EECS 322 Computer Architecture

Pipeline Control,

Data Hazards

and Branch Hazards

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

Models

Single-cycle model (non-overlapping)

- The instruction latency executes in a single cycle
- Every instruction and clock-cycle must be stretched to the slowest instruction (p.438)

Multi-cycle model (non-overlapping)

- The instruction latency executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- Ability to share functional units within the execution of a single instruction

Pipeline model (overlapping, p. 522)

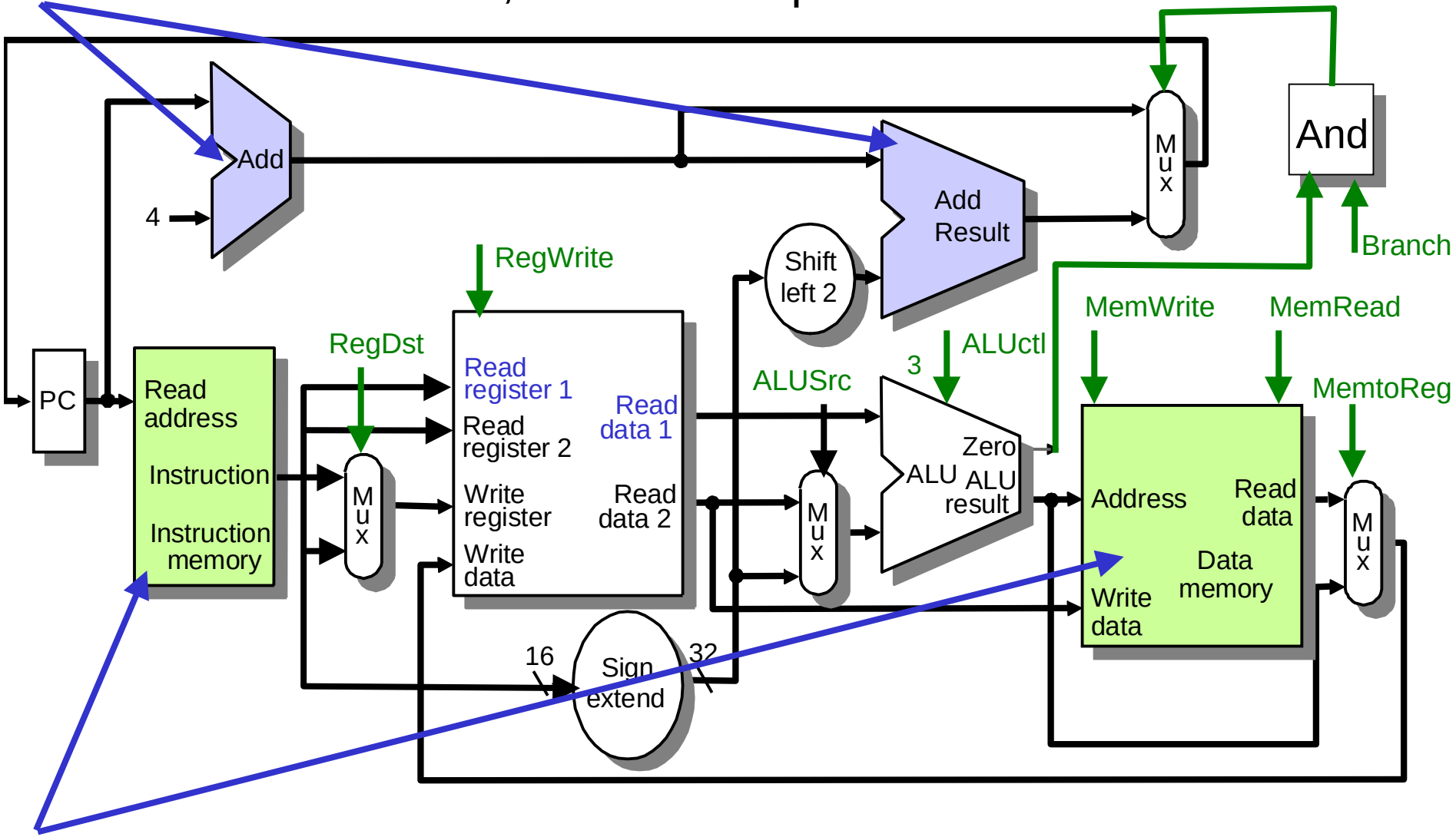
- The instruction latency executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- The throughput is mainly one clock-cycle/instruction
- Gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization. (p. 377)

Recap: Can pipelining get us into trouble?

- **Yes: Pipeline Hazards**
  - **structural hazards: attempt to use the same resource two different ways at the same time**
    - **e.g., multiple memory accesses, multiple register writes**
    - **solutions:**
      - **multiple memories (separate instruction & data memory)**
      - **stretch pipeline**
  - **control hazards: attempt to make a decision before condition is evaulated**
    - **e.g., any conditional branch**
    - **solutions: prediction, delayed branch**
  - **data hazards: attempt to use item before it is ready**
    - **e.g., add r1,r2,r3; sub r4, r1 ,r5; lw r6, 0(r7); or r8, r6 ,r9**
    - **solutions: forwarding/bypassing, stall/bubble**

Review: Single-Cycle Datapath

2 adders: PC+4 adder, Branch/Jump offset adder

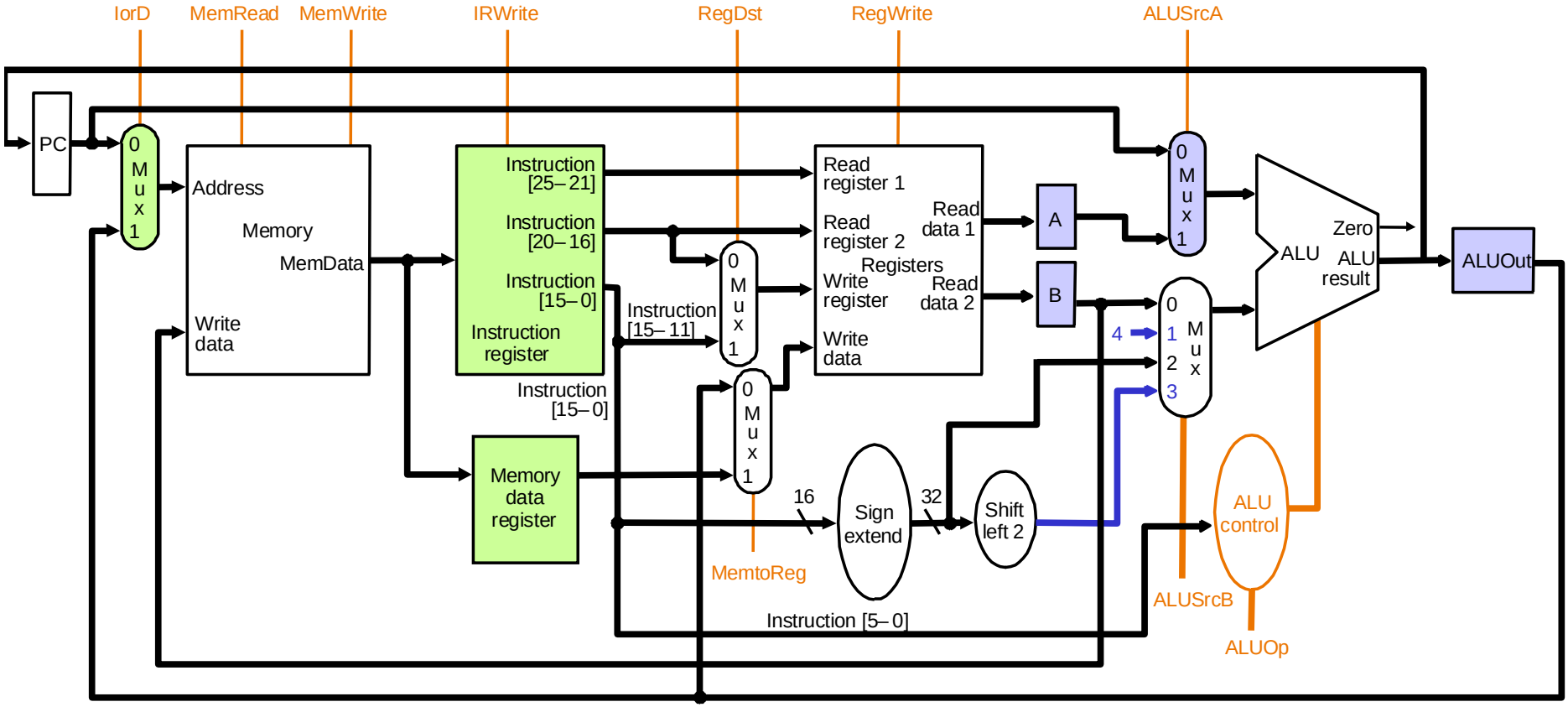


Harvard Architecture: Separate instruction and data memory

Review: Multi vs. Single-cycle Processor Datapath

Combine adders: add 1½ Mux & 3 temp. registers, A, B, ALUOut

Combine Memory: add 1 Mux & 2 temp. registers, IR, MDR



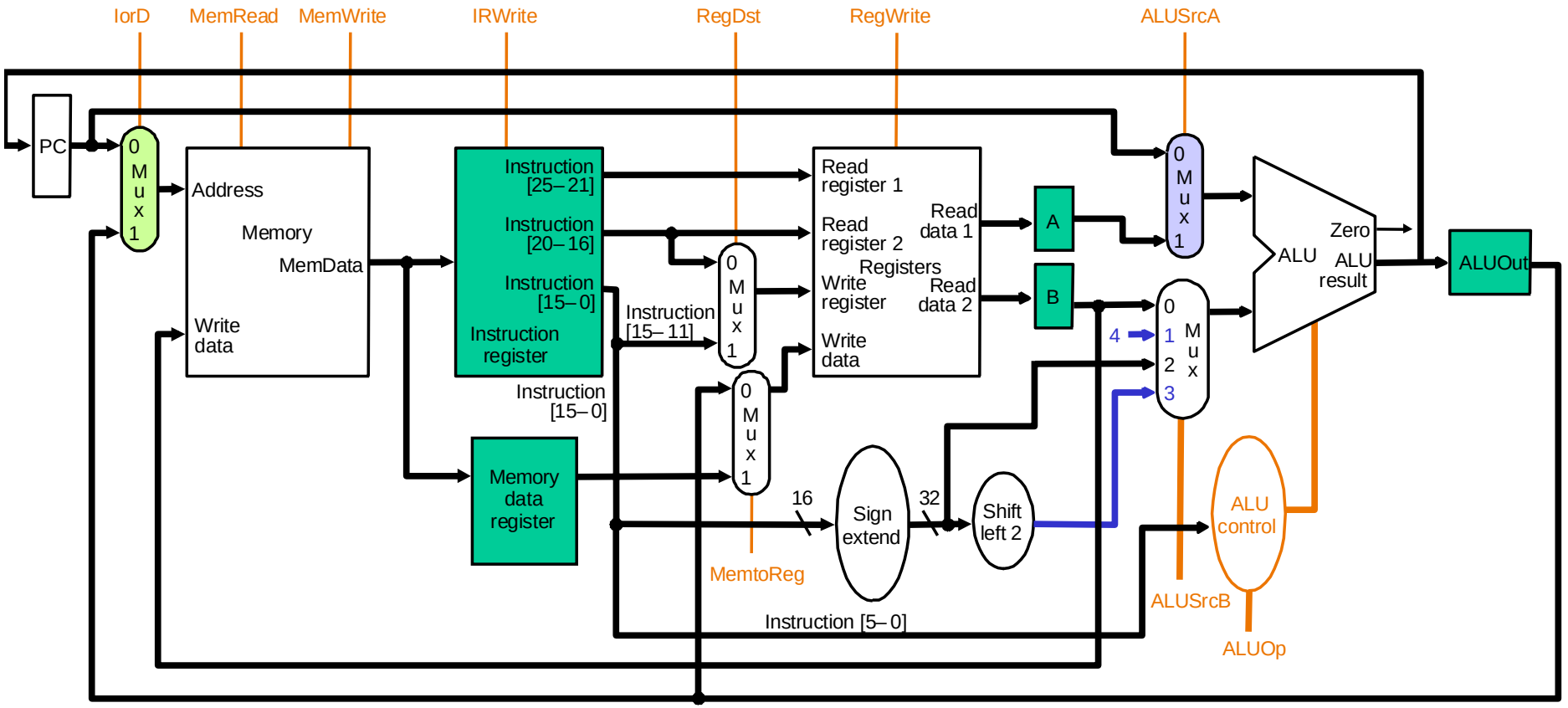
Single-cycle= 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM

Multi-cycle Processor Datapath

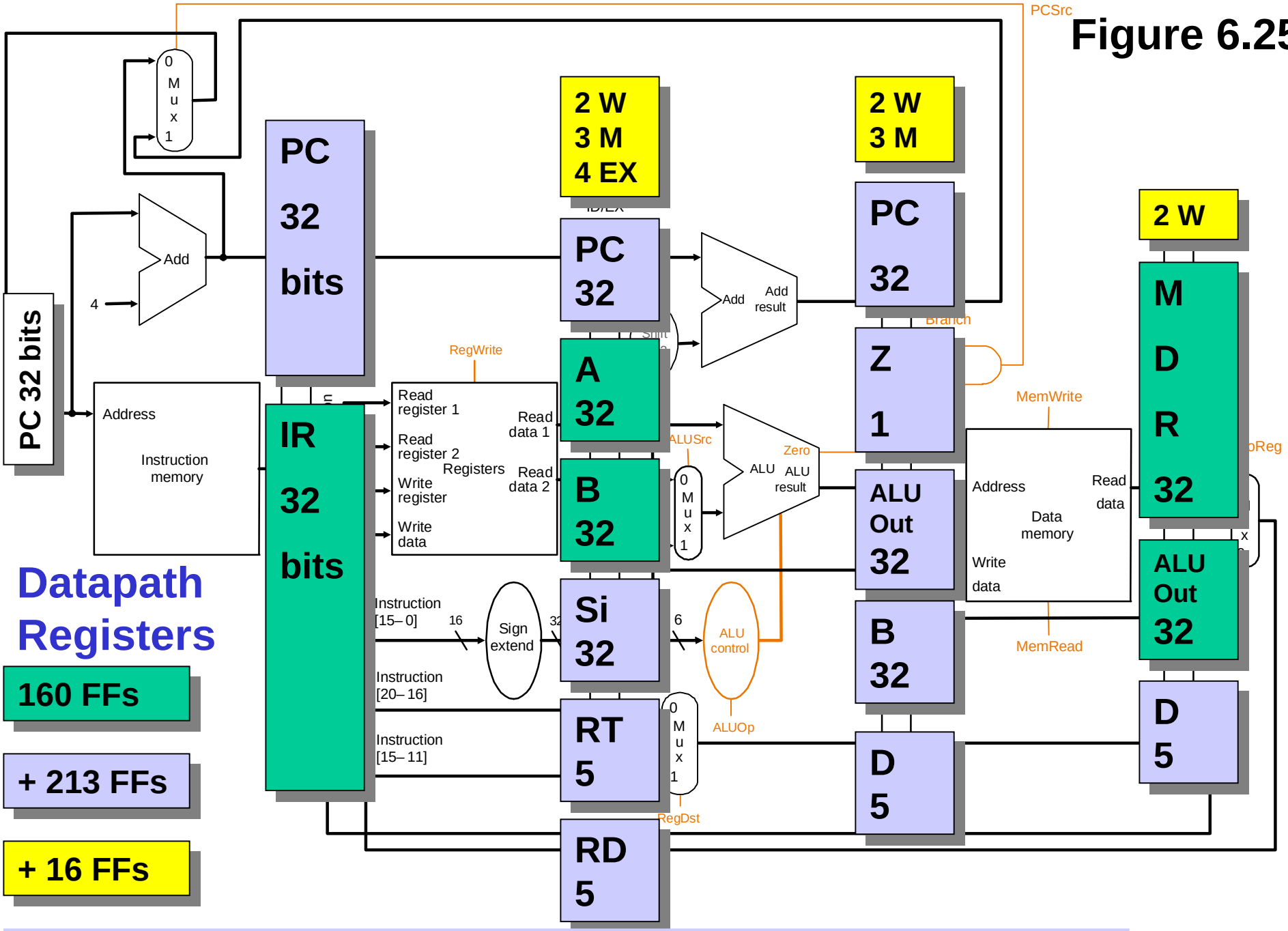
Single-cycle= 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM



5x32 = 160 additional FFs for multi-cycle processor over single-cycle processor

Figure 6.25



Datapath Registers

- 160 FFs
- + 213 FFs
- + 16 FFs

213+16 = 229 additional FFs for pipeline over multi-cycle processor

Overhead

Single-cycle model

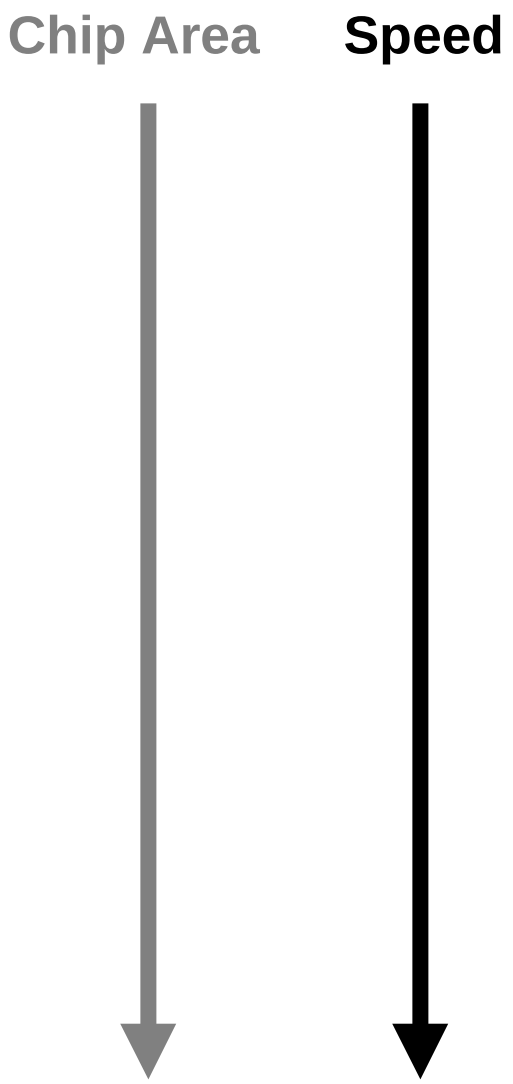
- 8 ns Clock (125 MHz), (non-overlapping)
- 1 ALU + 2 adders
- 0 Muxes
- 0 Datapath Register bits (Flip-Flops)

Multi-cycle model

- 2 ns Clock (500 MHz), (non-overlapping)
- 1 ALU + Controller
- 5 Muxes
- 160 Datapath Register bits (Flip-Flops)

Pipeline model

- 2 ns Clock (500 MHz), (overlapping)
- 2 ALU + Controller
- 4 Muxes
- 373 Datapath + 16 Controlpath Register bits (Flip-Flops)



Pipeline Control: Controlpath Register bits

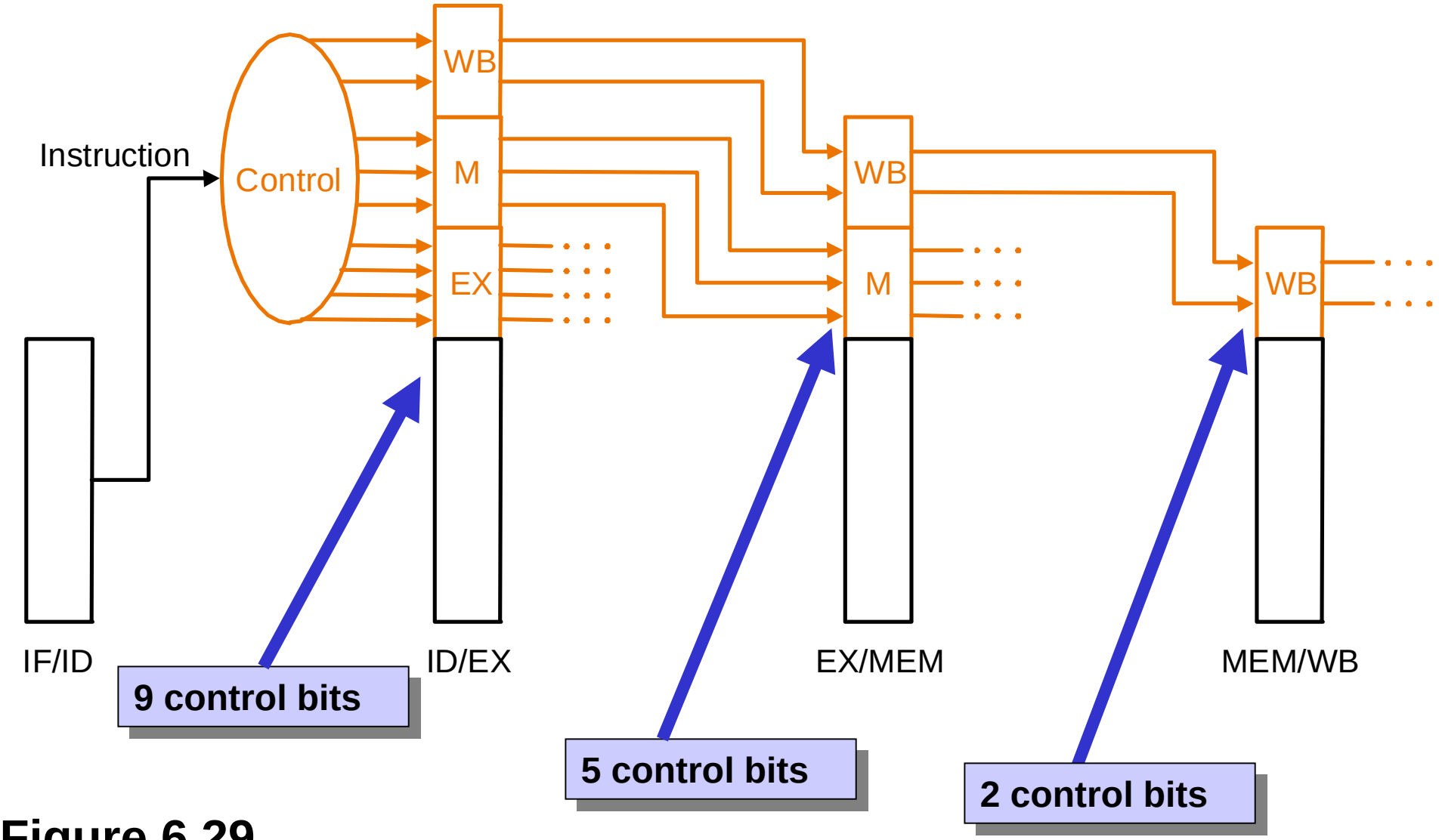


Figure 6.29

Pipeline Control: Controlpath table

Figure 5.20, Single Cycle

Instruction	Reg Dst	ALU Src	Mem Reg	Reg Wrt	Mem Red	Mem Wrt	Bra-nch	ALU op1	ALU op0
R-format	1	0	0	1	0	0	0	1	0
lw	1	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Figure 6.28

	ID / EX control lines				EX / MEM control lines			MEM / WB cntrl lines	
Instruction	Reg Dst	ALU Op1	ALU Op0	ALU Src	Bra-nch	Mem Red	Mem Wrt	Reg Wrt	Mem Reg
R-format	1	1	0	0	0	0	0	1	0
lw	1	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

Pipeline Hazards

Pipeline hazards

- **Solution #1 always works (for non-realtime) applications: stall, delay & procrastinate!**

Structural Hazards (i.e. fetching same memory bank)

- **Solution #2: partition architecture**

Control Hazards (i.e. branching)

