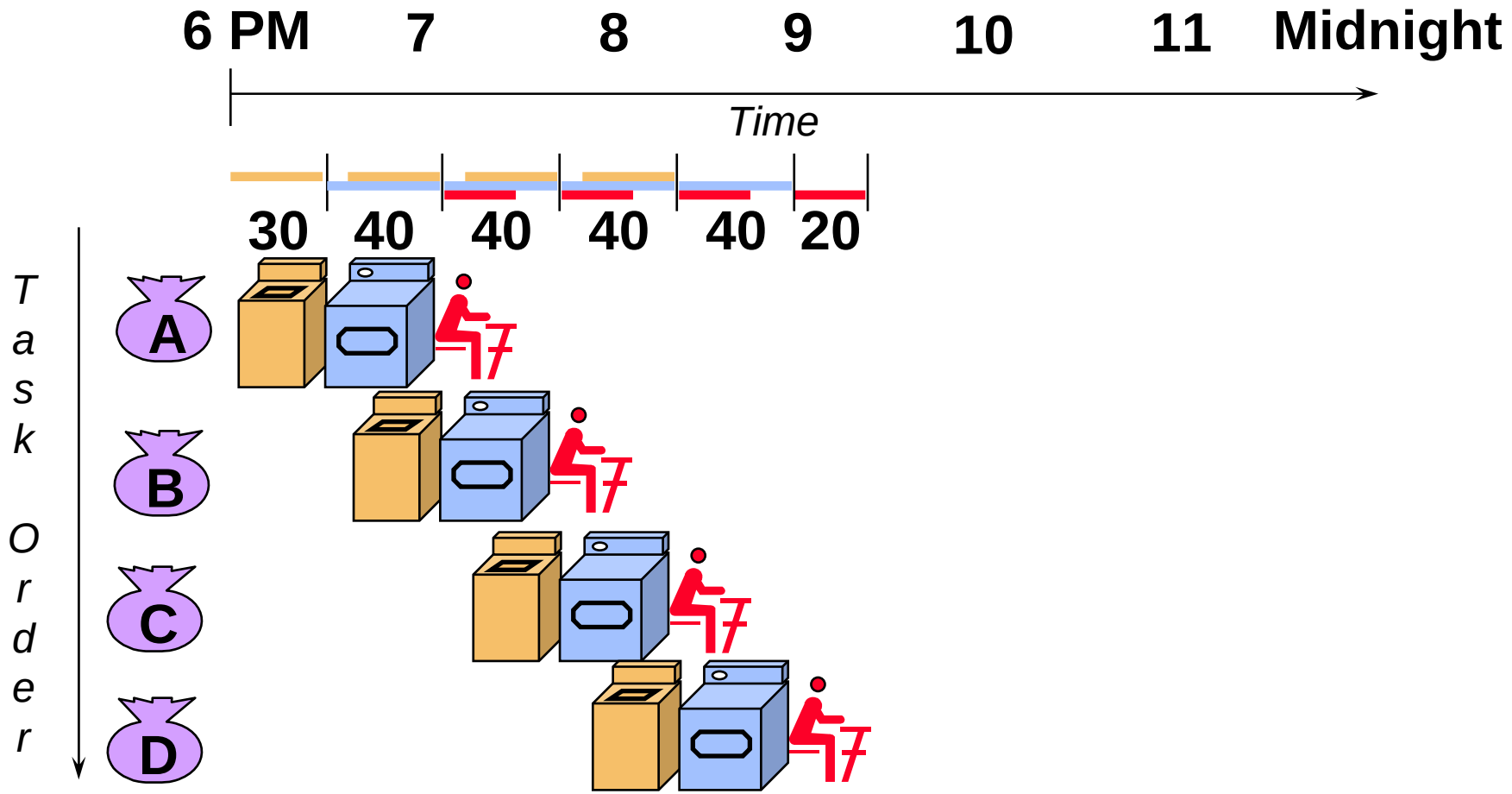
Pipelining

Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



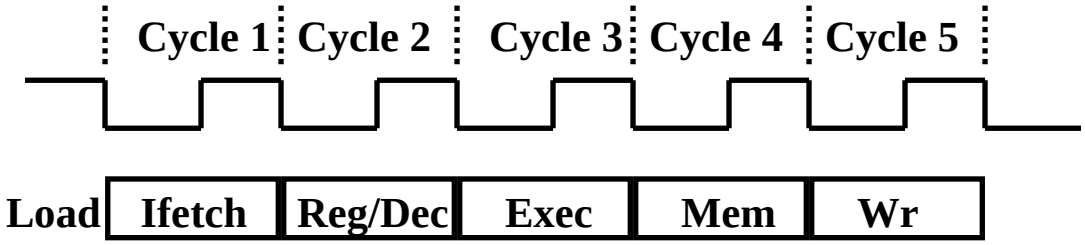
- Pipelined laundry takes 3.5 hours for 4 loads

Total Time for Eight Instructions

Instr. class	Instr. fetch	Register read	ALU operation	Data access	Register write	Total time
Load word	2 ns	1 ns	2 ns	2 ns	1 ns	**8 ns**
Store word	2 ns	1 ns	2 ns	2 ns		**7 ns**
R-type	2 ns	1 ns	2 ns		1 ns	**6 ns**
Branch	2 ns	1 ns	2 ns			**5 ns**

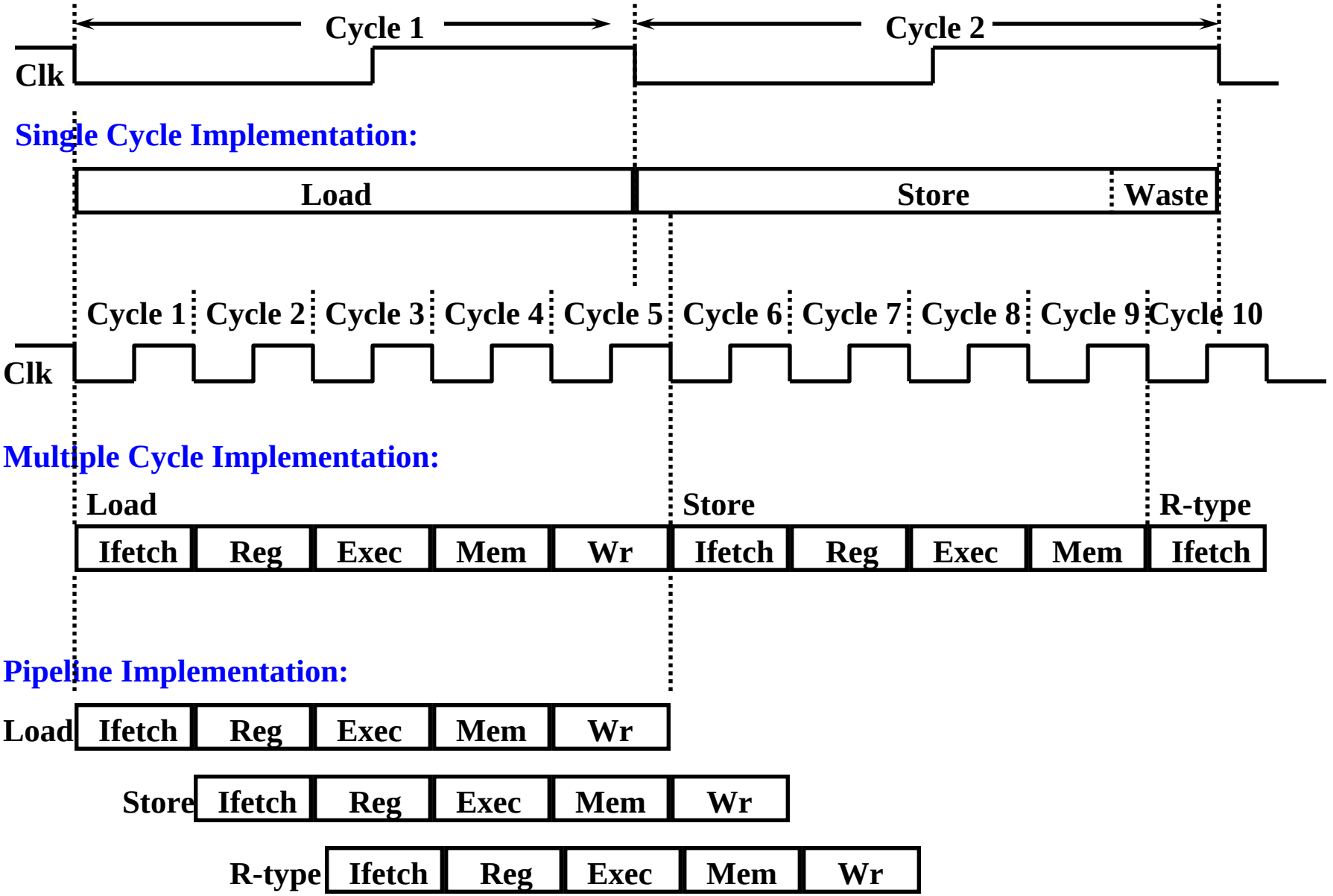
- R-type instructions: add, sub, and, or, slt

The Five **Stages** of the Load Instruction



- **Ifetch**: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- **Reg/Dec**: Registers Fetch and Instruction Decode
- **Exec**: Calculate the memory address
- **Mem**: Read the data from the Data Memory
- **Wr**: Write the data back to the register file

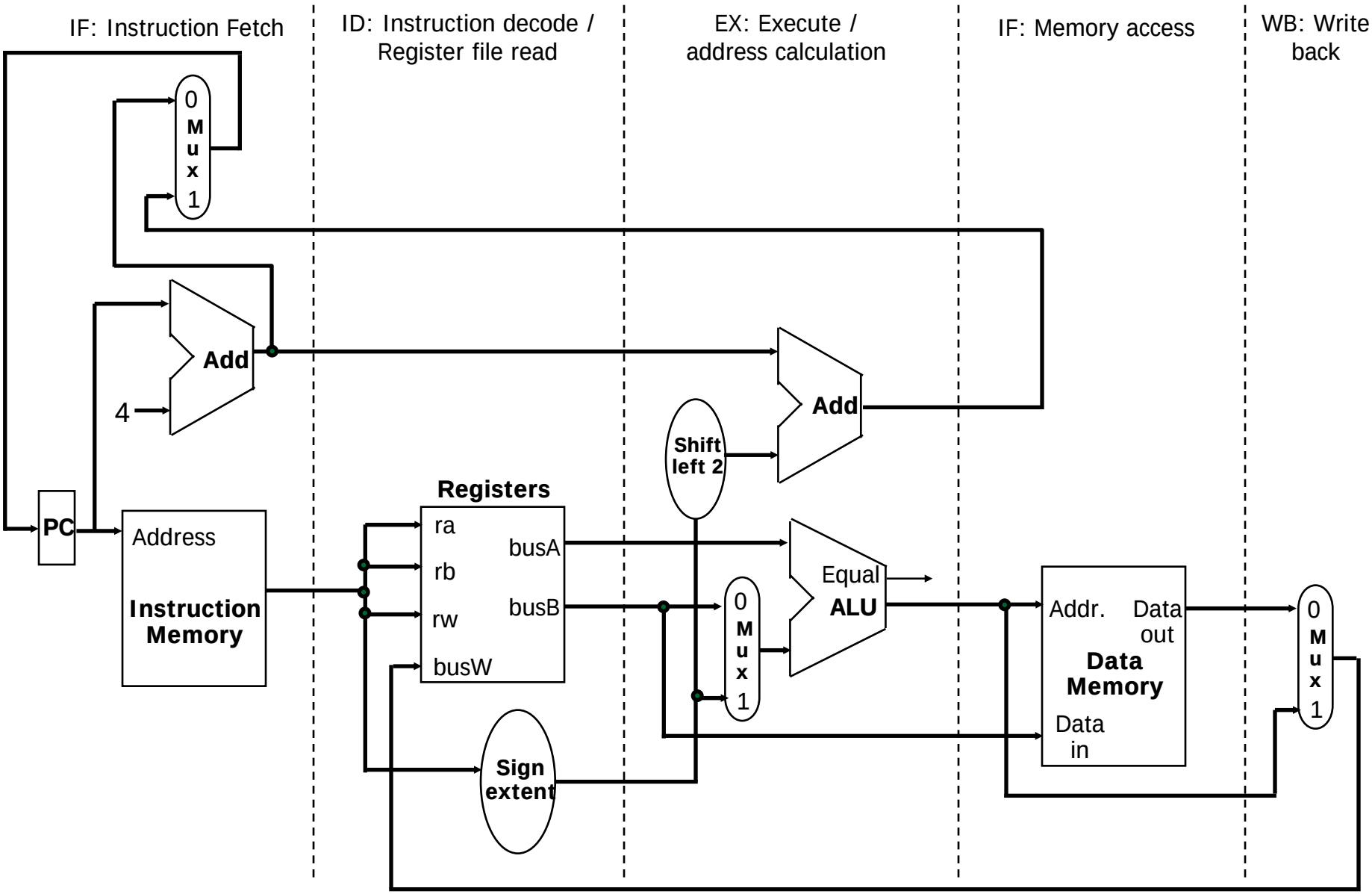
Single Cycle, Multiple Cycle, vs. Pipeline



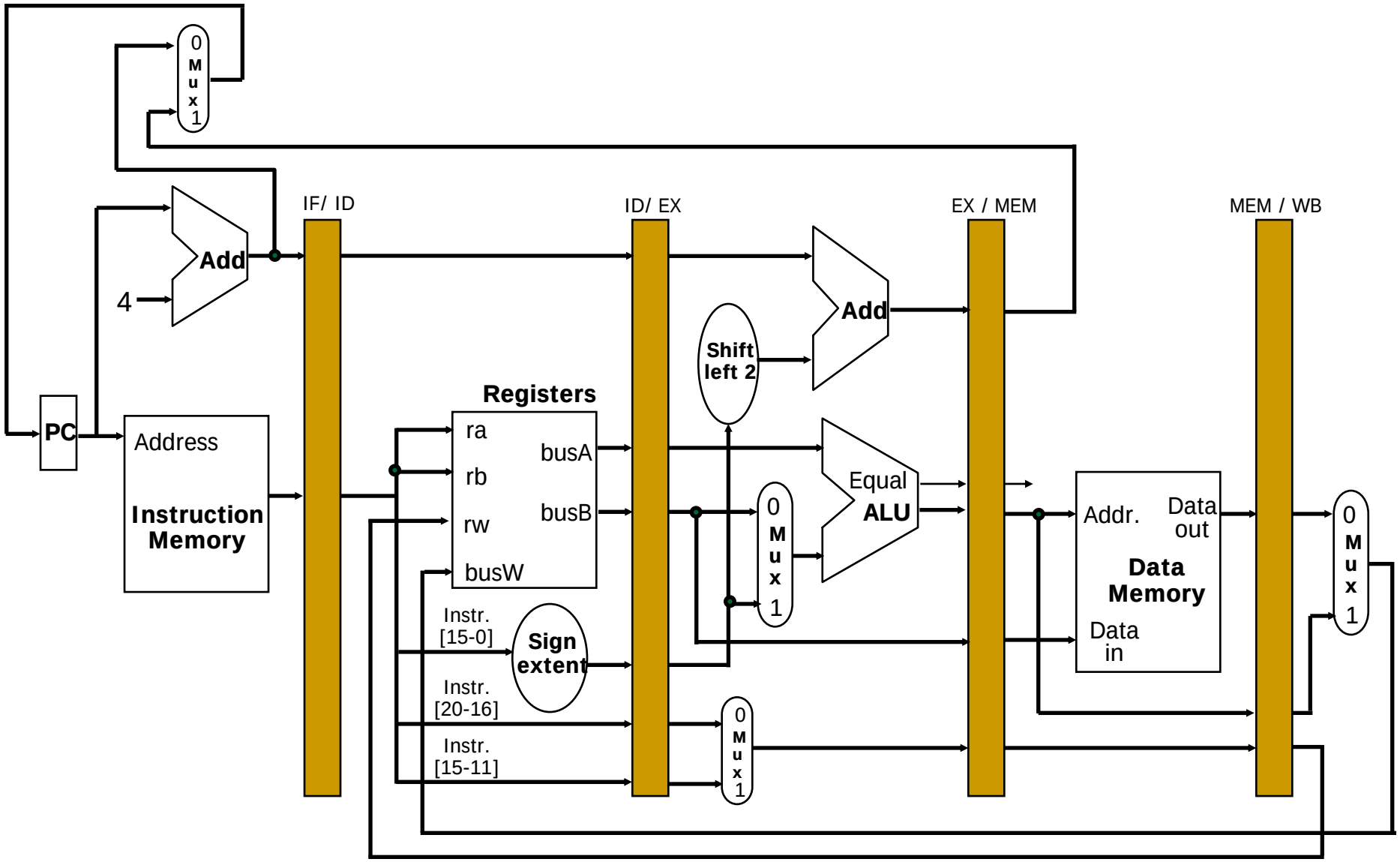
Why Pipeline?