- **Solution #1: stall! but decreases throughput**
- **Solution #2: guess and back-track**
- **Solution #3: delayed decision: delay branch & fill slot**

Data Hazards (i.e. register dependencies)

- **Worst case situation**
- **Solution #2: re-order instructions**
- **Solution #3: forwarding or bypassing: delayed load**

Pipeline Datapath and Controlpath

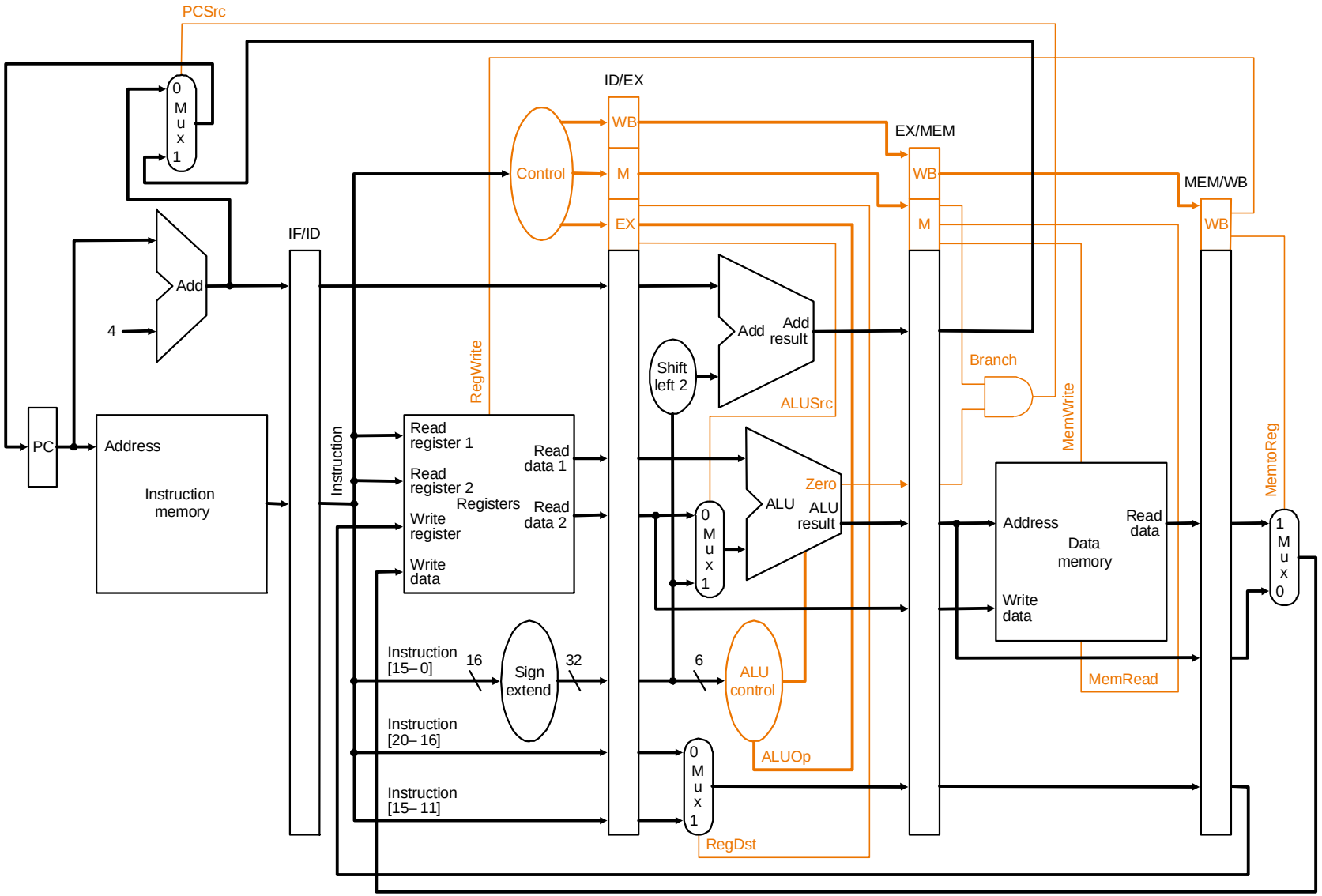


Figure 6.30

load inst.

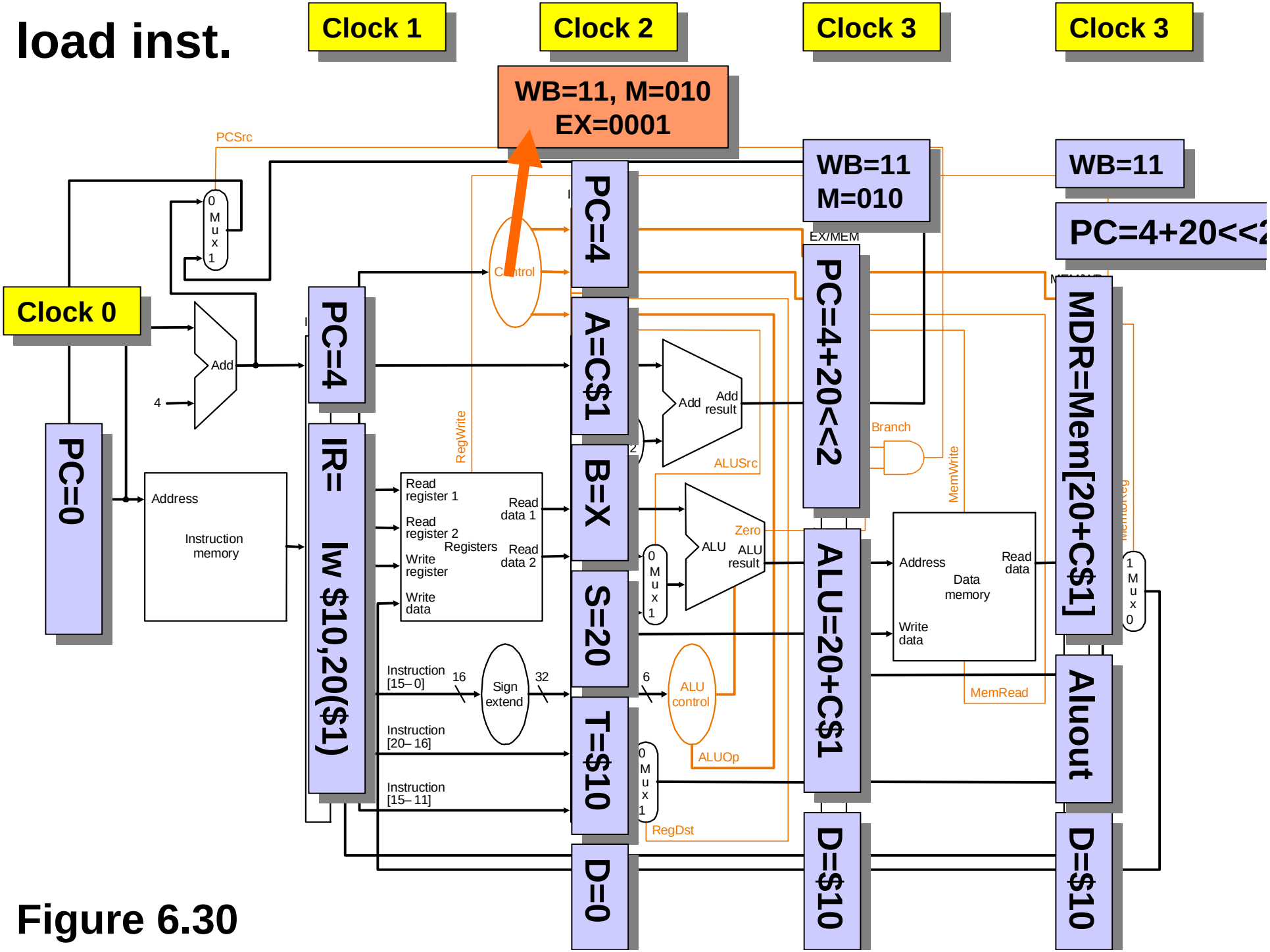


Figure 6.30

load inst.