- Suppose
  - 100 instructions are executed
  - The single cycle machine has a cycle time of 45 ns
  - The multicycle and pipeline machines have cycle times of 10 ns
  - The multicycle machine has a CPI of 3.6
- Single Cycle Machine
  - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
- Multicycle Machine
  - 10 ns/cycle x 3.6 CPI x 100 inst = 3600 ns
- Ideal pipelined machine
  - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns
- Ideal pipelined vs. single cycle speedup
  - 4500 ns / 1040 ns = 4.33
- What has not yet been considered?

Single Cycle Datapath



Pipelined Version of the Datapath

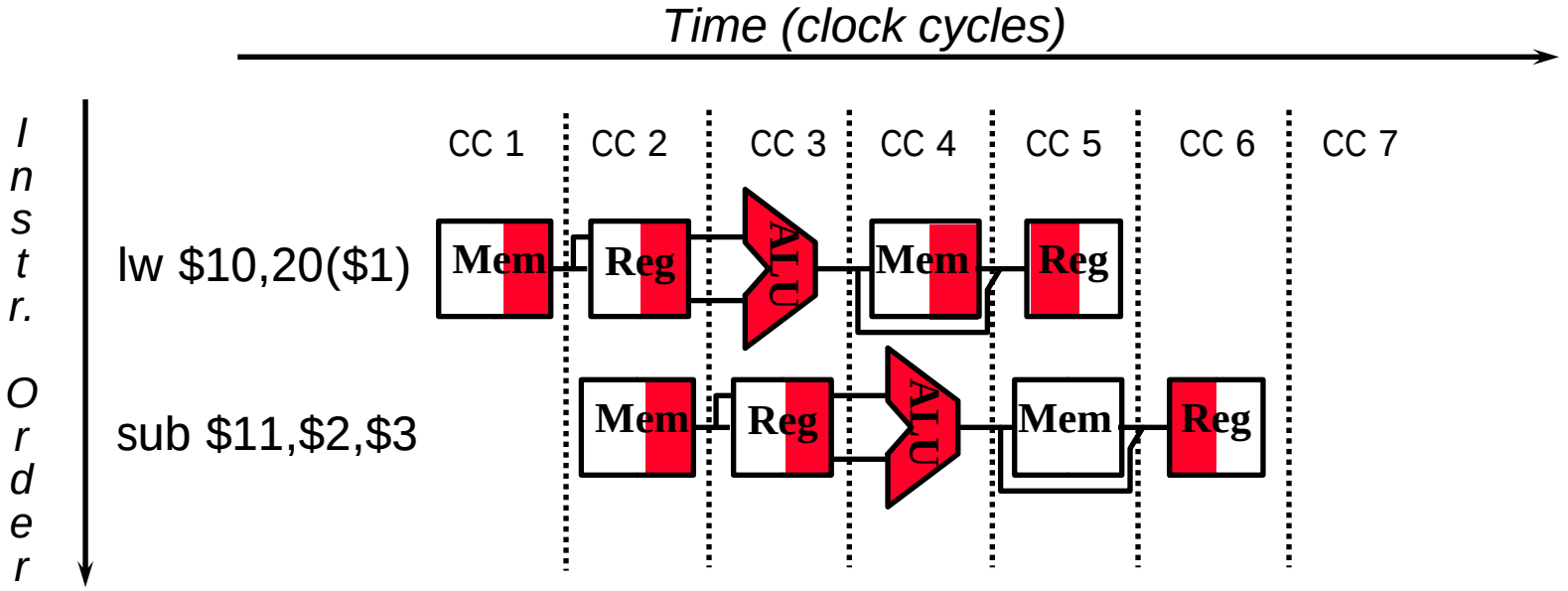


An Example to Clarify Pipelining

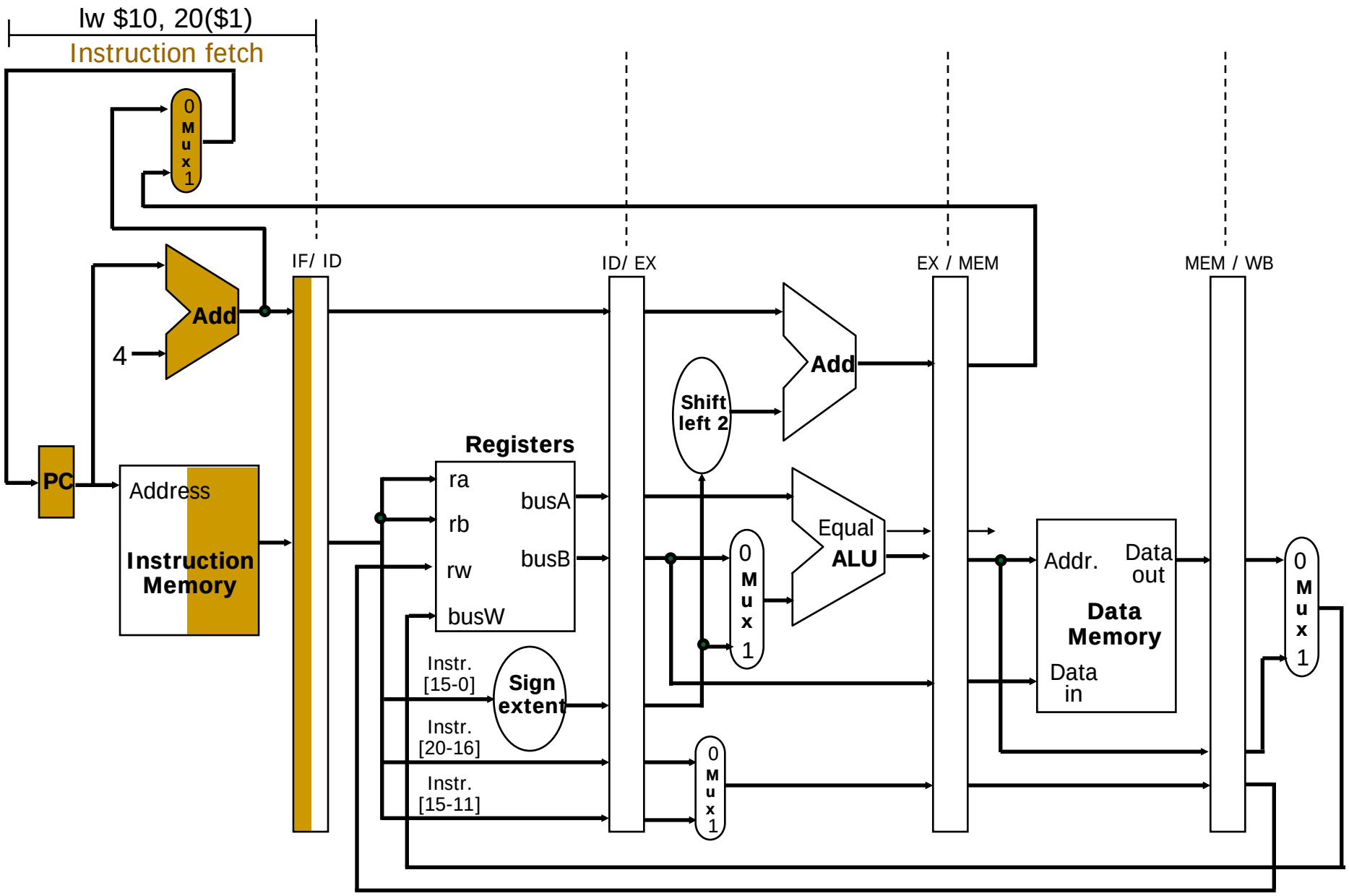
- Since many instructions are simultaneously are executing in a single cycle datapath, it can be difficult to understand.
- The following code will be examined:

  lw $10, 20($1)