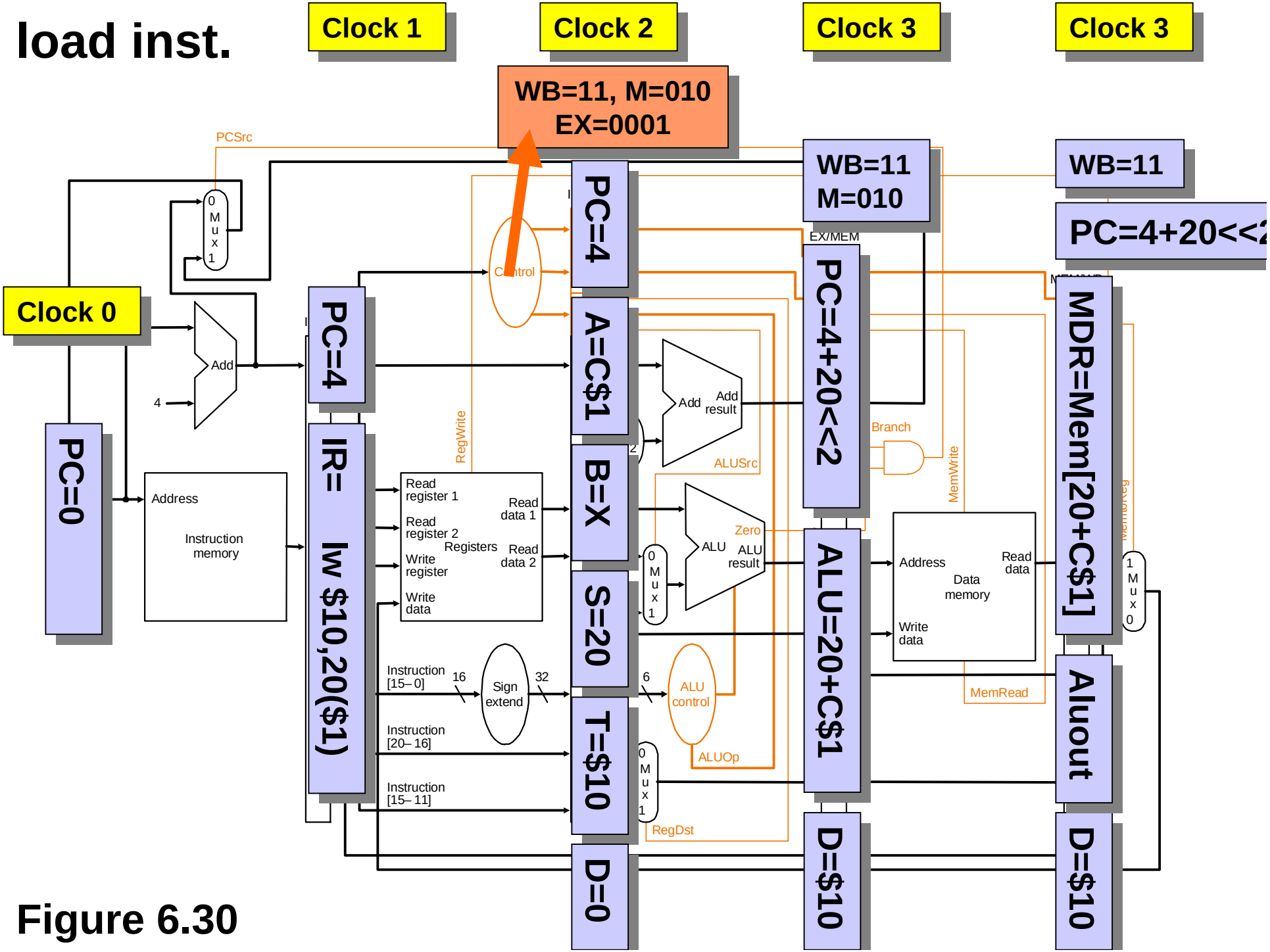


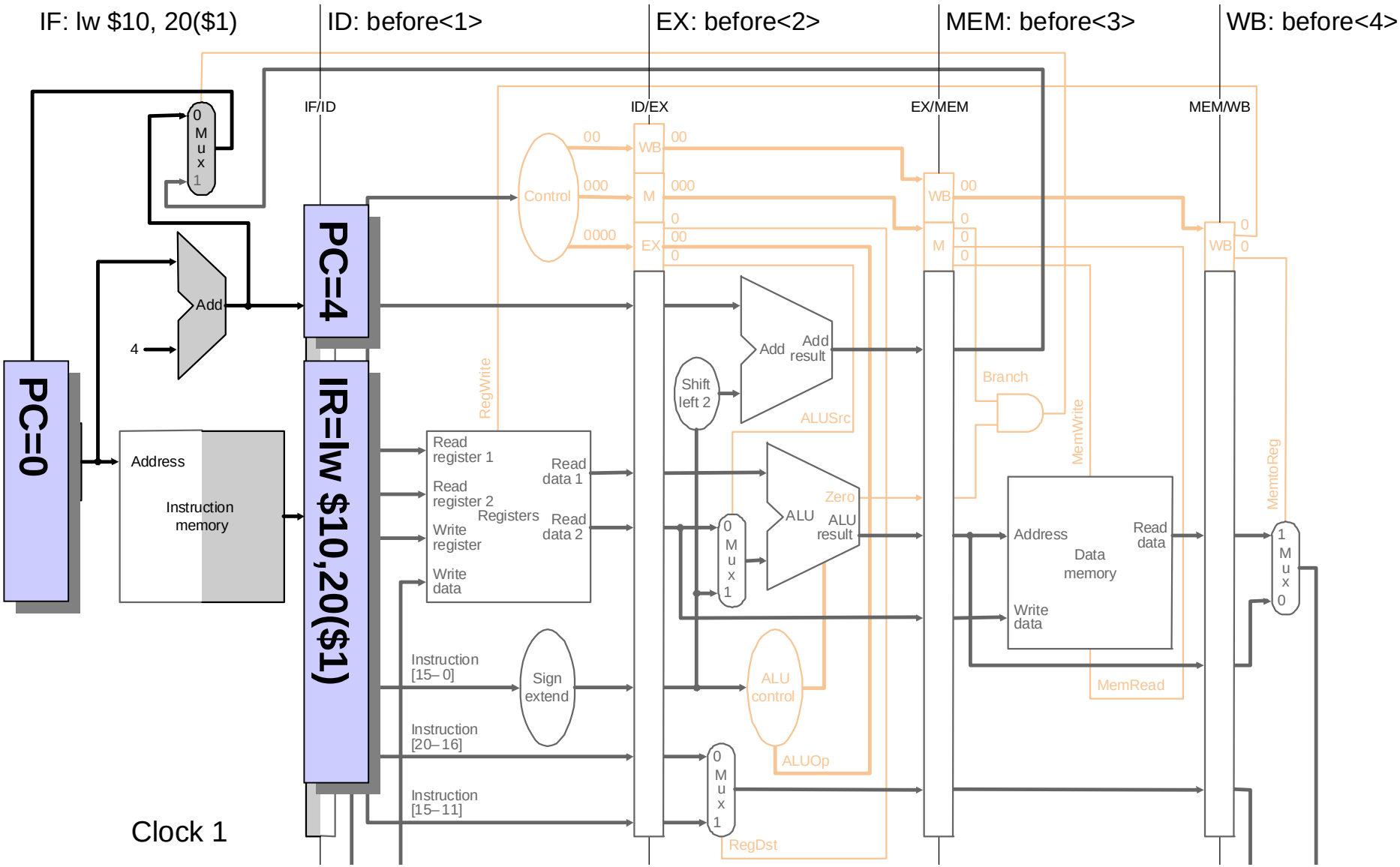
Figure 6.30

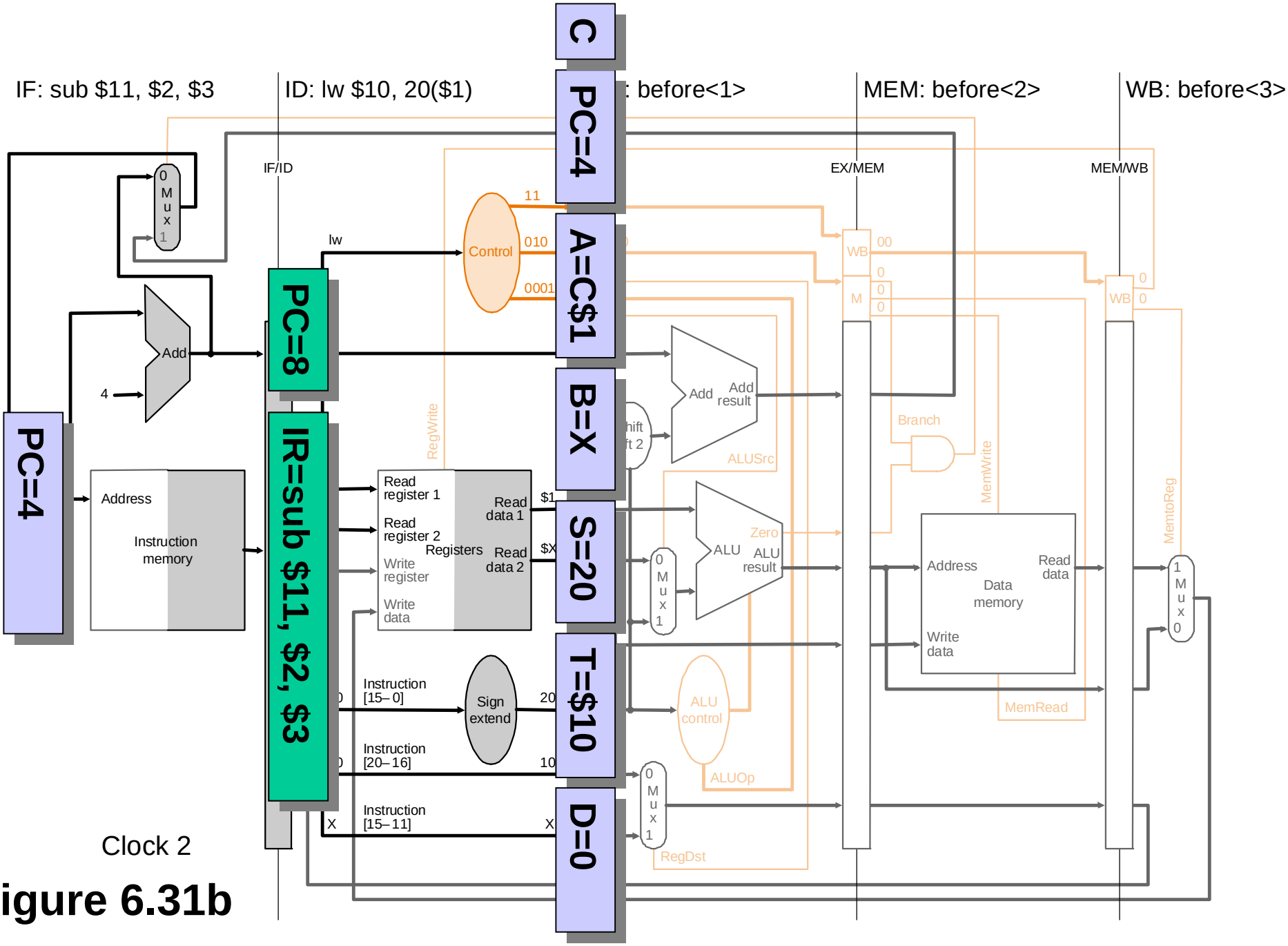
Pipeline single stepping

Contents of Register 1 = C$1 = 3; C$2=4; C$3=4; C$4=6; C$5=7; C$10=8; … Memory[23]=9;
Formats: add $rd,$rs=A,$rt=B; lw $rt=B,@($rs=A)

Clock	<IF/ID>	<ID/EX>	<EX/MEM>	<MEM/WB>
	<PC, IR>	<PC, A, B, S, Rt, Rd>	<PC, Z, ALU, B, R>	<MDR, ALU, R>
0	<0,?>	<?,?,?,?,?,?>	<?,?,?,?,?>	<?,?,?>
1	<4,lw $10,20($1)>	<0,?,?,?,?,?>	<?,?,?,?,?>	<?,?,?>
2	<8,sub $11,$2,$3>	<4,C$1→3,C$10→8,20,$10,0>	<0,?,?,?,?>	<?,?,?>
3	<12,and $12,$4,$5>	<8,C$2→4,C$3→4,X,$3,$11>	<4+20<<2→84,0,20+3→23,8,$10>	<?,?,?>
4	<16,or $13,$6,$7>	<12,C$4→6,C$5→7,X,$5,$12>	<X,1,4-4=0,4,$11>	<Mem[23]→9,23,$10>
5	<20,add $14,$8,$9>	<16,C$6 ,C$7,X,$7,$13>	<X,0,1,7,$12>	<X,0,$11>

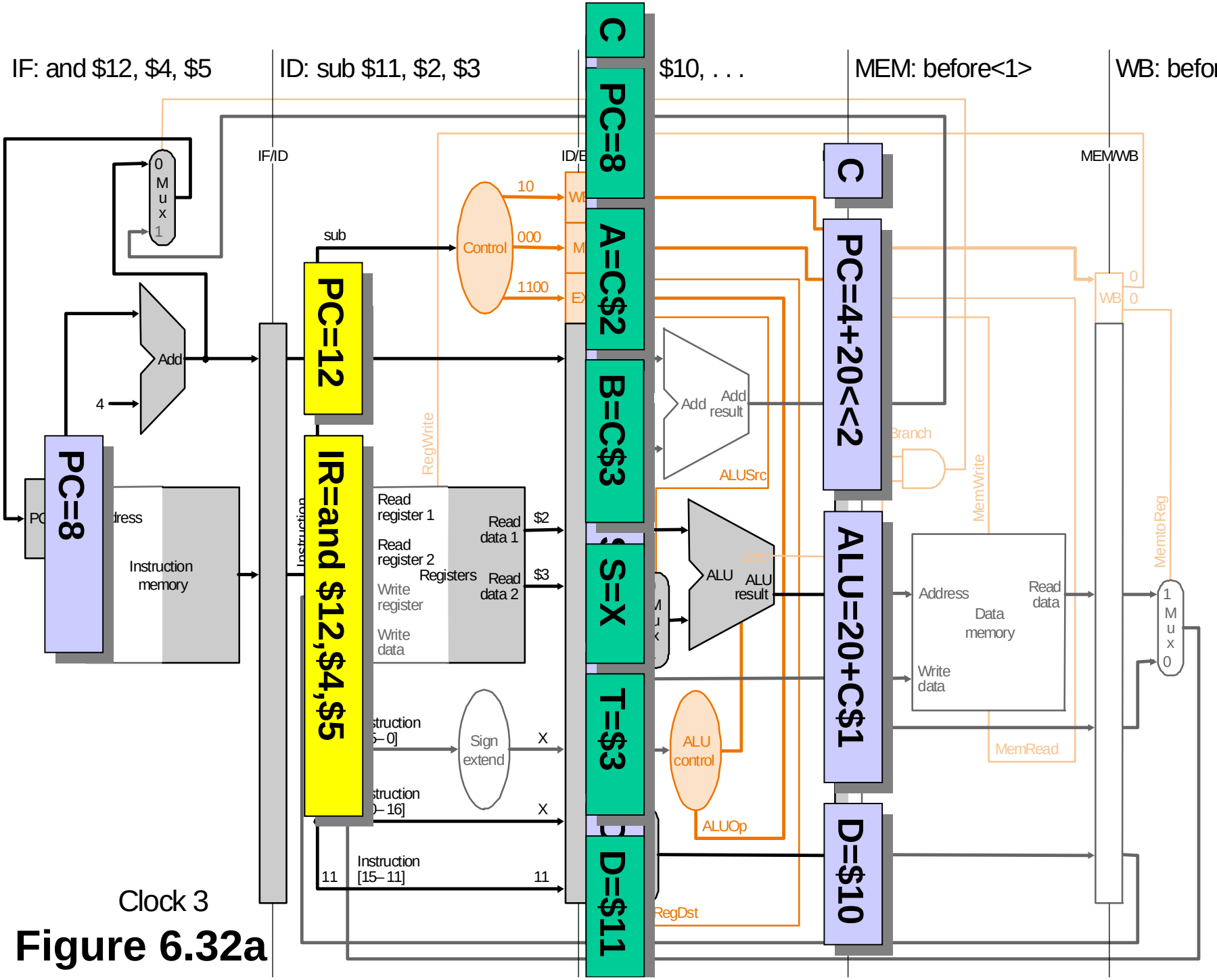
Clock 1: Figure 6.31a





Clock 2

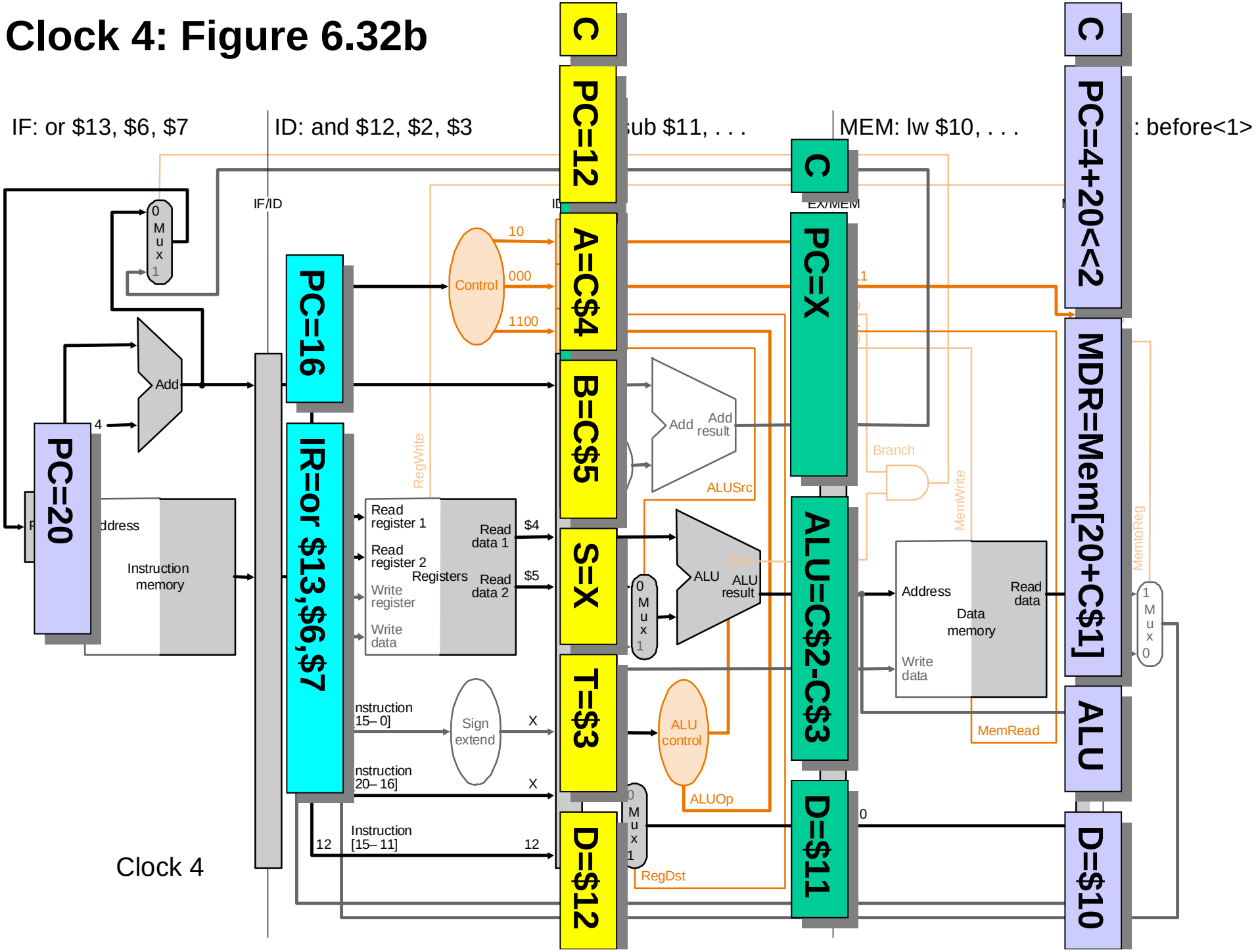
Figure 6.31b



Clock 3

Figure 6.32a

Clock 4: Figure 6.32b



Data Dependencies: that can be resolved by forwarding

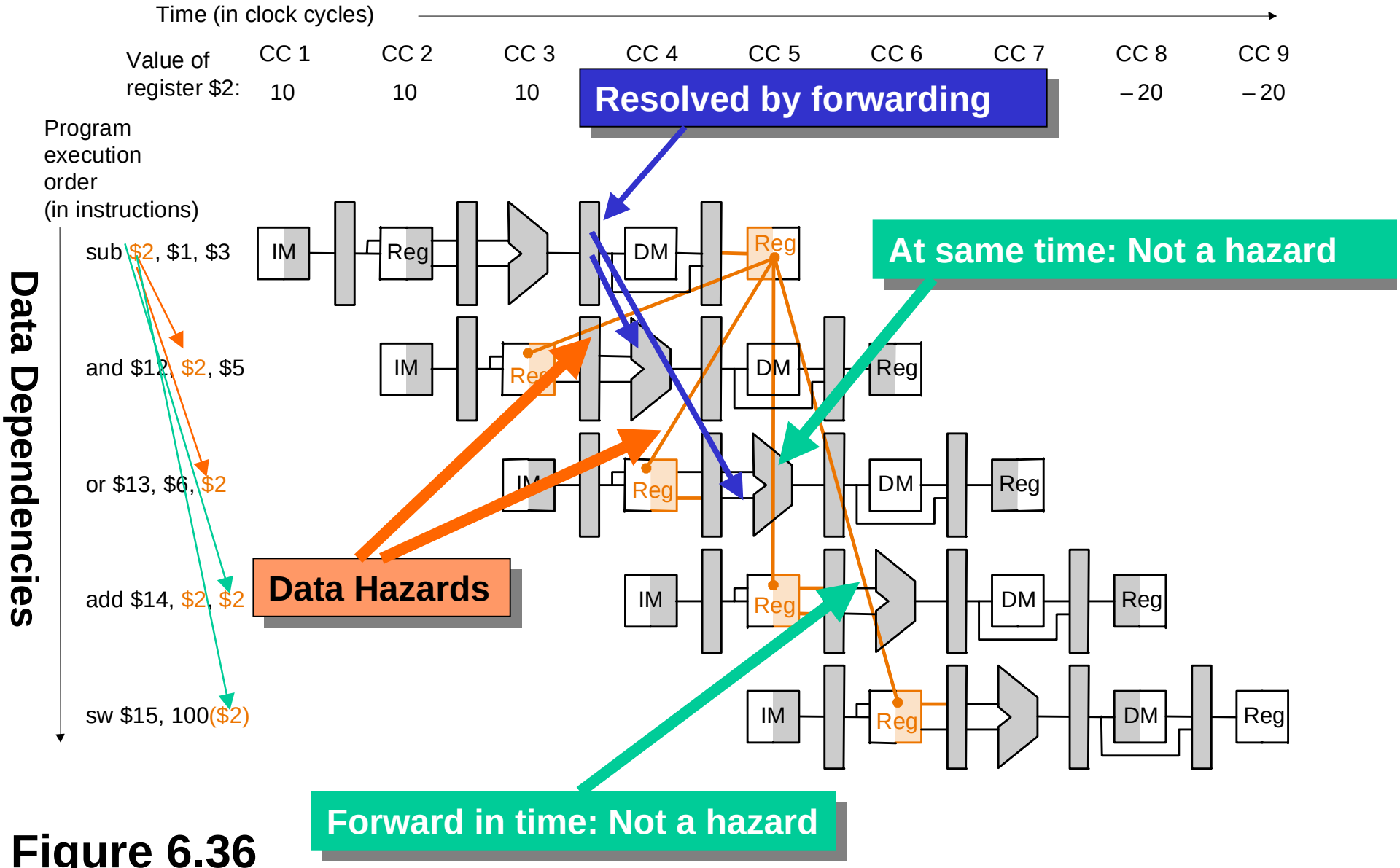


Figure 6.36

Data Hazards: arithmetic

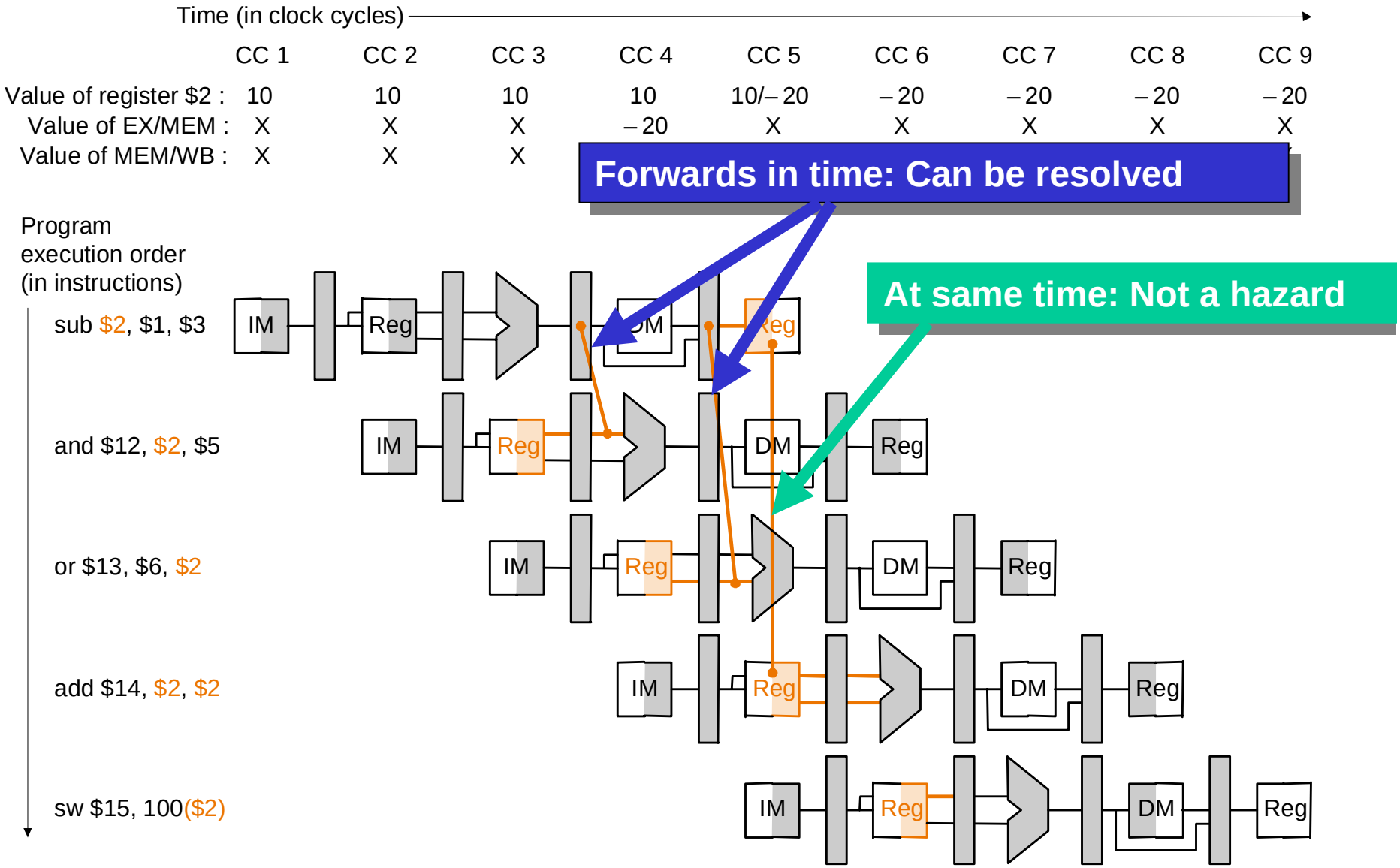
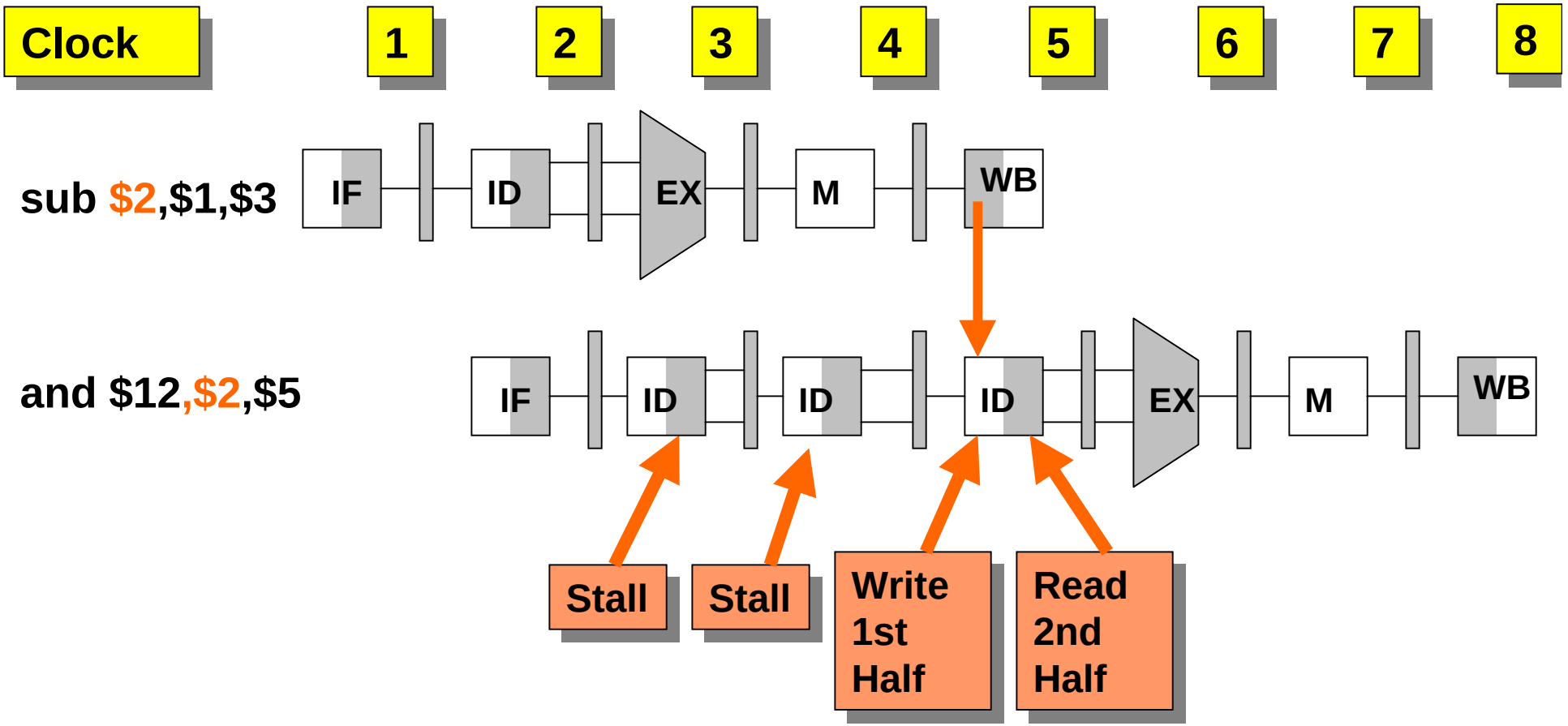


Figure 6.37

Data Dependencies: no forwarding



Suppose every instruction is dependant = 1 + 2 stalls = 3 clocks

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{3} = 167\ \text{MIPS}$$

Data Dependencies: no forwarding

A dependant instruction will take = 1 + 2 stalls = 3 clocks

An independent instruction will take = 1 + 0 stalls = 1 clocks

Suppose 10% of the time the instructions are dependant?

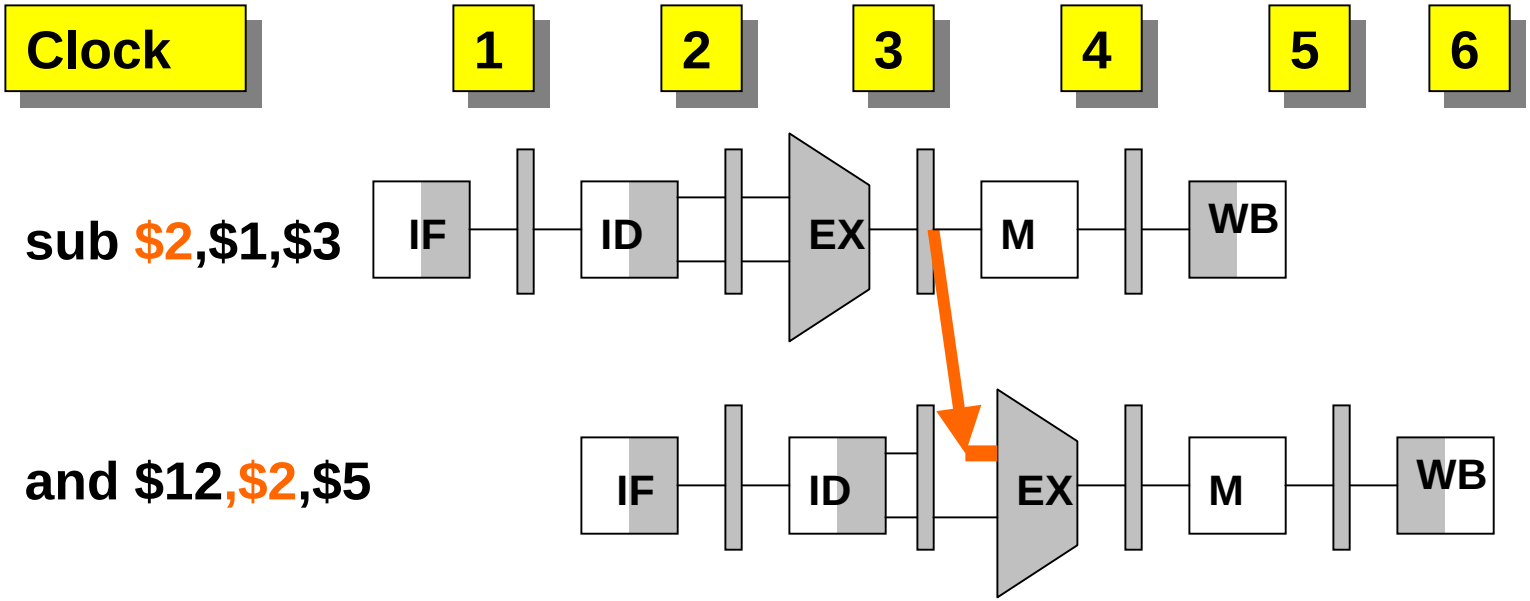
Averge instruction time = 10%*3 + 90%*1 = 0.10*3 + 0.90*1 = 1.2 clocks

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{1.2} = 417\ \text{MIPS}\ \ (10\%\ \text{dependency})$$

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{3} = 167\ \text{MIPS}\ \ (100\%\ \text{dependency})$$

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{1} = 500\ \text{MIPS}\ \ (0\%\ \text{dependency})$$

Data Dependencies: with forwarding



Detected
Data Hazard 1a
ID/EX.$rs = EX/M.$rd