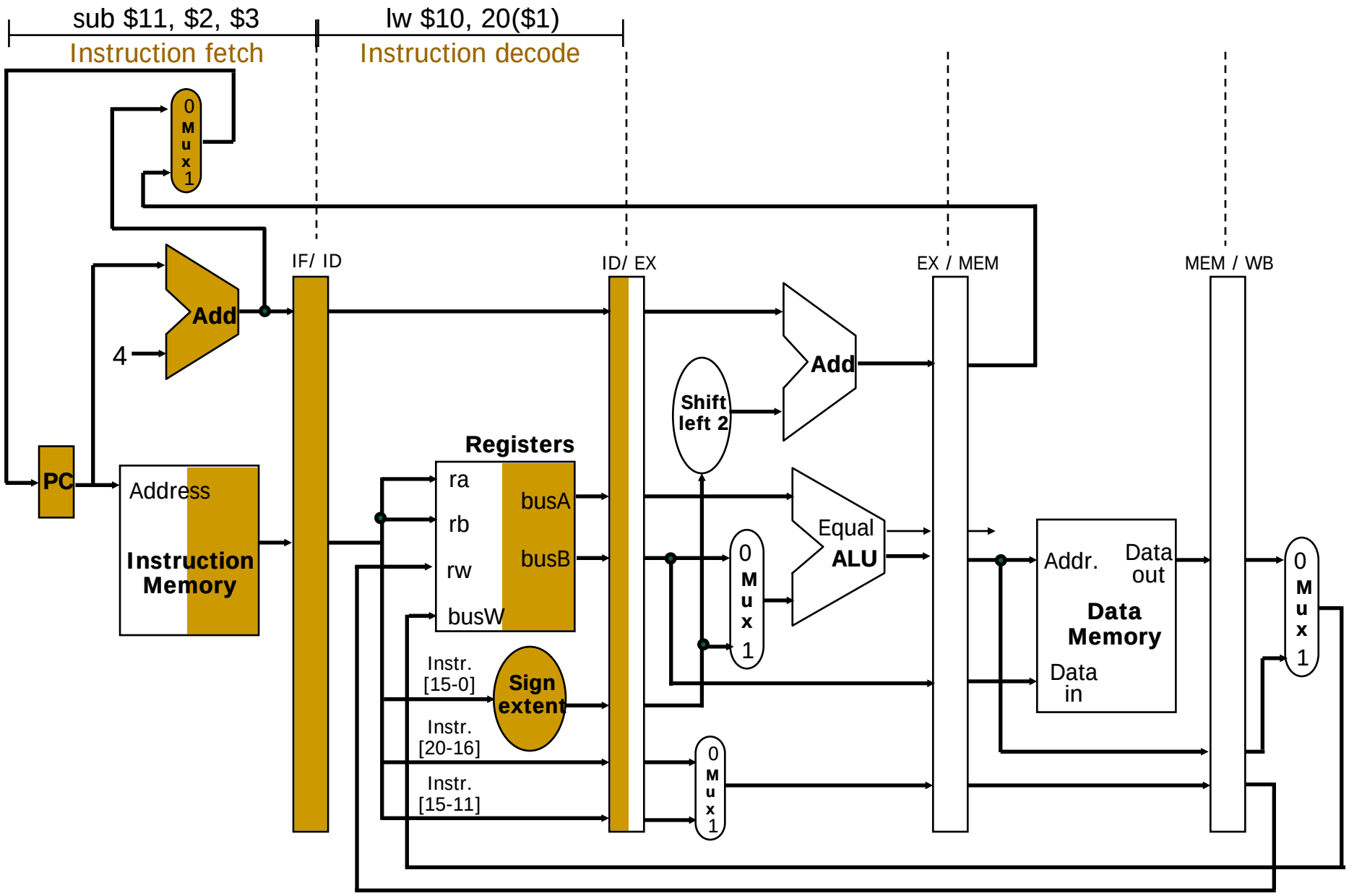
  sub $11, $2, $3



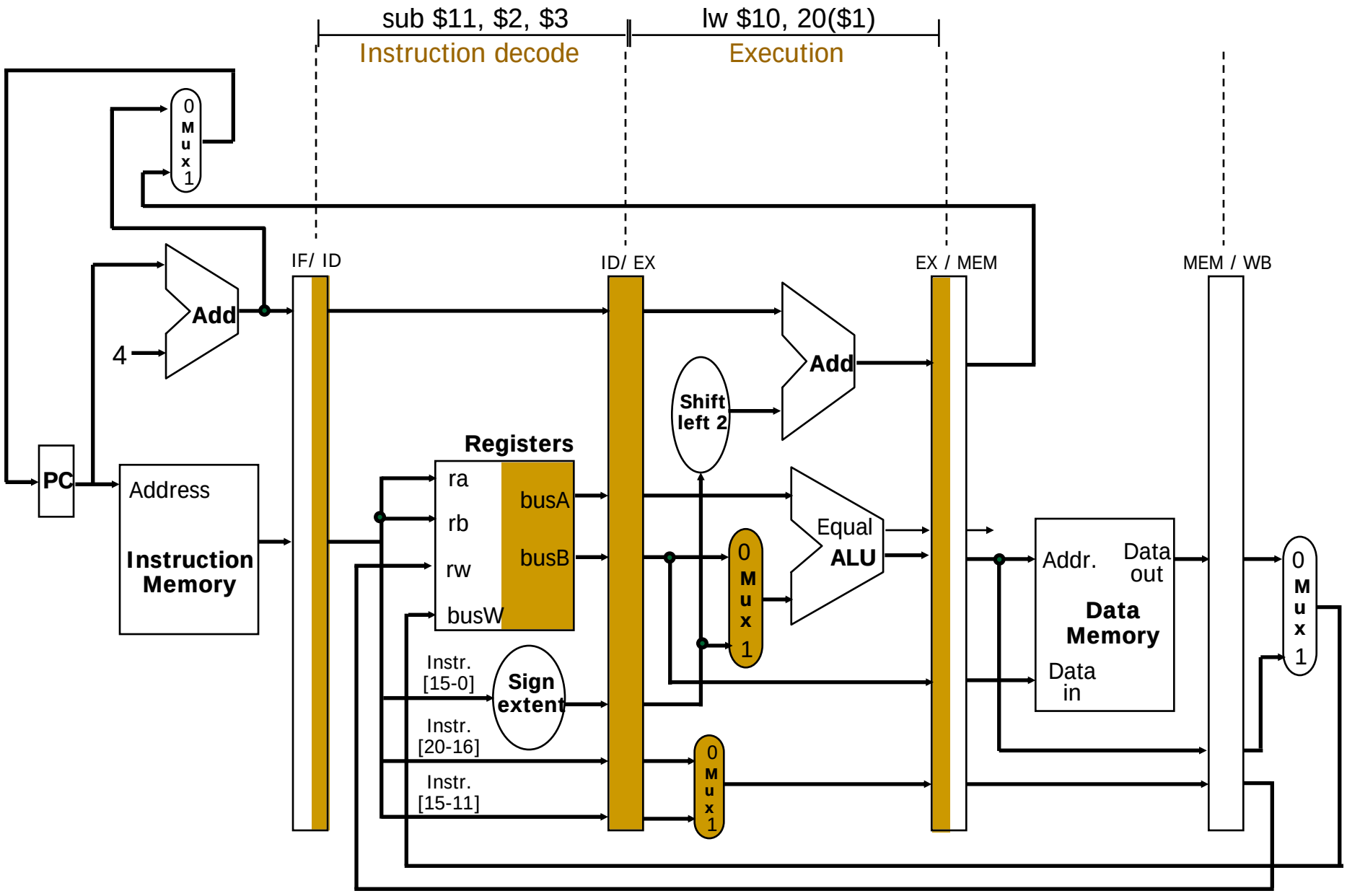
Clock 1


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

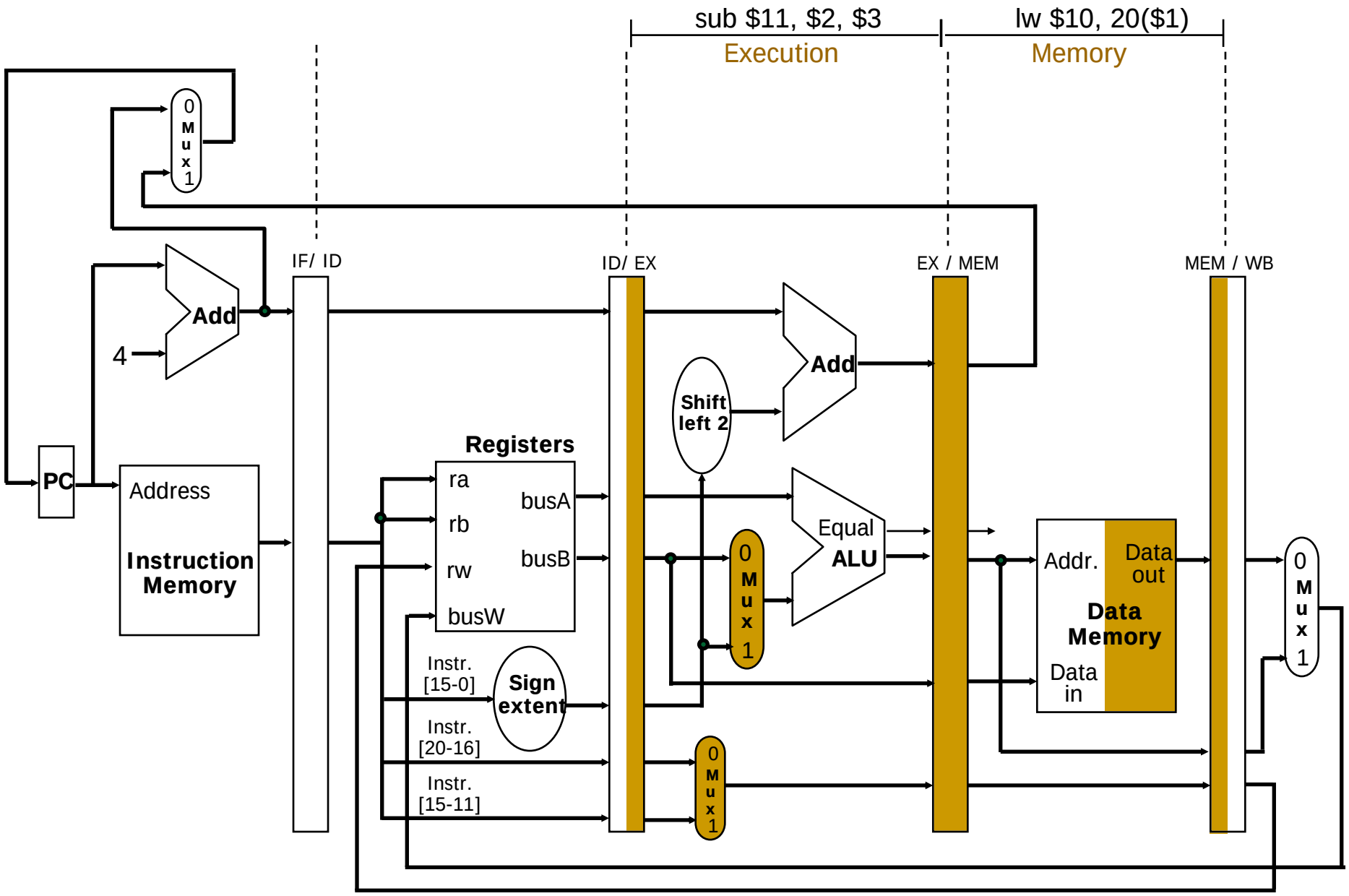
Clock 2


sub $11, $2, $3
lw $10, 20($1)
Instruction fetch
Instruction decode
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0
M
u
x
1
Add
4
PC
Address
Instruction Memory
Registers
ra
rb
rw
busW
busA
busB
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent
Shift left 2
Add
Equal
ALU
Addr.
Data out
Data Memory
Data in

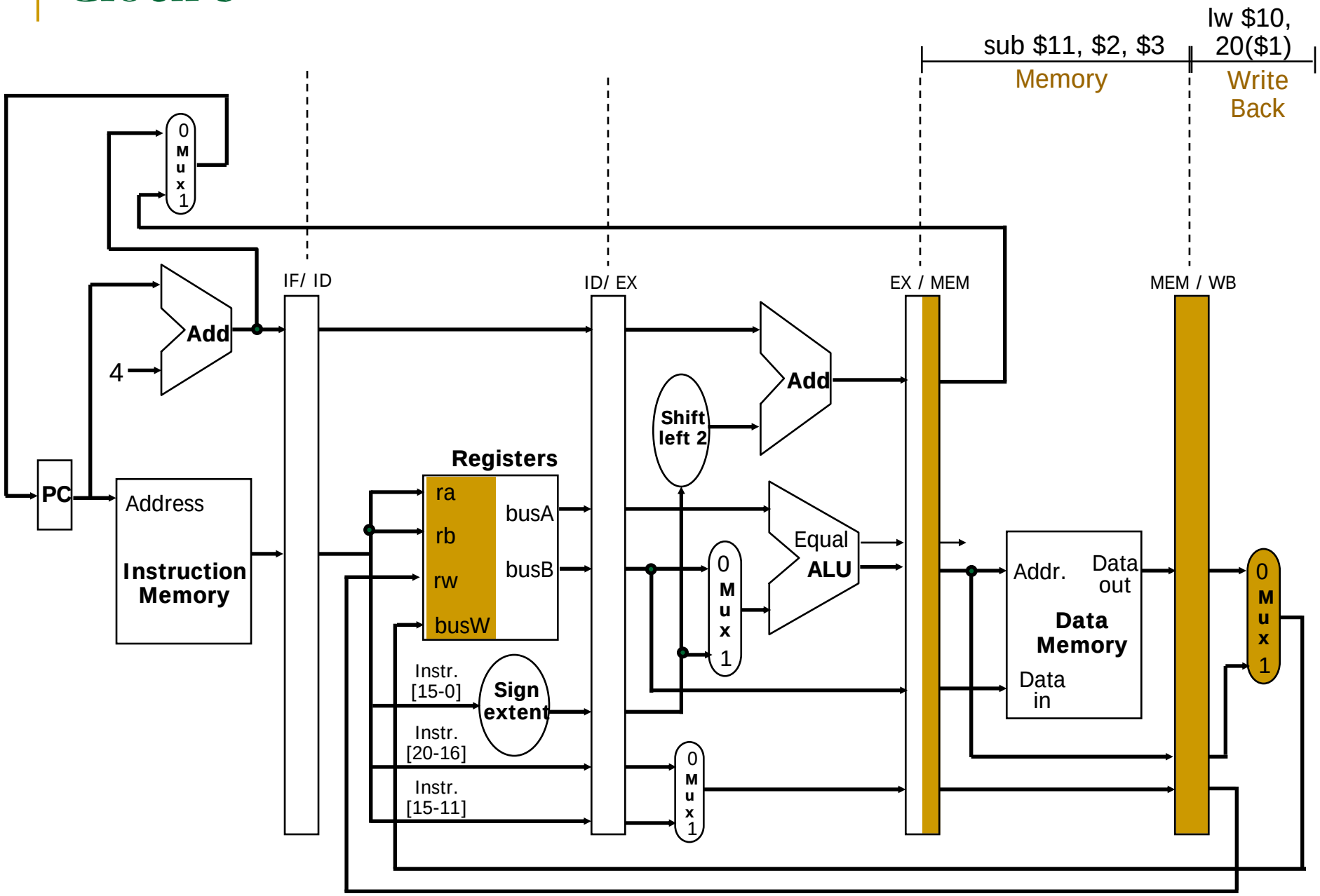
Clock 3


sub $11, $2, $3
Instruction decode
lw $10, 20($1)
Execution
IF/ ID
ID/ EX
EX / MEM
MEM / WB
Add
4
PC
Address
Instruction Memory
Registers
ra
rb
rw
busW
busA
busB
Shift left 2
Add
Equal
ALU
Addr.
Data out
Data Memory
Data in
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent
M u x
0
1

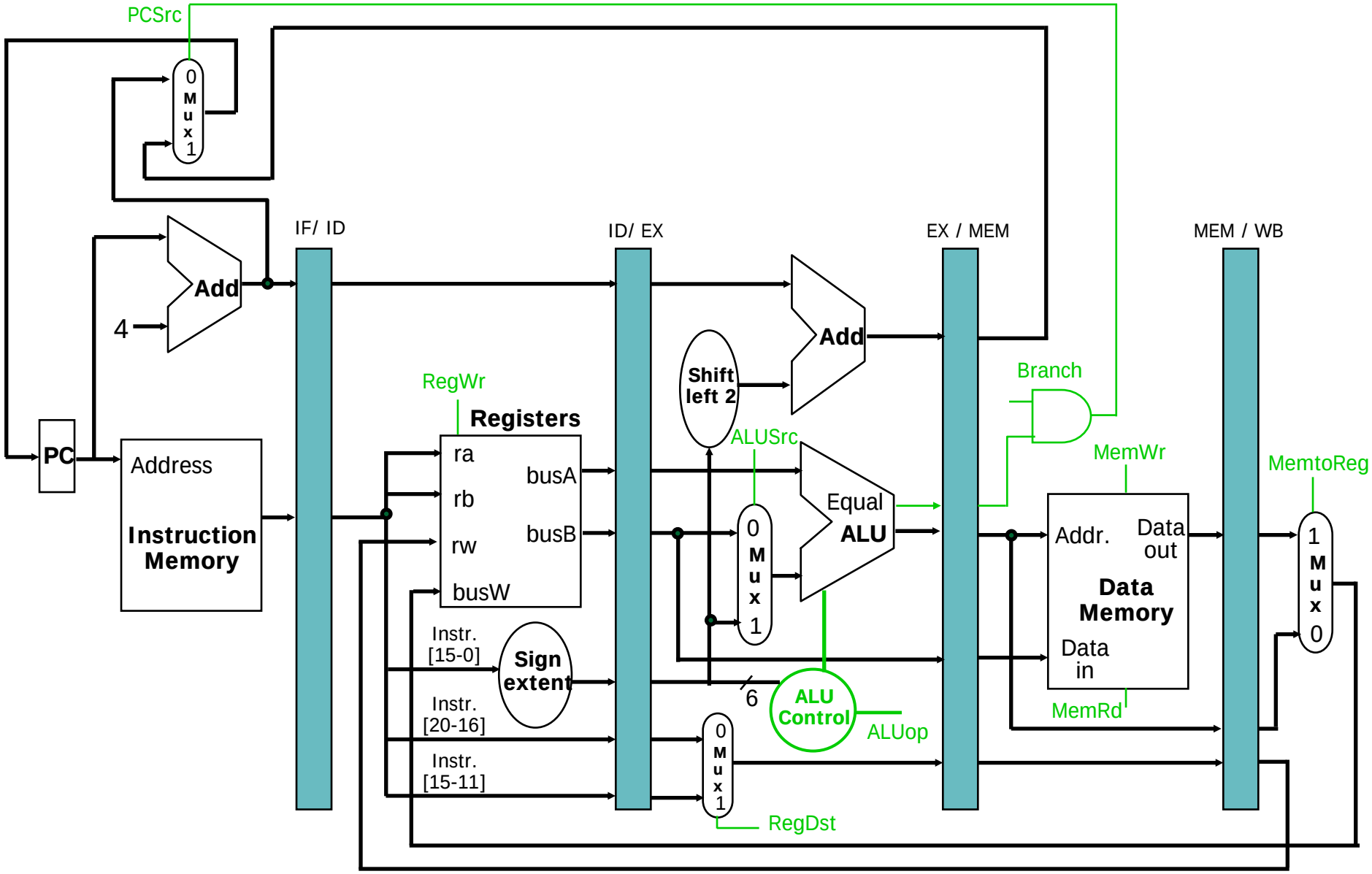
Clock 4


sub $11, $2, $3
Execution
lw $10, 20($1)
Memory
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0
M
u
x
1
Add
4
PC
Address
Instruction Memory
Registers
ra
rb
rw
busW
busA
busB
Shift left 2
Add
Equal
ALU
Addr.
Data out
Data Memory
Data in
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent

Clock 5


lw $10, 20($1)
sub $11, $2, $3
Memory
Write Back
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0 M u x 1
Add
4
PC
Address
Instruction Memory
Registers
ra
rb
rw
busW
busA
busB
Shift left 2
Add
Equal
ALU
Addr.
Data out
Data Memory
Data in
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent

Pipelined Datapath with Control Signals

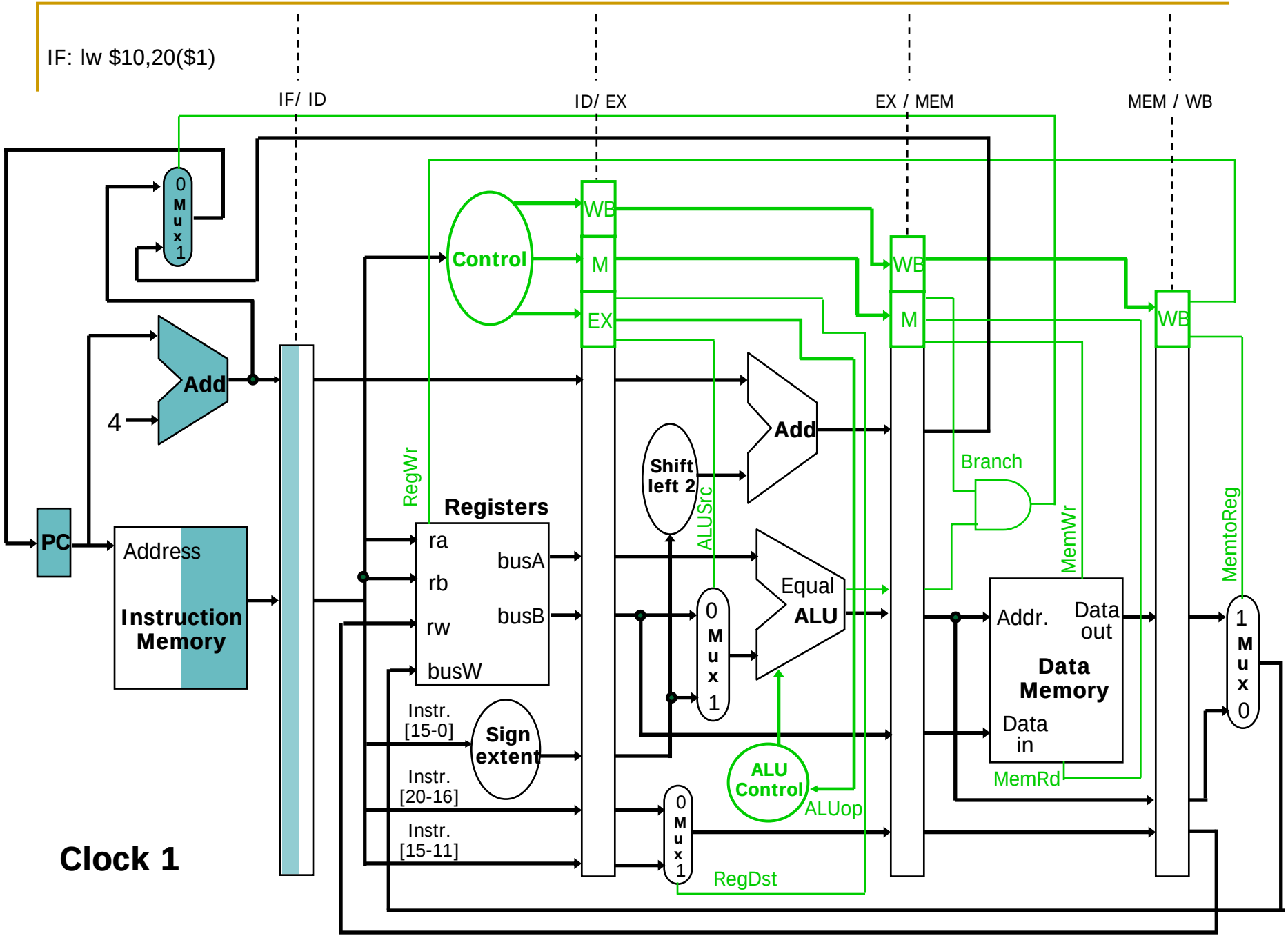


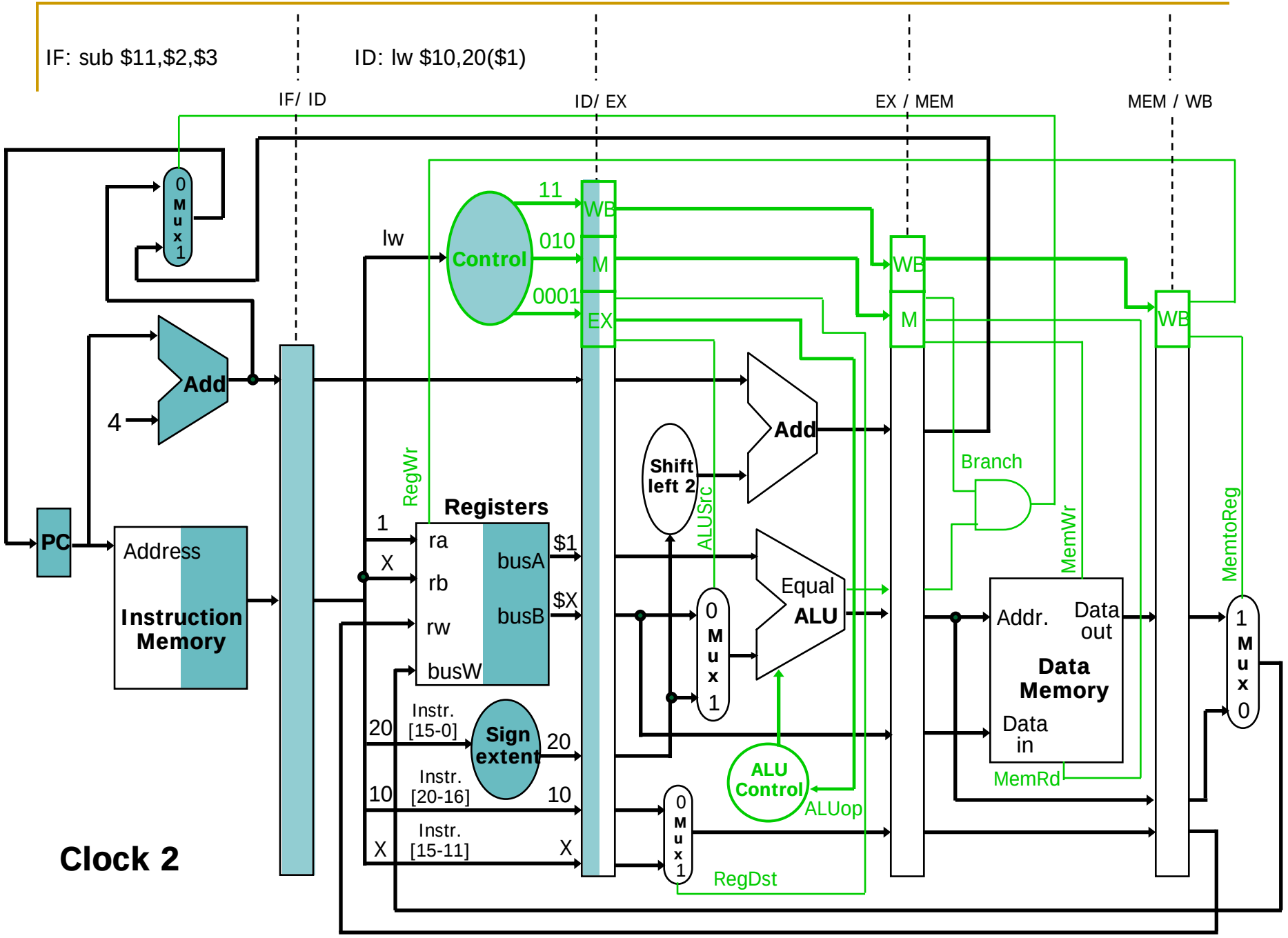
An Example to Clarify Pipelined Control

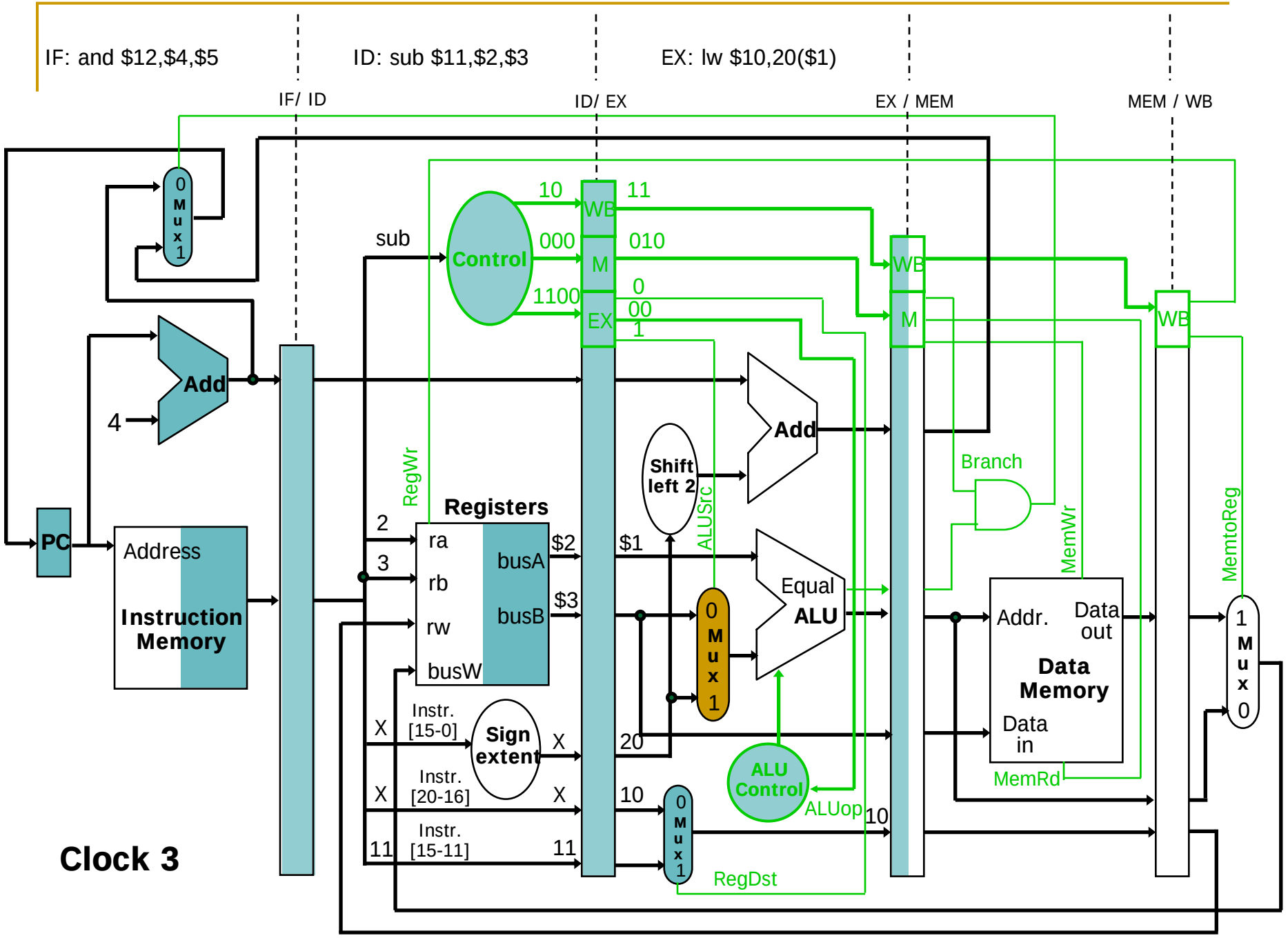
- Let's look at what's happing in the pipeline for the following program.

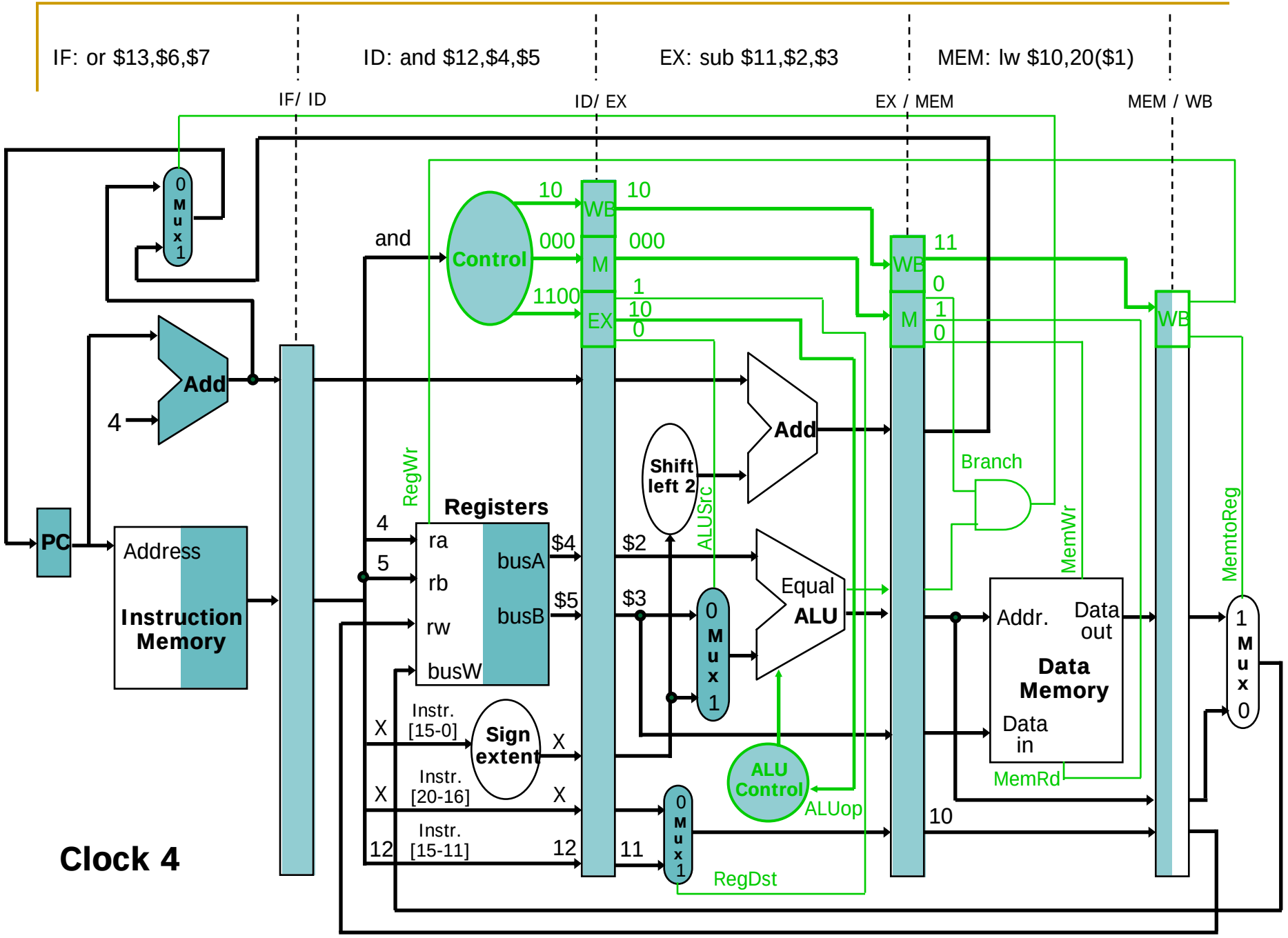
```
lw   $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or   $13, $6, $7
add $14, $8, $9
```

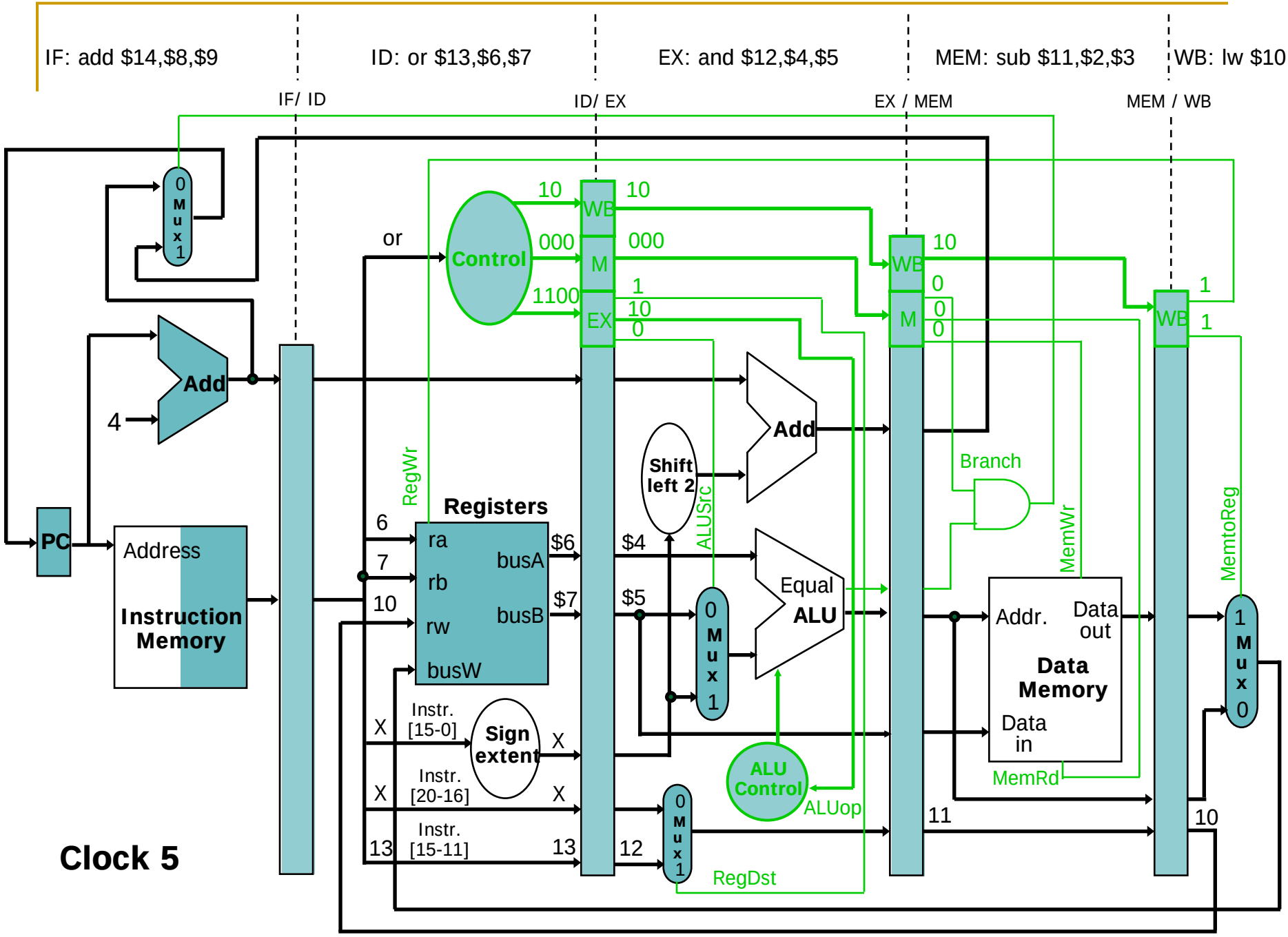
- Code does not have any data, control, or structural hazard.


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

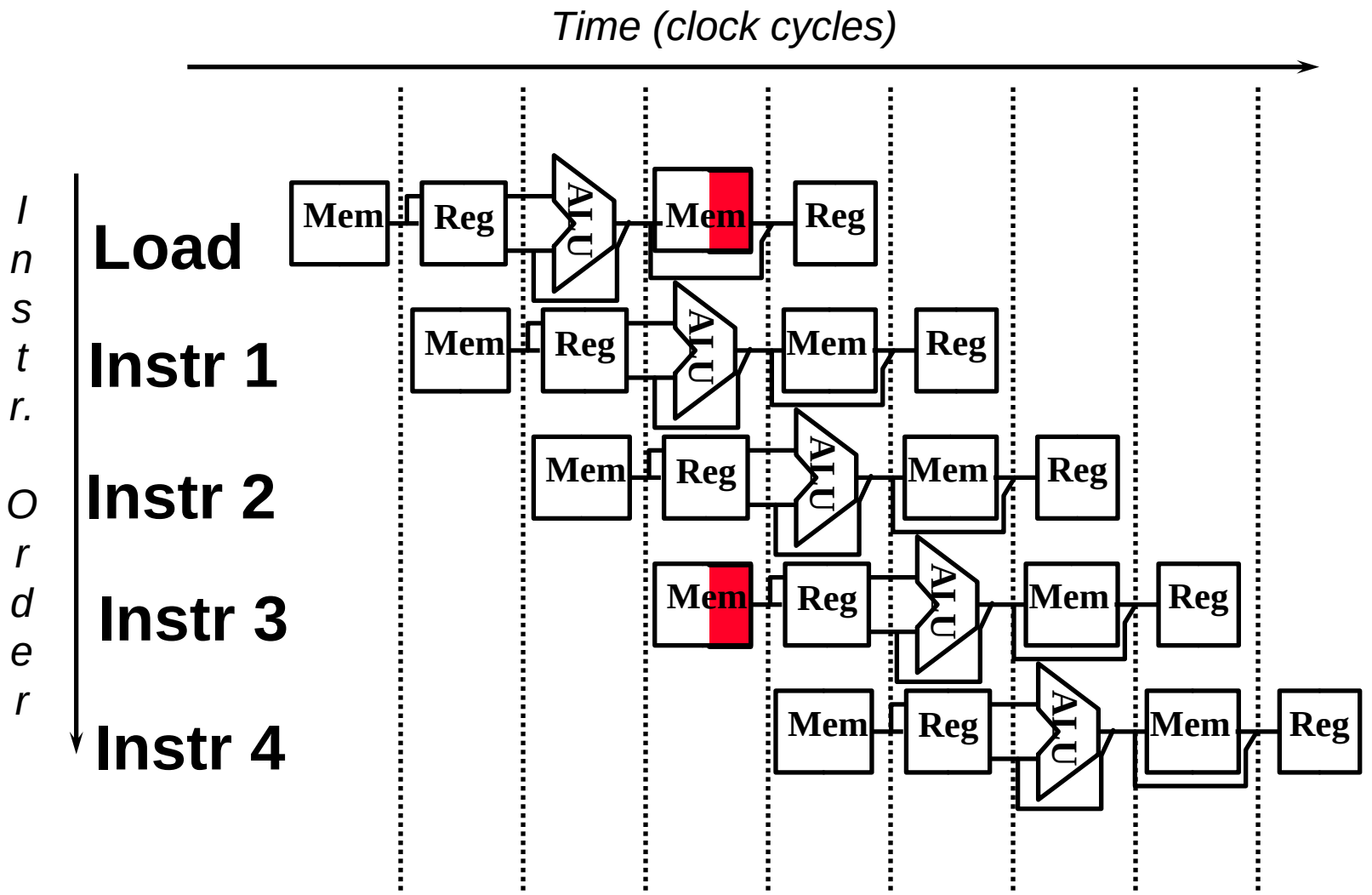

IF: sub $11,$2,$3
ID: lw $10,20($1)
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0
M
u
x
1
lw
Control
11
010
0001
WB
M
EX
WB
M
WB
Add
4
PC
Address
Instruction Memory
RegWr
Registers
1
X
ra
rb
rw
busW
busA
busB
$1
$X
Shift left 2
Add
ALUSrc
Equal
ALU
0
M
u
x
1
Branch
MemWr
MemtoReg
Addr.
Data out
Data Memory
Data in
MemRd
1
M
u
x
0
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
ALUSrc
Registers
2
3
ra
rb
rw
busW
busA
busB
$2
$3
$1
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
X
Instr. [20-16]
X
X
Instr. [15-11]
11
11
20
10
0 M u x 1
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
Clock 4
PC
Address
Instruction Memory
Add
4
0 M u x 1
and
Control
10
000
1100
WB
M
EX
10
000
1
10
0
11
0
1
0
Registers
ra
rb
rw
busW
busA
busB
4
5
$4
$5
RegWr
Instr. [15-0]
Instr. [20-16]
Instr. [15-11]
Sign extent
X
12
Shift left 2
$2
$3
ALUSrc
Equal
ALU
ALU Control
ALUop
RegDst
11
Branch
MemWr
MemRd
MemtoReg
Addr.
Data out
Data Memory
Data in
10
1 M u x 0

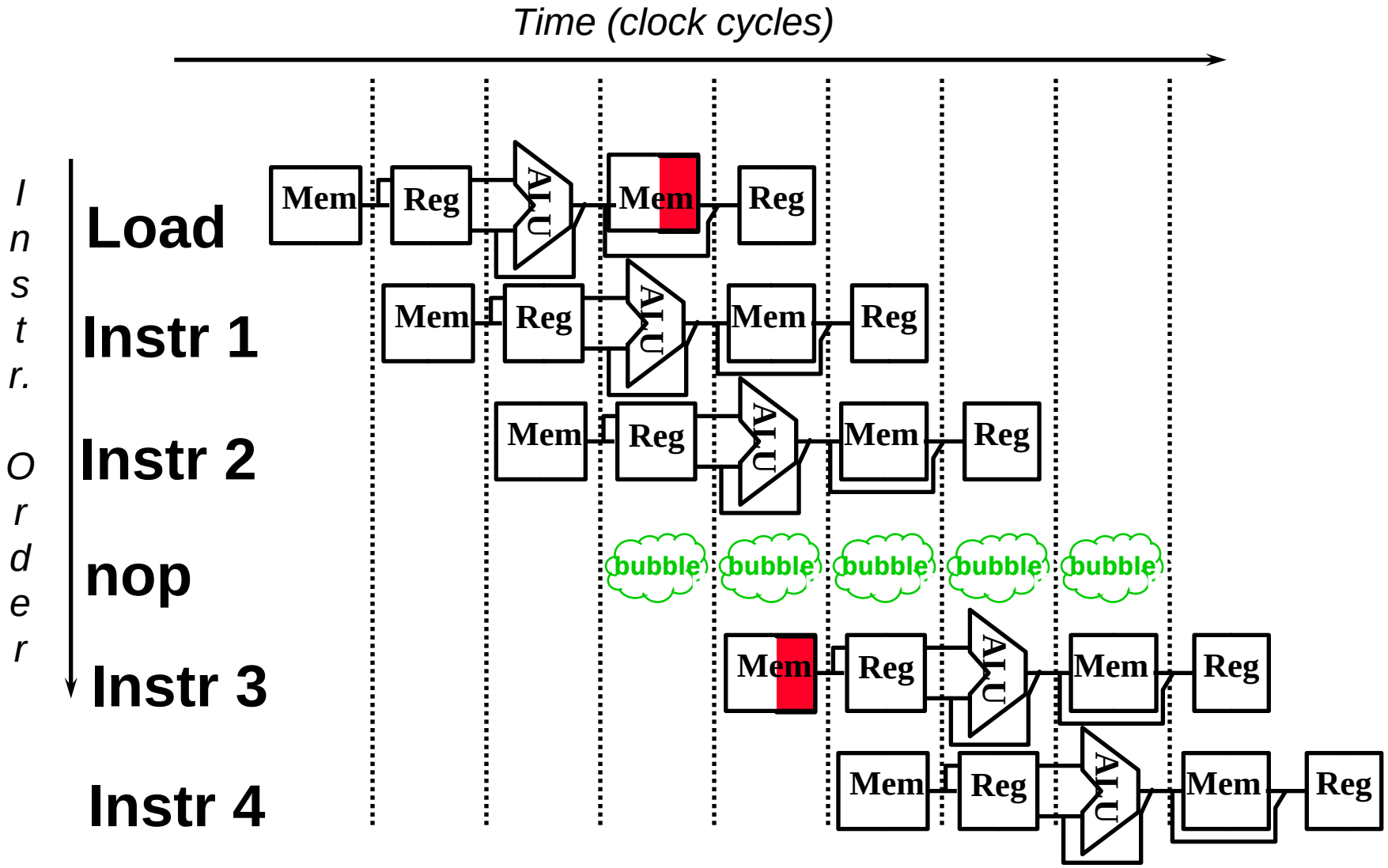

IF: add $14,$8,$9
ID: or $13,$6,$7
EX: and $12,$4,$5
MEM: sub $11,$2,$3
WB: lw $10...
IF/ ID
ID/ EX
EX / MEM
MEM / WB
0 M u x 1
Add
4
PC
Address
Instruction Memory
or
Control
10
000
1100
WB
M
EX
10
000
1
10
0
WB
M
WB
10
0
0
1
1
Add
Shift left 2
RegWr
ALUSrc
Branch
MemWr
MemtoReg
Registers
6
7
10
ra
rb
rw
busW
busA
busB
$6
$7
$4
$5
Equal
ALU
0 M u x 1
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
13
Instr. [15-11]
13
12
ALU Control
ALUop
11
10
0 M u x 1
RegDst
Clock 5

Single Memory is a Structural Hazard



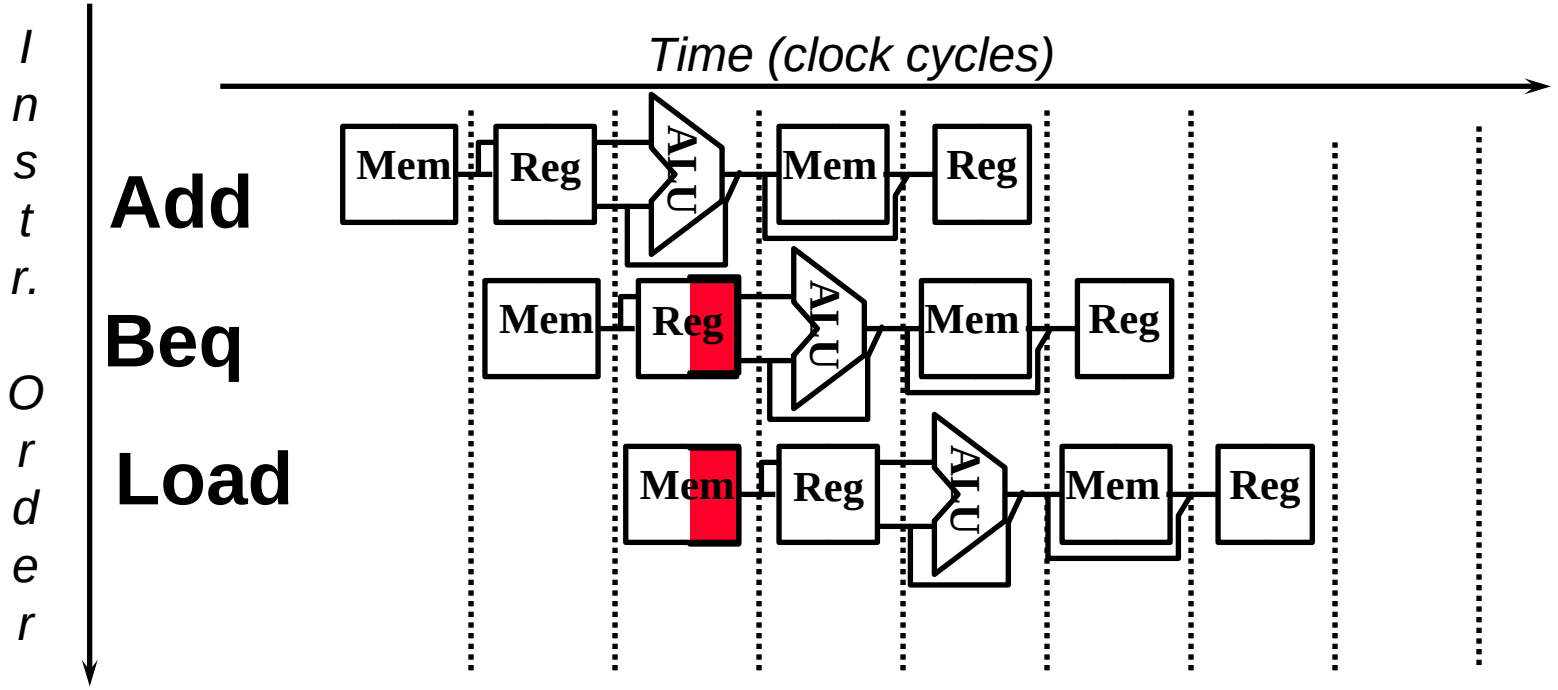
Detection is easy in this case! (right half highlight means read, left half write)

Solution 3: Stall



Control Hazard Solutions

- Predict: guess one direction then back up if wrong
  - Predict not taken



- Impact: 1 clock cycle per branch instruction if right, 2 if wrong (right - 50% of time)
- More dynamic scheme: history of 1 branch (- 90%)