Suppose every instruction is dependant = 1 + 0 stalls = 1 clock

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{1} = 500\ \text{MIPS}$$

Data Dependencies: Hazard Conditions

Data Hazard Condition

occurs whenever a data source needs a previous unavailable result due to a data destination.

Example

```
sub   $2,  $1,  $3        sub   $rd,  $rs,  $rt
and  $12,  $2,  $5        and   $rd,  $rs,  $rt
```

Data Hazard Detection

is always comparing a destination with a source.

Destination	Source	Hazard Type
EX/MEM.$rdest =	ID/EX.$rs	1a.
	ID/EX.$rt	1b.
MEM/WB.$rdest =	ID/EX.$rs	2a.
	ID/EX.$rt	2b.

Data Dependencies: Hazard Conditions

1a Data Hazard:

```
sub  $2,  $1,  $3
and  $12, $2,  $5
```

EX/MEM.$rd = ID/EX.$rs

```
sub  $rd, $rs, $rt
and  $rd, $rs, $rt
```

1b Data Hazard:

```
sub  $2,  $1,  $3
and  $12, $1,  $2
```

EX/MEM.$rd = ID/EX.$rt

```
sub  $rd, $rs, $rt
and  $rd, $rs, $rt
```

2a Data Hazard:

```
sub  $2,  $1,  $3
and  $12, $1,  $5
or   $13, $2,  $1
```

MEM/WB.$rd = ID/EX.$rs

```
sub  $rd, $rs, $rt
sub  $rd, $rs, $rt
and  $rd, $rs, $rt
```

2b Data Hazard:

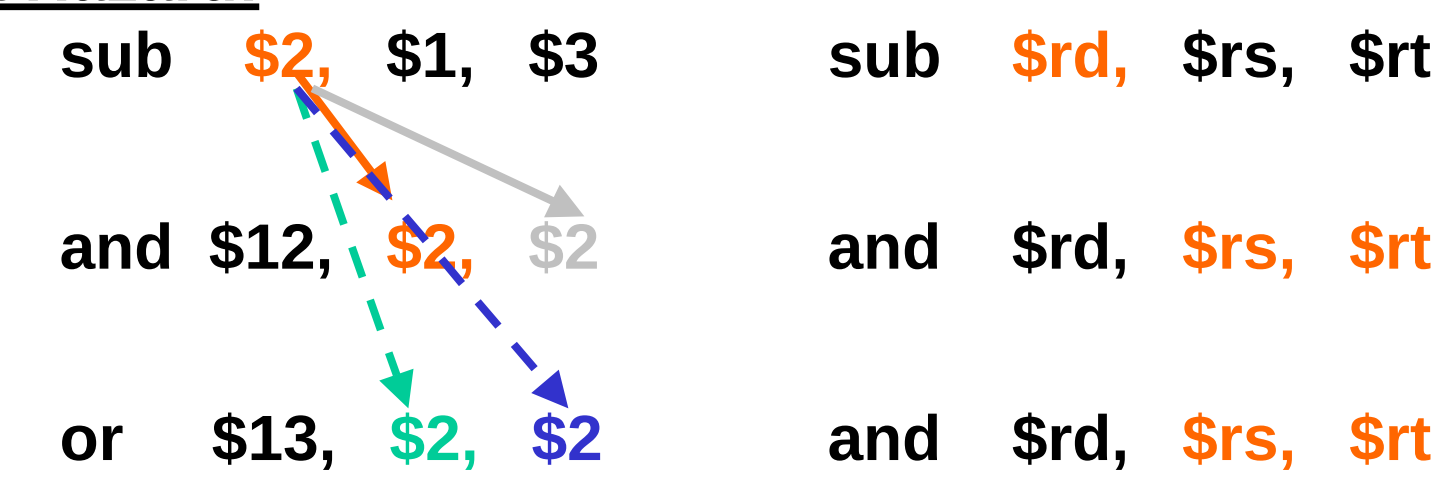
```
sub  $2,  $1,  $3
and  $12, $1,  $5
or   $13, $6,  $2
```

MEM/WB.$rd = ID/EX.$rt

```
sub  $rd, $rs, $rt
sub  $rd, $rs, $rt
and  $rd, $rs, $rt
```

Data Dependencies: Worst case

Data Hazard:



Data Hazard 1a: EX/MEM.$rd = ID/EX.$rs

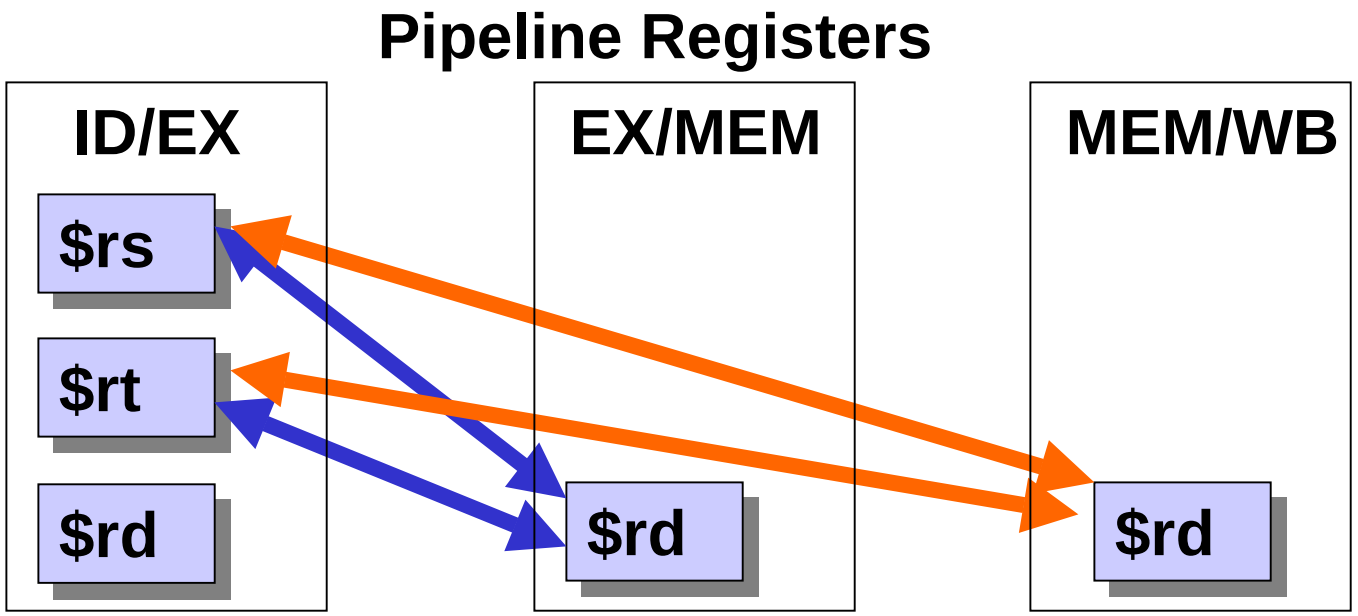
Data Hazard 1b: EX/MEM.$rd = ID/EX.$rt

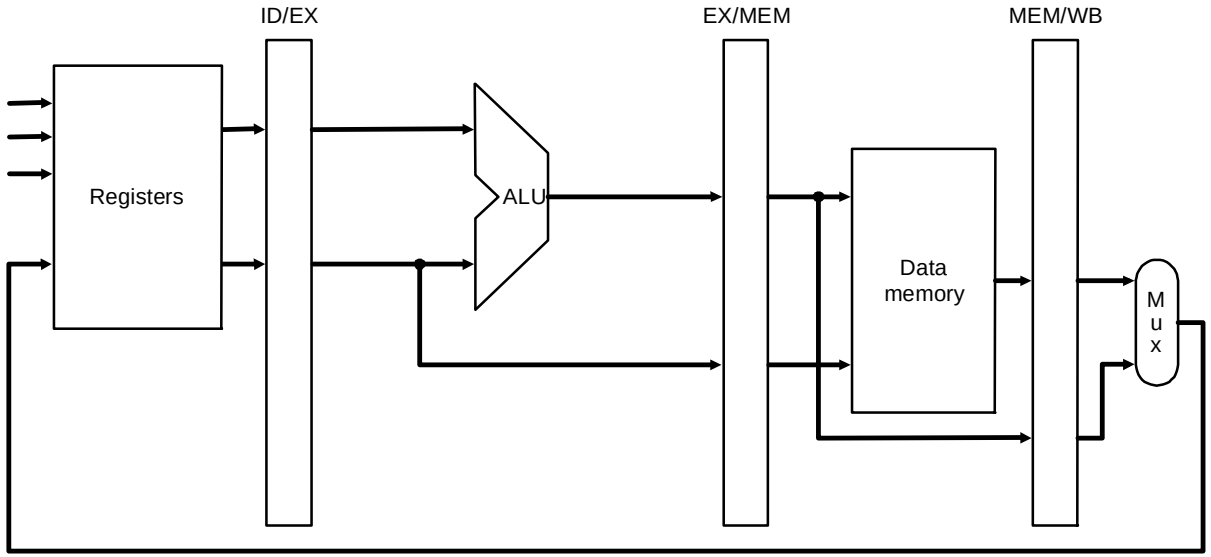
Data Hazard 2a: MEM/WB.$rd = ID/EX.$rs

Data Hazard 2b: MEM/WB.$rd = ID/EX.$rt

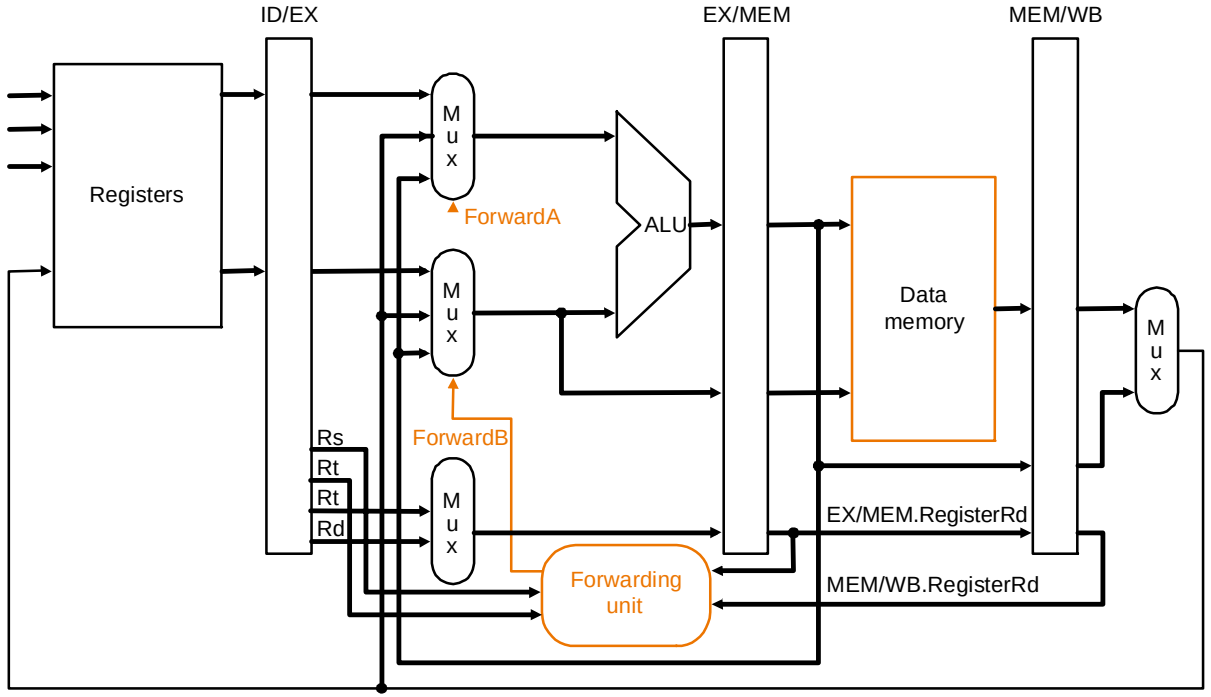
Data Dependencies: Hazard Conditions

Hazard Type	Source	Destination
1a.	ID/EX.$rs	= EX/MEM.$rdest
1b.	ID/EX.$rt	= EX/MEM.$rdest
2a.	ID/EX.$rs	= MEM/WB.$rdest
2b.	ID/EX.$rt	= MEM/WB.$rdest