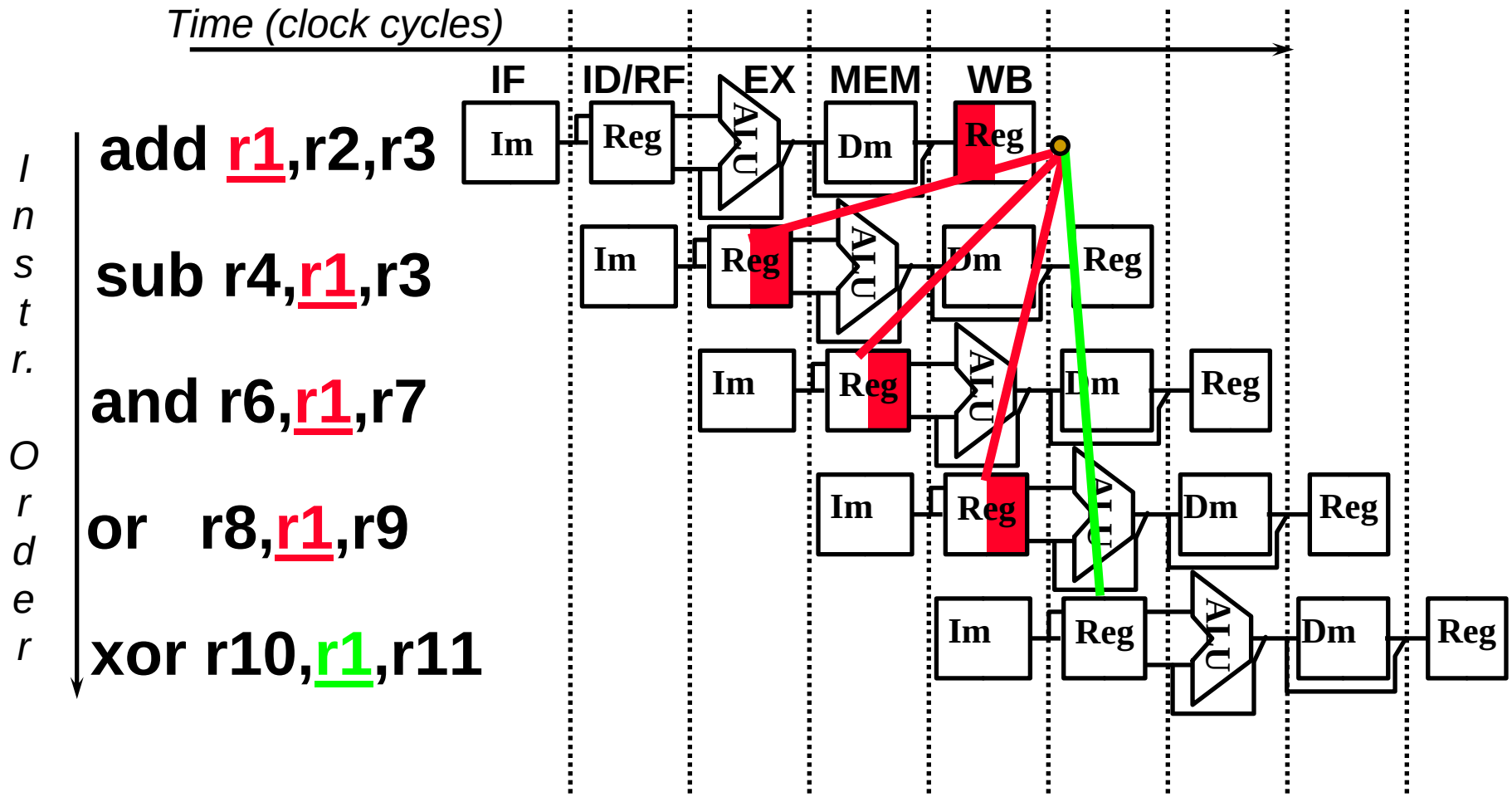
Data Hazard on r1

- Problem: r1 cannot be read by other instructions before it is written by the add.

```
add   r1 , r2, r3
sub   r4, r1 , r3
and   r6, r1 , r7
or    r8, r1 , r9
xor   r10, r1 , r11
```

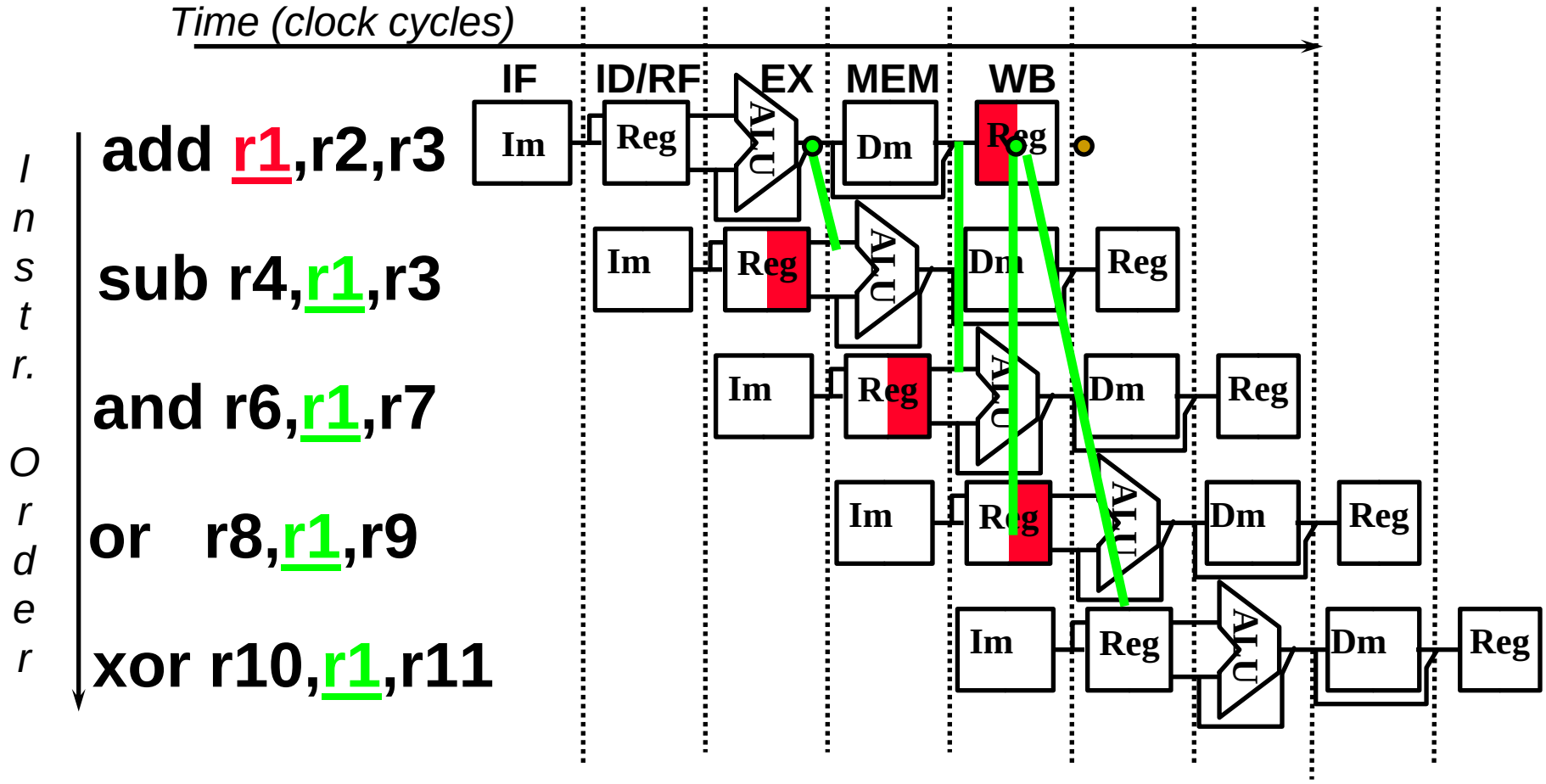
Data Hazard on r1:

- **Dependencies backwards in time are hazards**



Data Hazard Solution:

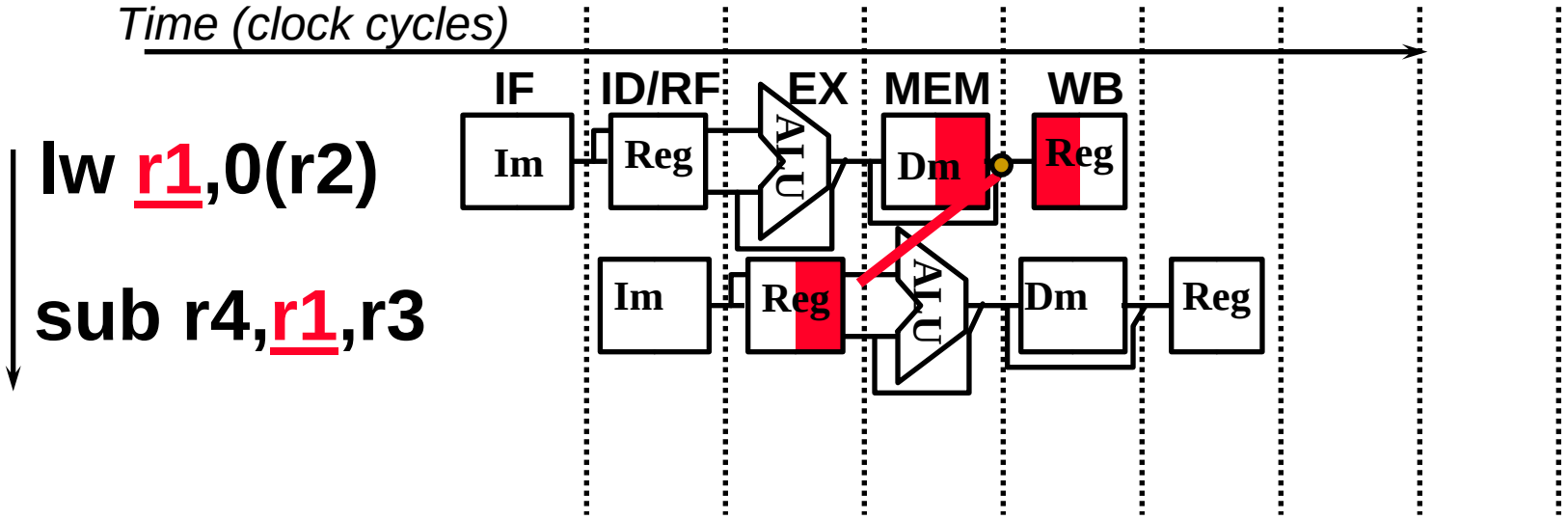
- **"Forward" result from one stage to another**



- **"or" instruction is OK if define read/write properly**

Forwarding (or Bypassing): What about Loads

- **Dependencies backwards in time are hazards**



- **Can't solve with forwarding:**
- **Must delay/stall instruction dependent on loads**

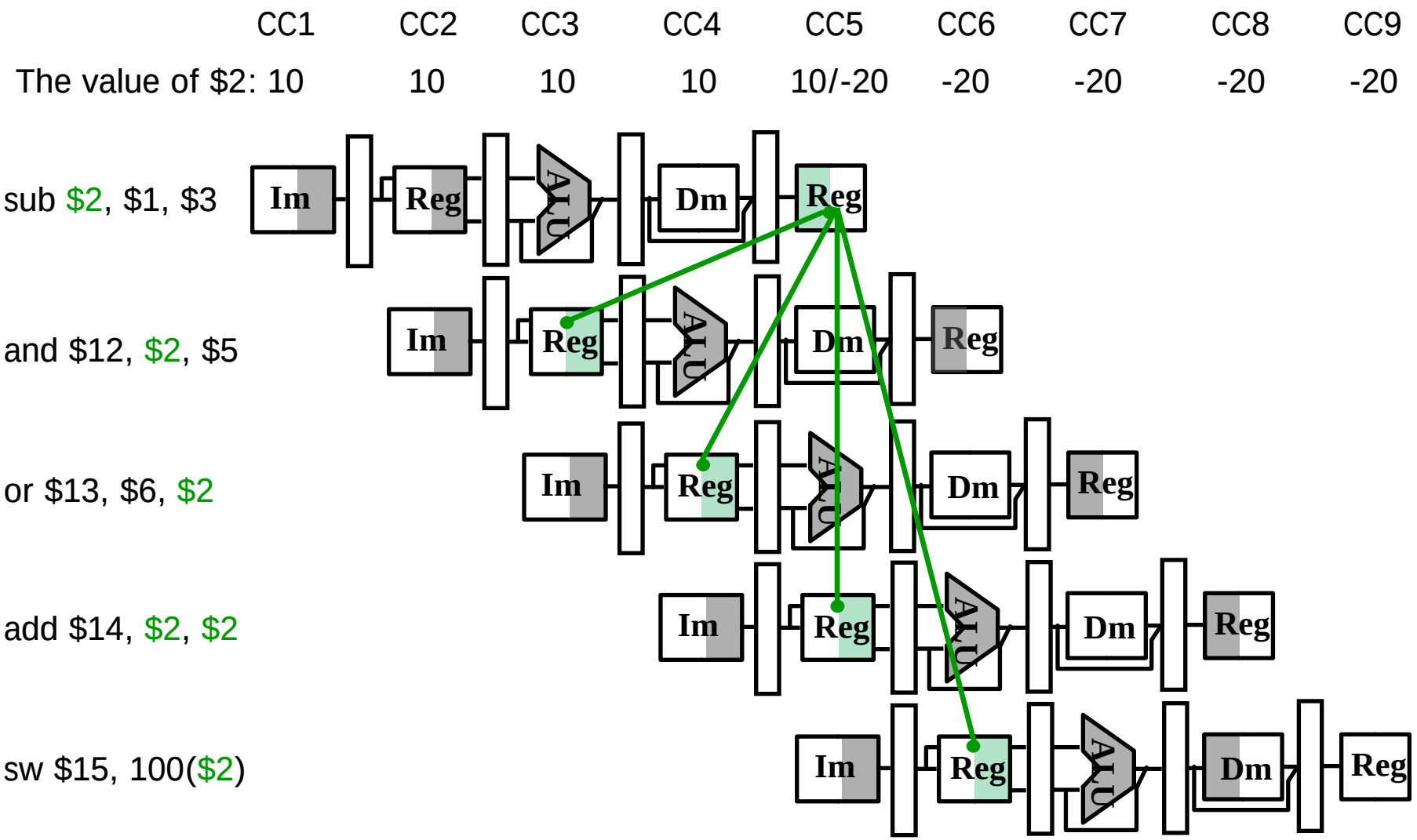
Data Hazards

- 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)
```

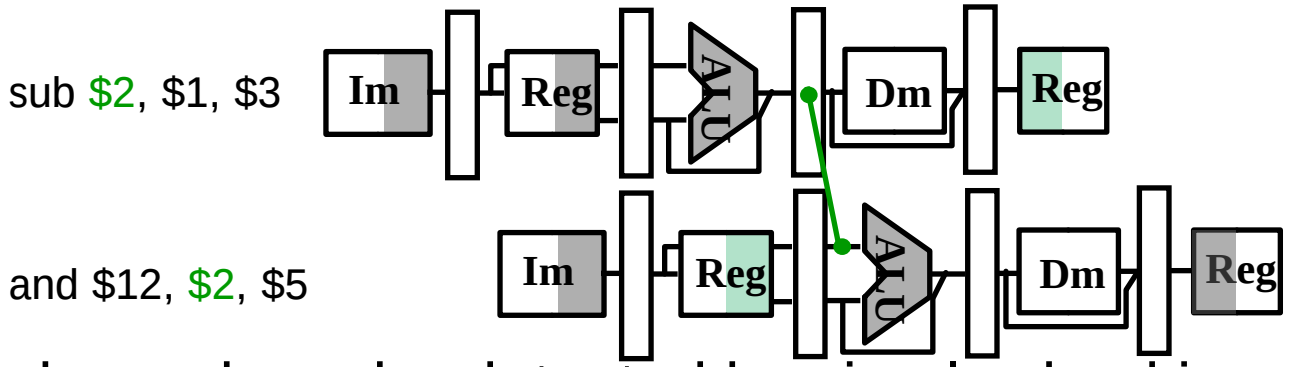
- The last four instructions are all dependent on the register $2 of the first instruction.
- Assume that register $2 had the value of 10 before the subtract instruction and -20 afterwards.

Data Hazards (Cont')



Data 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:

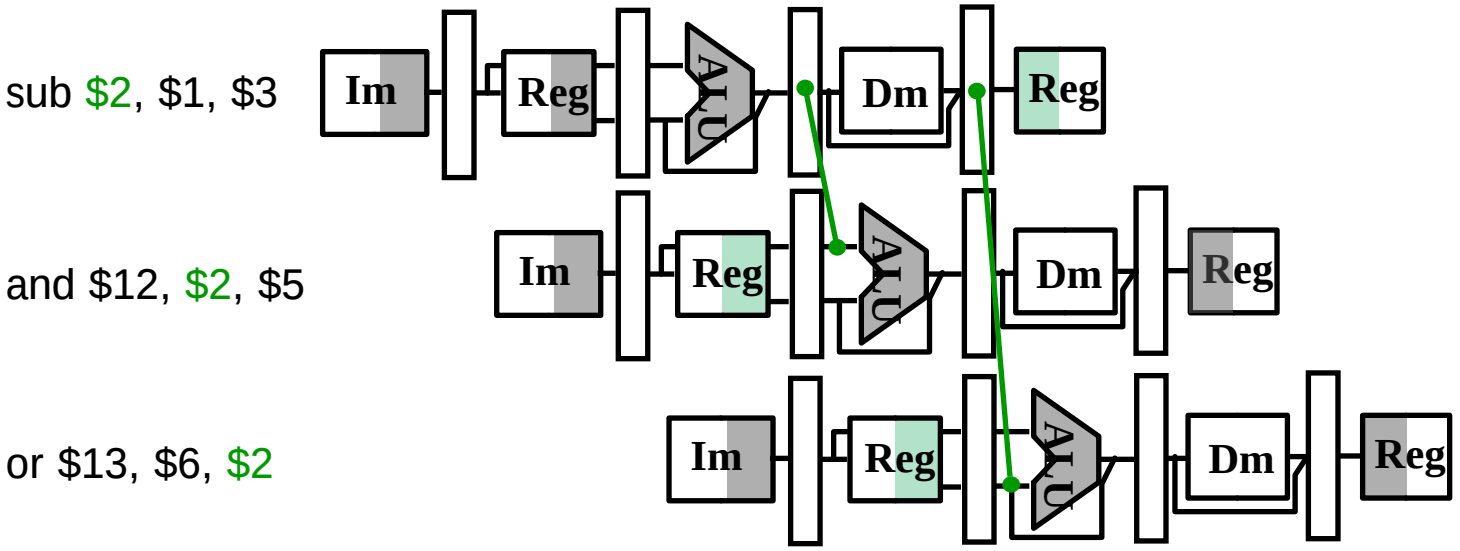


- This hazard can be detected by simply checking:

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.

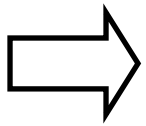
Summary of Data Hazard Conditions

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:

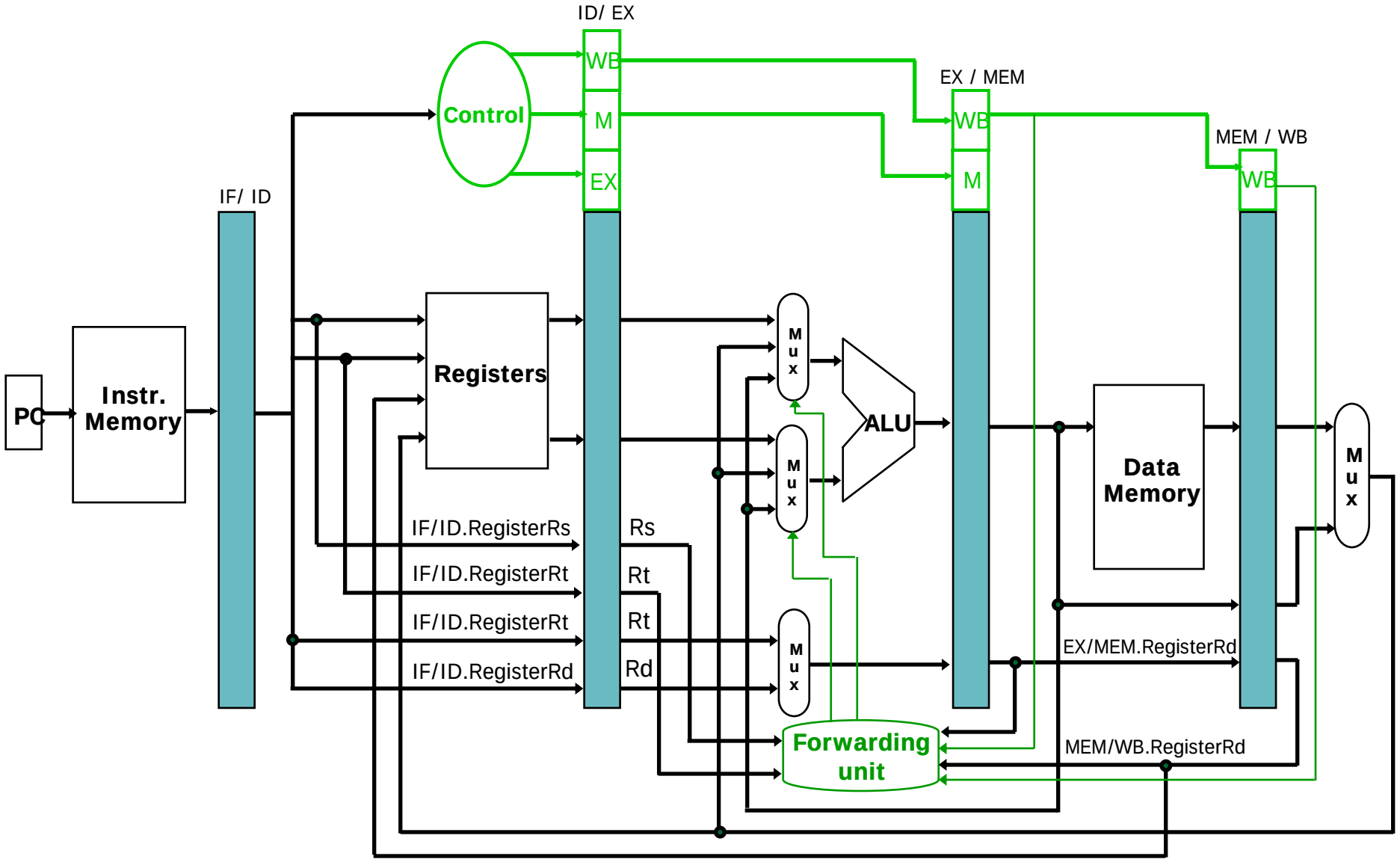
if (MEM/WB.RegWr
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA=01

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

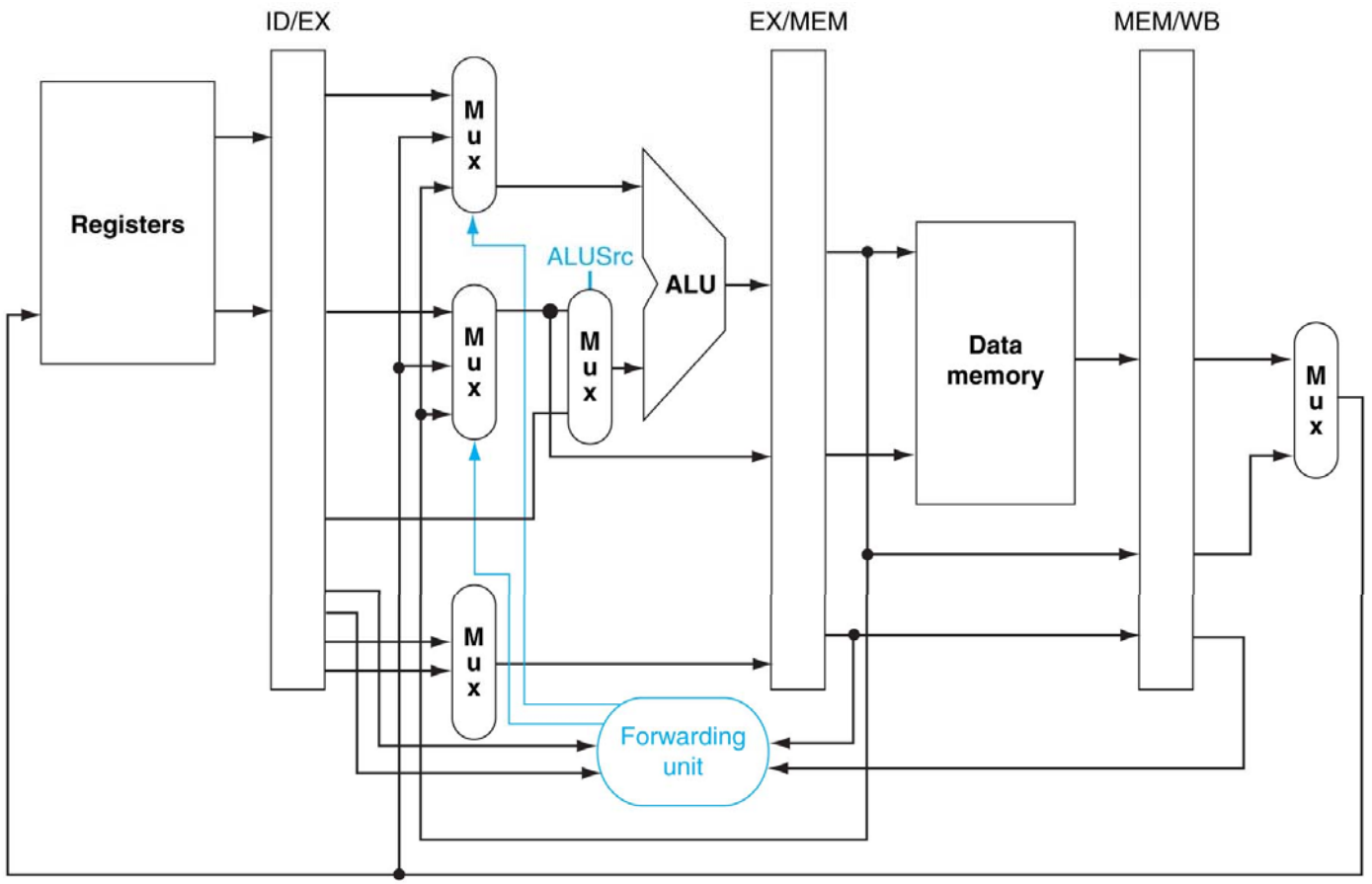
Summary of Data Hazard Conditions (Cont')

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.

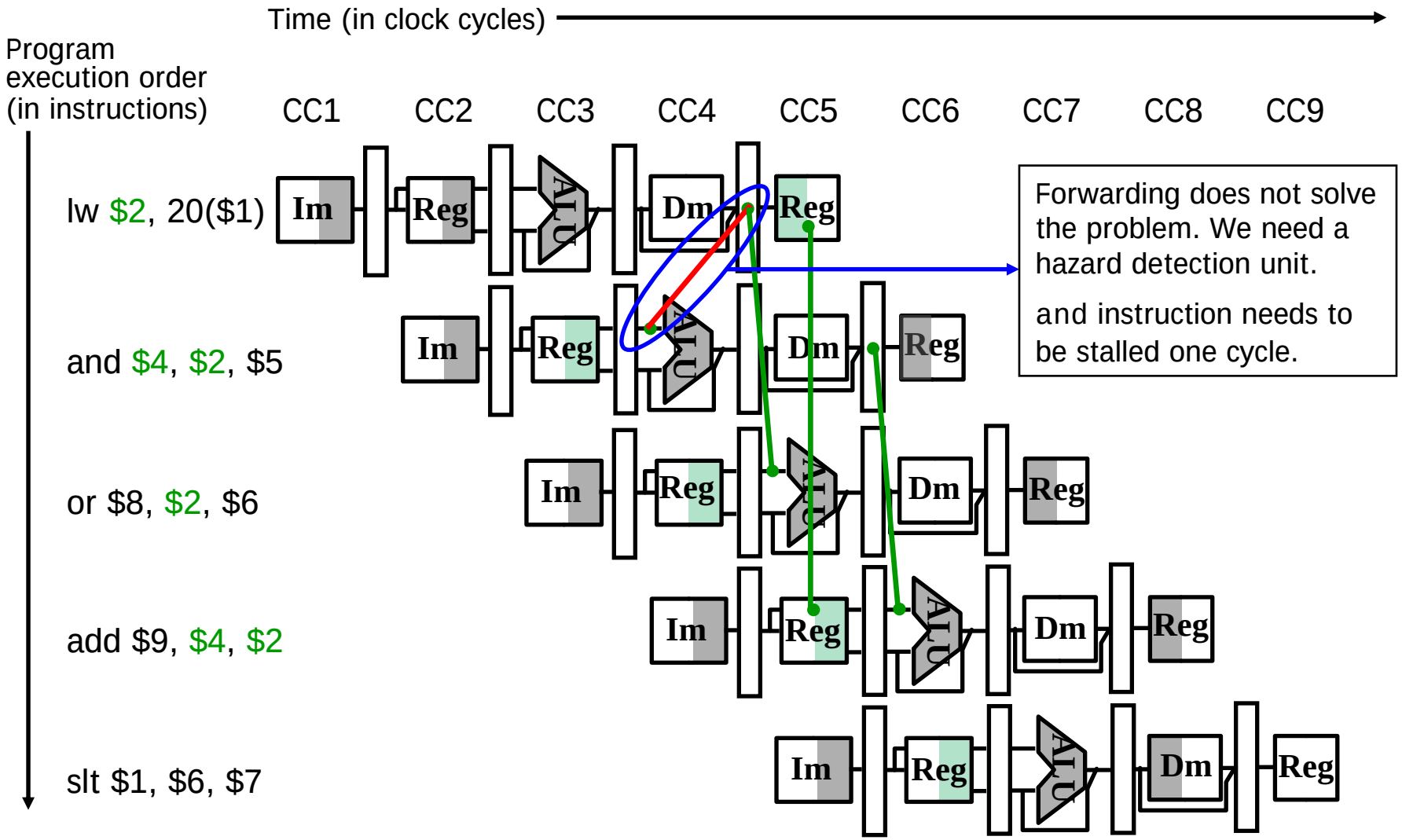
The Pipelined Datapath with Forwarding



Signed Immediate Enhancement



Data Hazards and Stalls



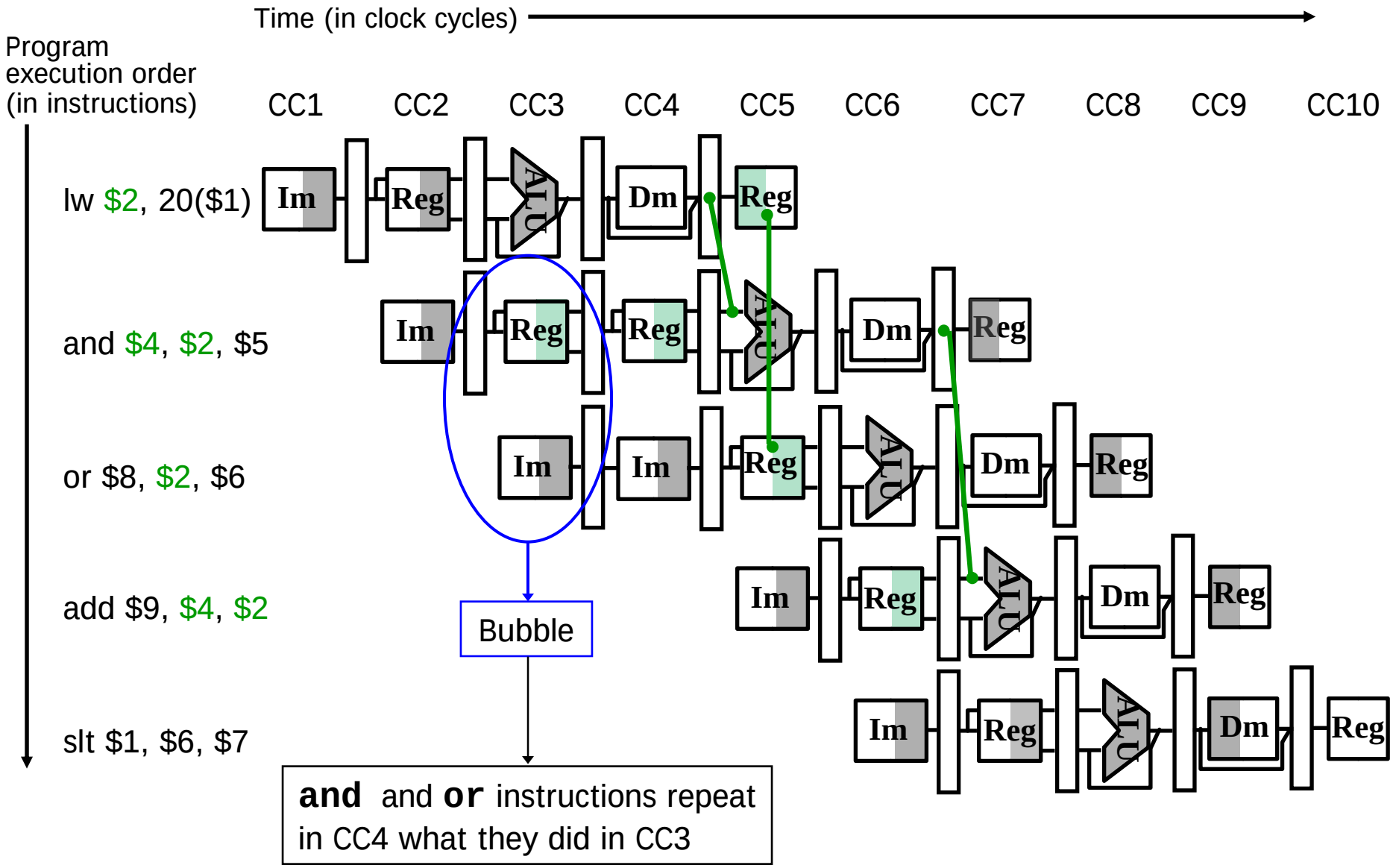
Hazard Detection

- The control for the hazard detection unit is:

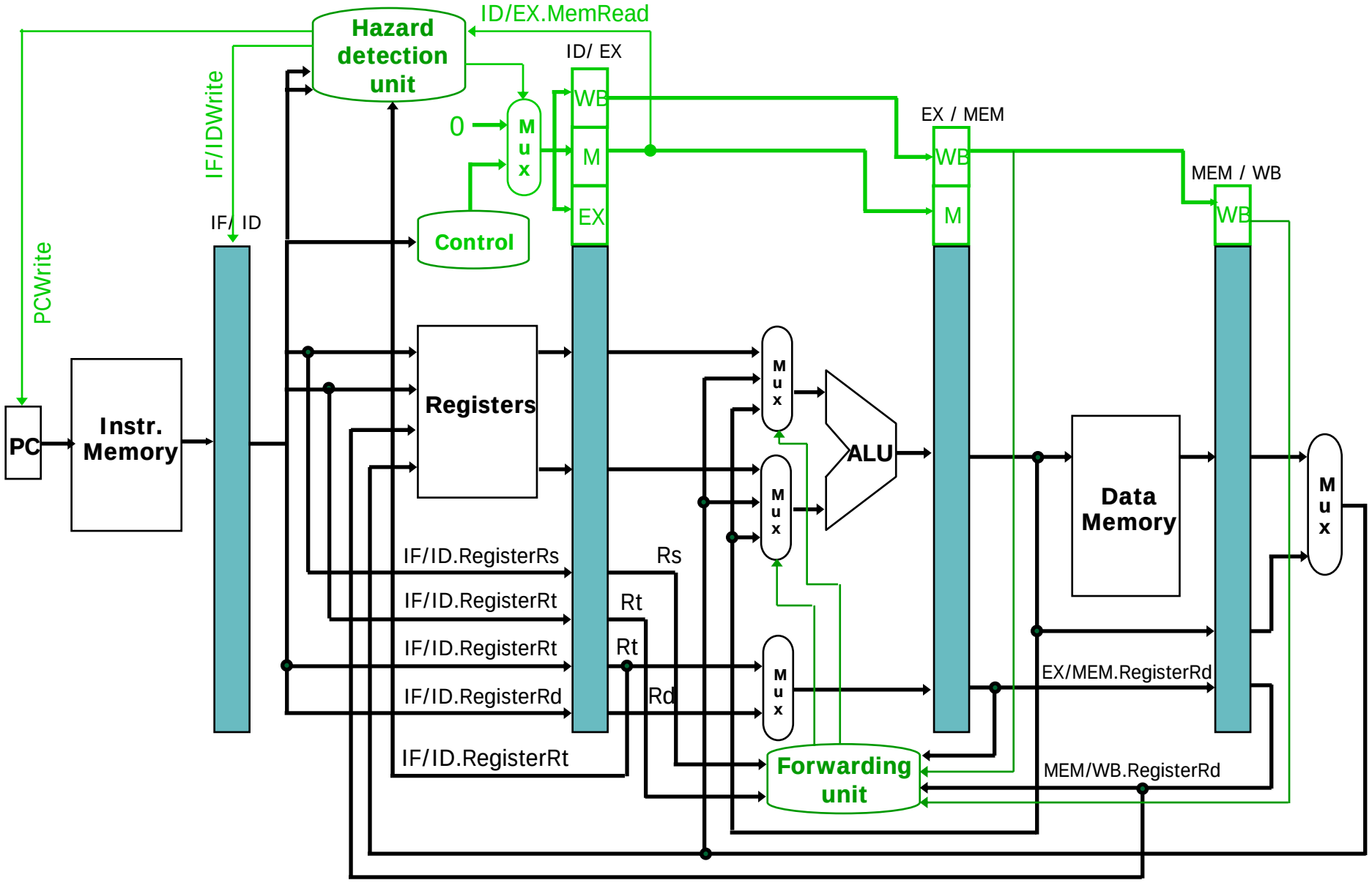
```
if (ID/EX.MemRd
  and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
```

Checks if the instruction is a load

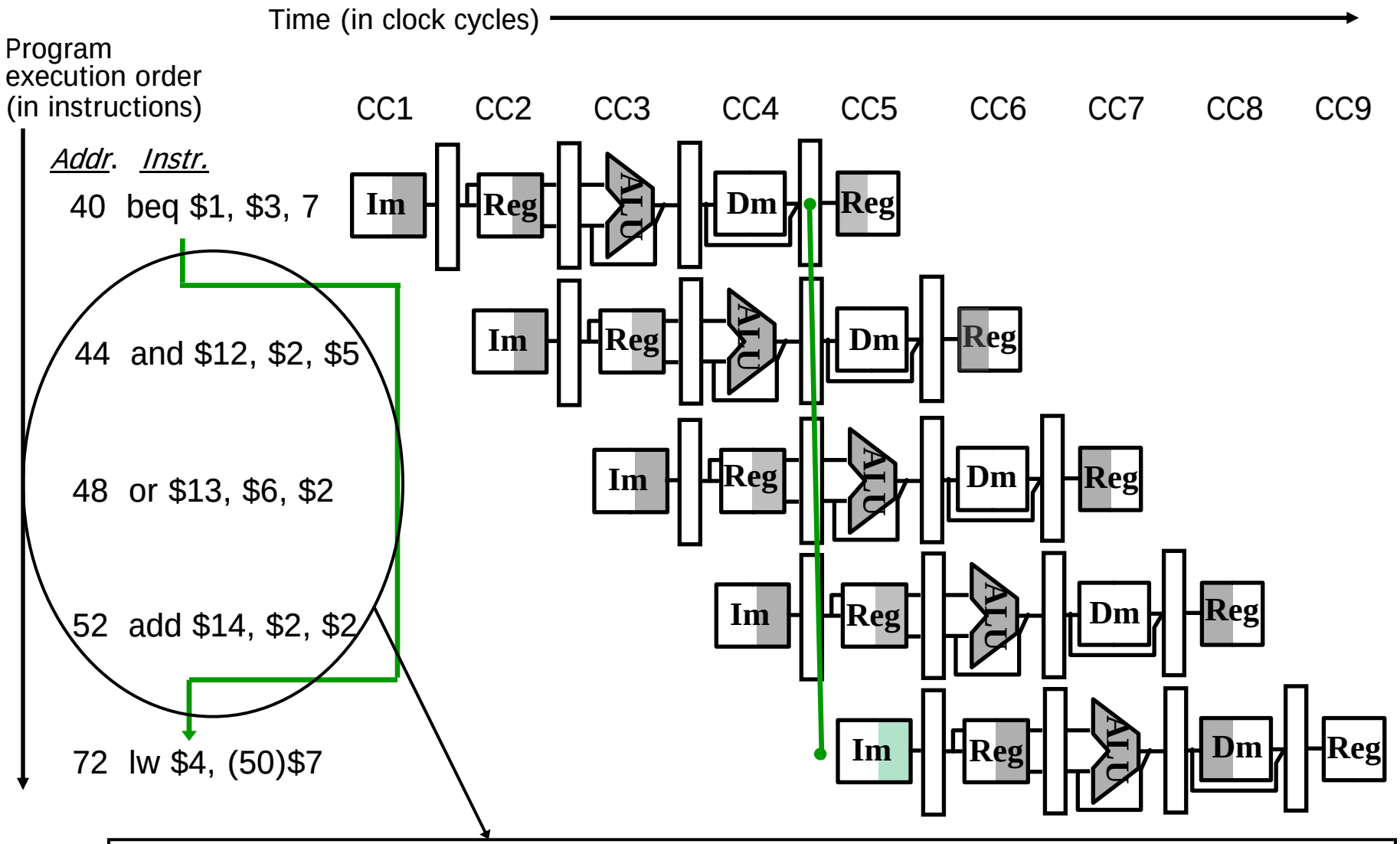
Data Hazards and Stalls (Cont')



The Pipelined Datapath with Forwarding and Hazard Detection Unit

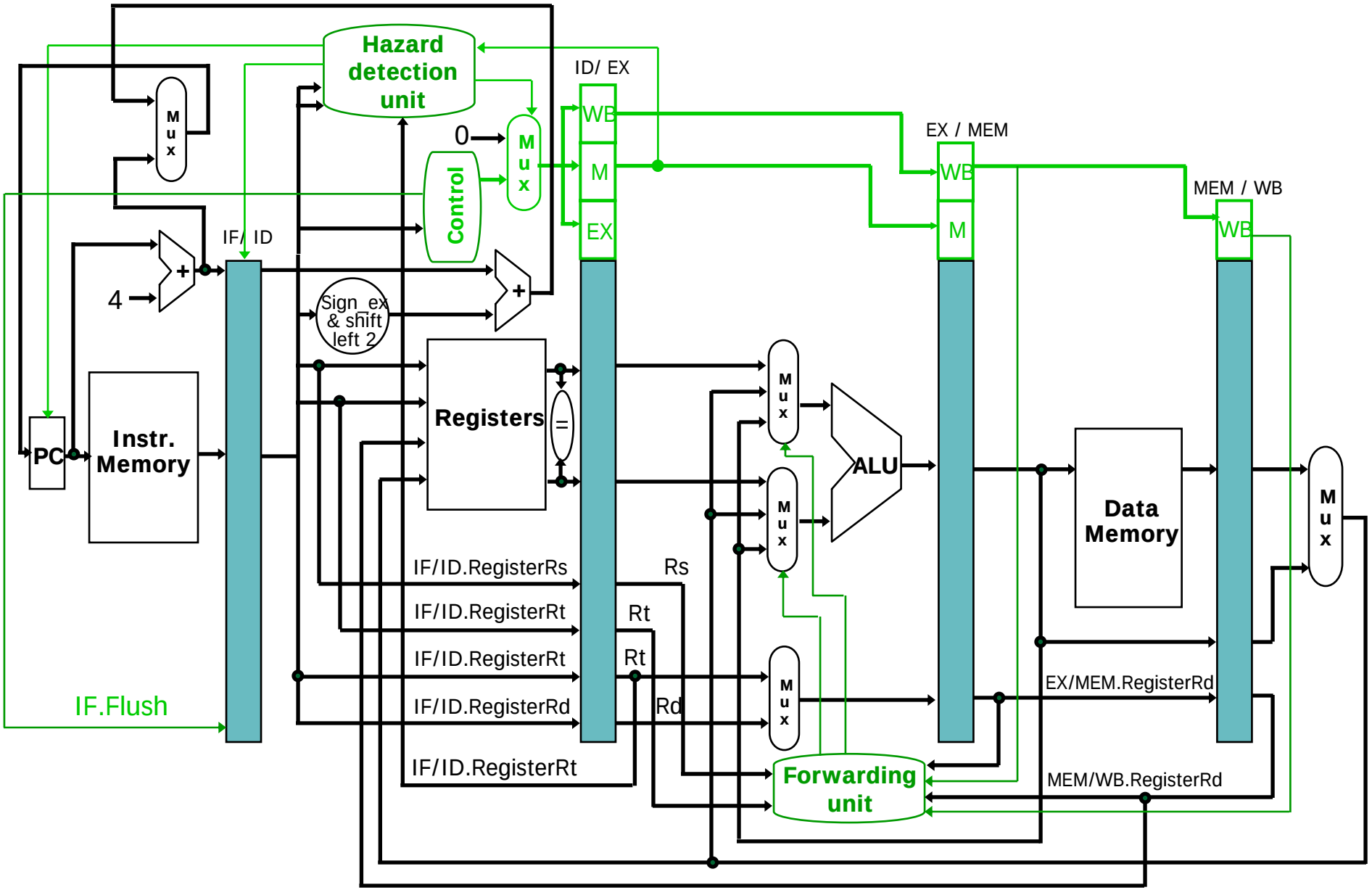


Branch Hazards



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)



Question 1

6.2 [10] <§6.1> A computer architect needs to design the pipeline of a new microprocessor. She has an example workload program core with 10^6 instructions. Each instruction takes 100 ps to finish.

a. How long does it take to execute this program core on a nonpipelined processor?

b. The current state-of-the-art microprocessor has about 20 pipeline stages. Assume it is perfectly pipelined. How much speedup will it achieve compared to the nonpipelined processor?

c. Real pipelining isn't perfect, since implementing pipelining introduces some overhead per pipeline stage. Will this overhead affect instruction latency, instruction throughput, or both?

6.2

a. It takes 100 ps * 10^6 instructions = 100 microseconds to execute on a nonpipelined processor (ignoring start and end transients in the pipeline).

b. A perfect 20-stage pipeline would speed up the execution by 20 times.

c. Pipeline overhead impacts both latency and throughput.

Question 2

6.17 [5] <§§6.4, 6.5> Consider executing the following code on the pipelined datapath of Figure 6.36 on page 416:

```
add    $2,  $3,  $1
sub    $4,  $3,  $5
add    $5,  $3,  $7
add    $7,  $6,  $1
add    $8,  $2,  $6
```

At the end of the fifth cycle of execution, which registers are being read and which register will be written?

6.18 [5] <§§6.4, 6.5> With regard to the program in Exercise 6.17, explain what the forwarding unit is doing during the fifth cycle of execution. If any comparisons are being made, mention them.

Answer 2

6.17 At the end of the first cycle, instruction 1 is fetched.

At the end of the second cycle, instruction 1 reads registers.

At the end of the third cycle, instruction 2 reads registers.

At the end of the fourth cycle, instruction 3 reads registers.

At the end of the fifth cycle, instruction 4 reads registers, and instruction 1 writes registers.

Therefore, at the end of the fifth cycle of execution, registers $6 and $1 are being read and register $2 will be written.

6.18 The forwarding unit is seeing if it needs to forward. It is looking at the instructions in the fourth and fifth stages and checking to see whether they intend to write to the register file and whether the register written is being used as an ALU input. Thus, it is comparing 3 = 4? 3 = 2? 7 = 4? 7 = 2?

Question 3

6.22 [5] <§§6.4, 6.5> Consider executing the following code on the pipelined datapath of Figure 6.36 on page 416:

```
lw   $4, 100($2)
sub  $6, $4, $3
add  $2, $3, $5
```

How many cycles will it take to execute this code? Draw a diagram like that of Figure 6.34 on page 414 that illustrates the dependencies that need to be resolved, and provide another diagram like that of Figure 6.35 on page 415 that illustrates how the code will actually be executed (incorporating any stalls or forwarding) so as to resolve the identified problems.

Answer 3

6.22 It will take 8 cycles to execute this code, including a bubble of 1 cycle due to the dependency between the `lw` and `sub` instructions.