a. No forwarding



b. With forwarding

Figure 6.38

Data Hazards: Loads

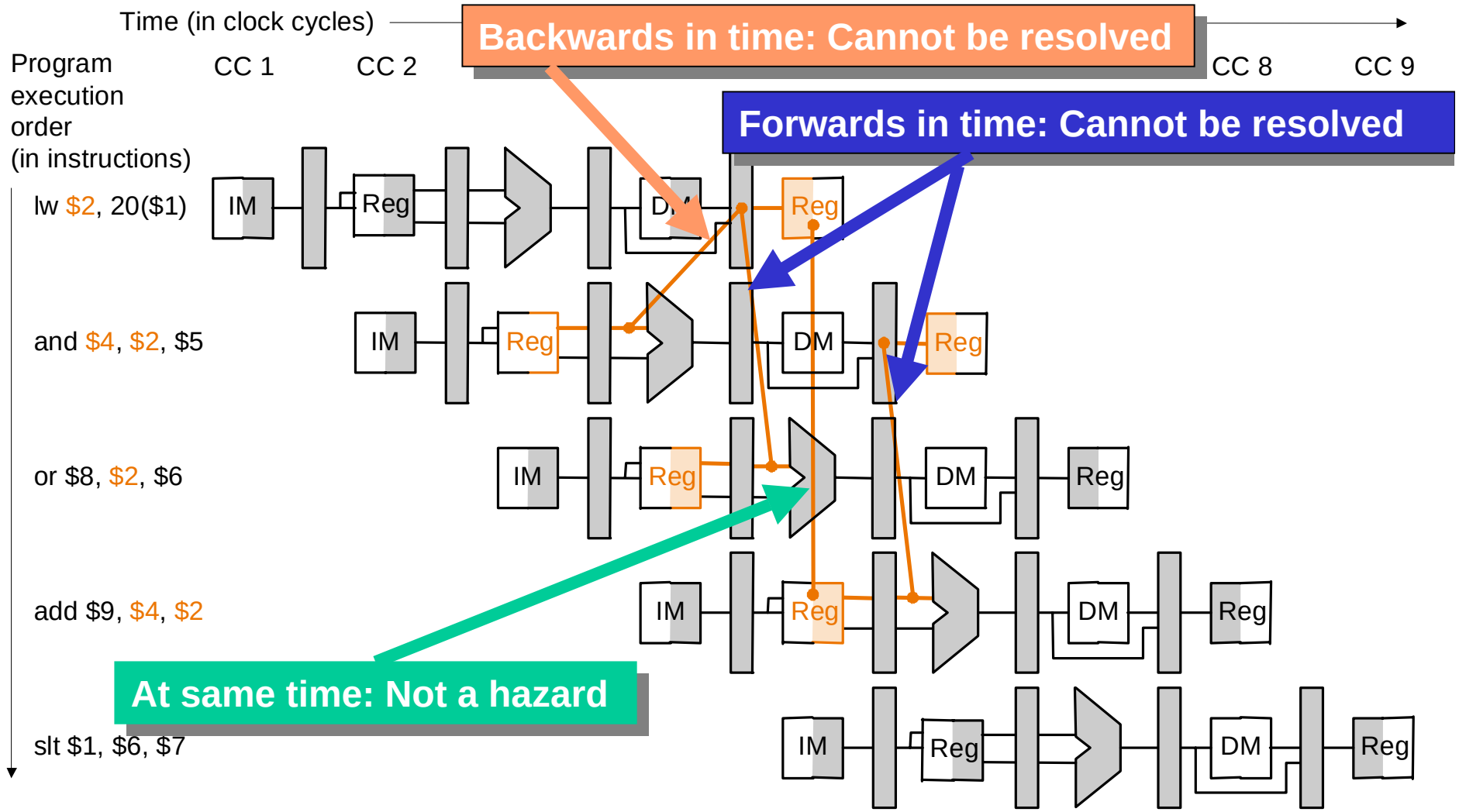


Figure 6.44

Data Hazards: load stalling

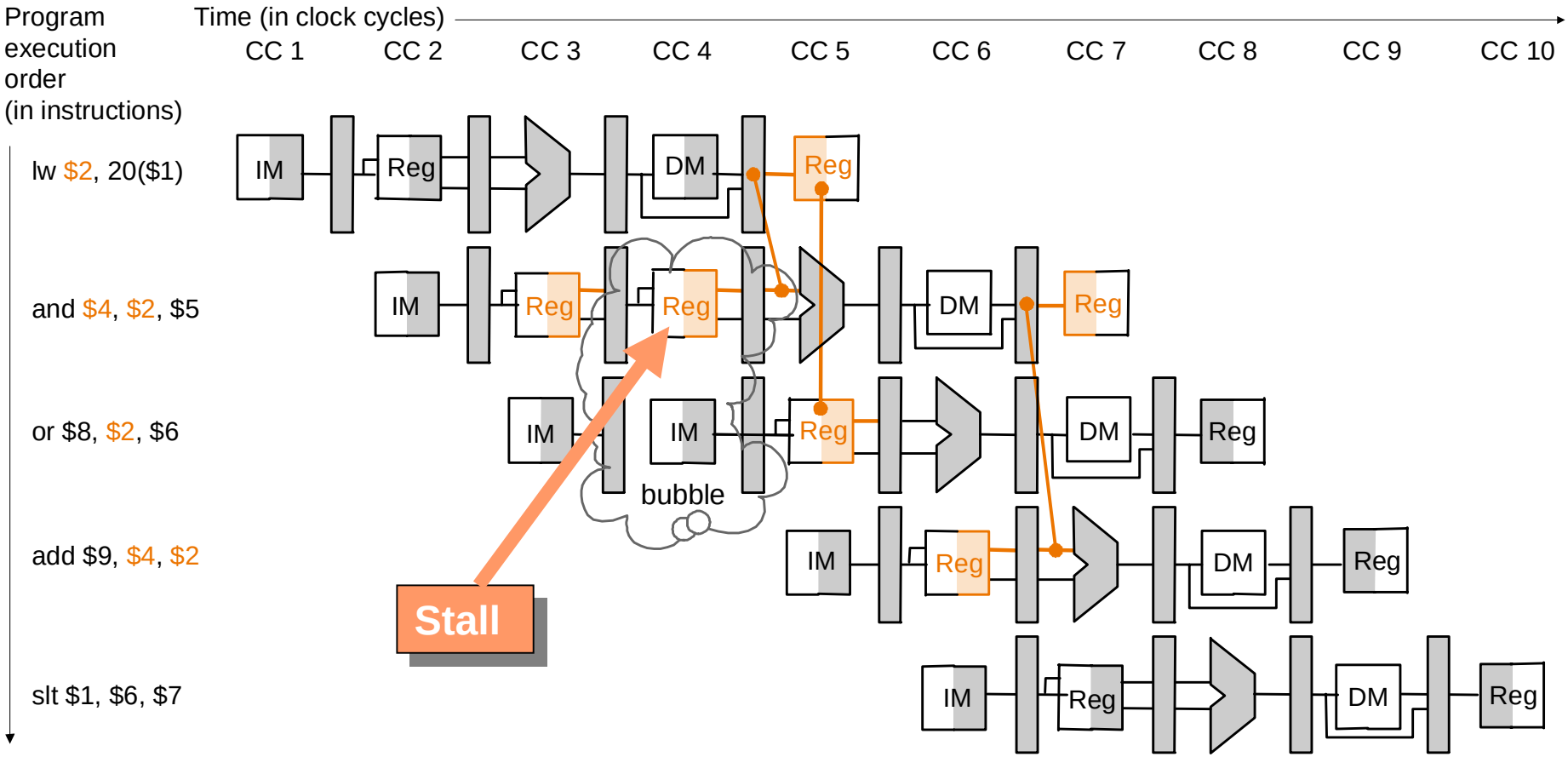


Figure 6.45

Data Hazards: Hazard detection unit (page 490)

Stall Condition

Source **Destination**

IF/ID.$rs
IF/ID.$rt
} = ID/EX.$rt Λ ID/EX.MemRead=1

Stall Example

```
lw   $2,  20($1)          lw   $rt,  addr($rs)
and  $4,  $2,  $5         and  $rd,  $rs,  $rt
```

No Stall Example: (only need to look at next instruction)

```
lw   $2,  20($1)          lw   $rt,  addr($rs)
and  $4,  $1,  $5         and  $rd,  $rs,  $rt
or   $8,  $2,  $6         or   $rd,  $rs,  $rt
```

Data Hazards: Hazard detection unit (page 490)

No Stall Example: (only need to look at next instruction)

```
lw    $2,  20($1)        lw    $rt,  addr($rs)
and   $4,  $1,  $5       and   $rd,  $rs,  $rt
or    $8,  $2,  $6       or    $rd,  $rs,  $rt
```

Example

load: assume half of the instructions are immediately followed by an instruction that uses it.

What is the average number of clocks for the load?

load instruction time: 50%*(1 clock) + 50%*(2 clocks)=1.5

Hazard Detection Unit: when to stall

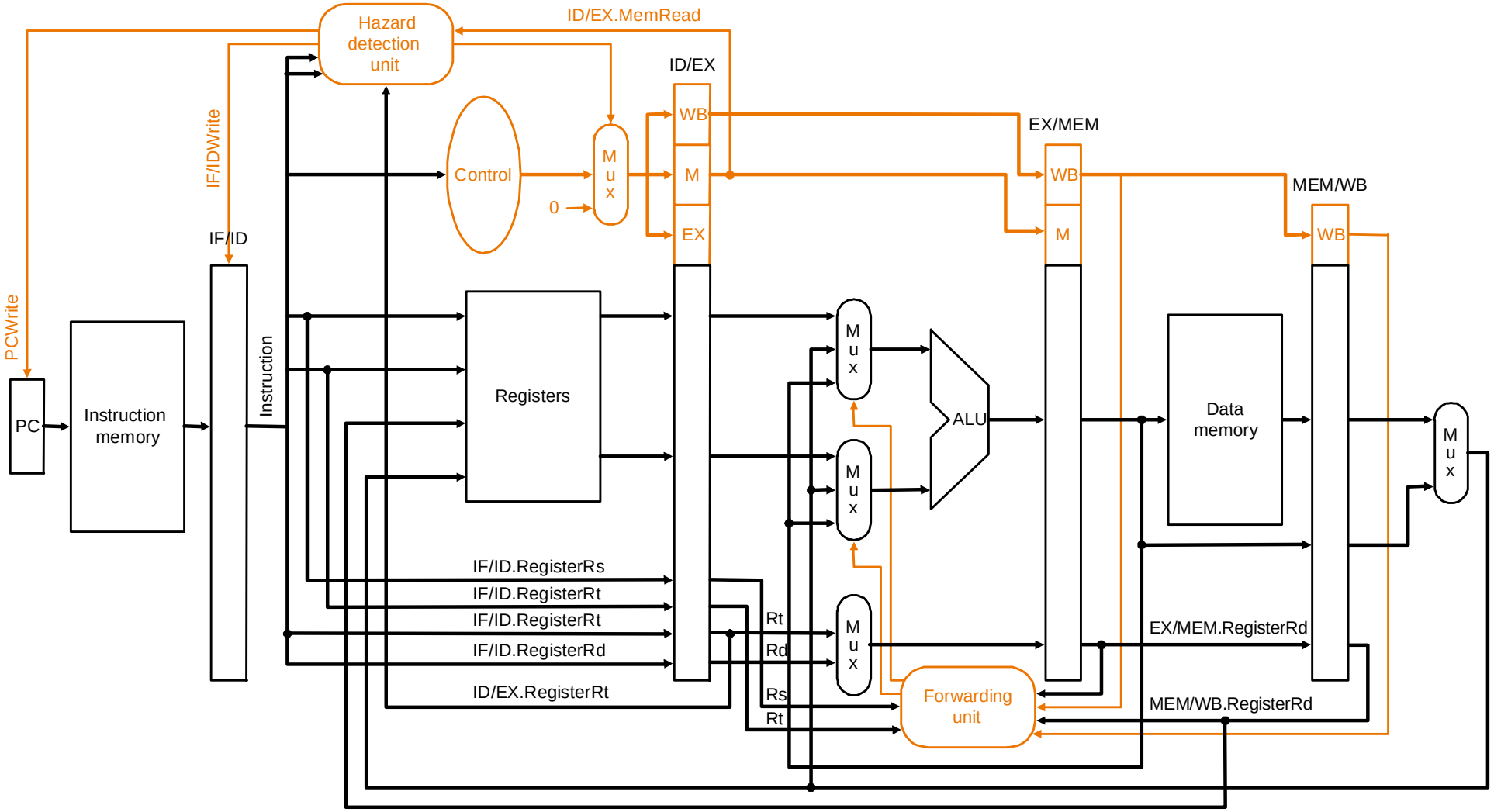


Figure 6.46

Data Dependency Units

Forwarding Condition

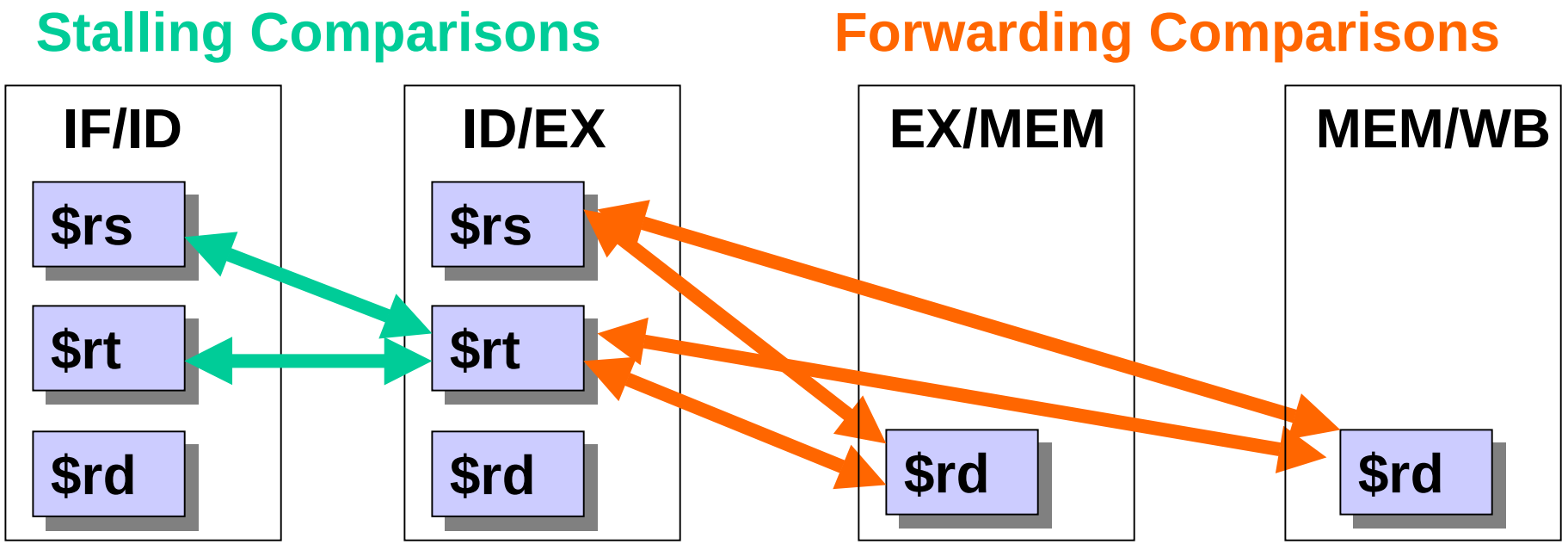
Source		Destination
ID/EX.$rs ID/EX.$rt	}	= EX/MEM.$rd
ID/EX.$rs ID/EX.$rt	}	= MEM/WB.$rd

Stall Condition

Source		Destination
IF/ID.$rs IF/ID.$rt	}	= ID/EX.$rt Λ ID/EX.MemRead=1

Data Dependency Units

Pipeline Registers



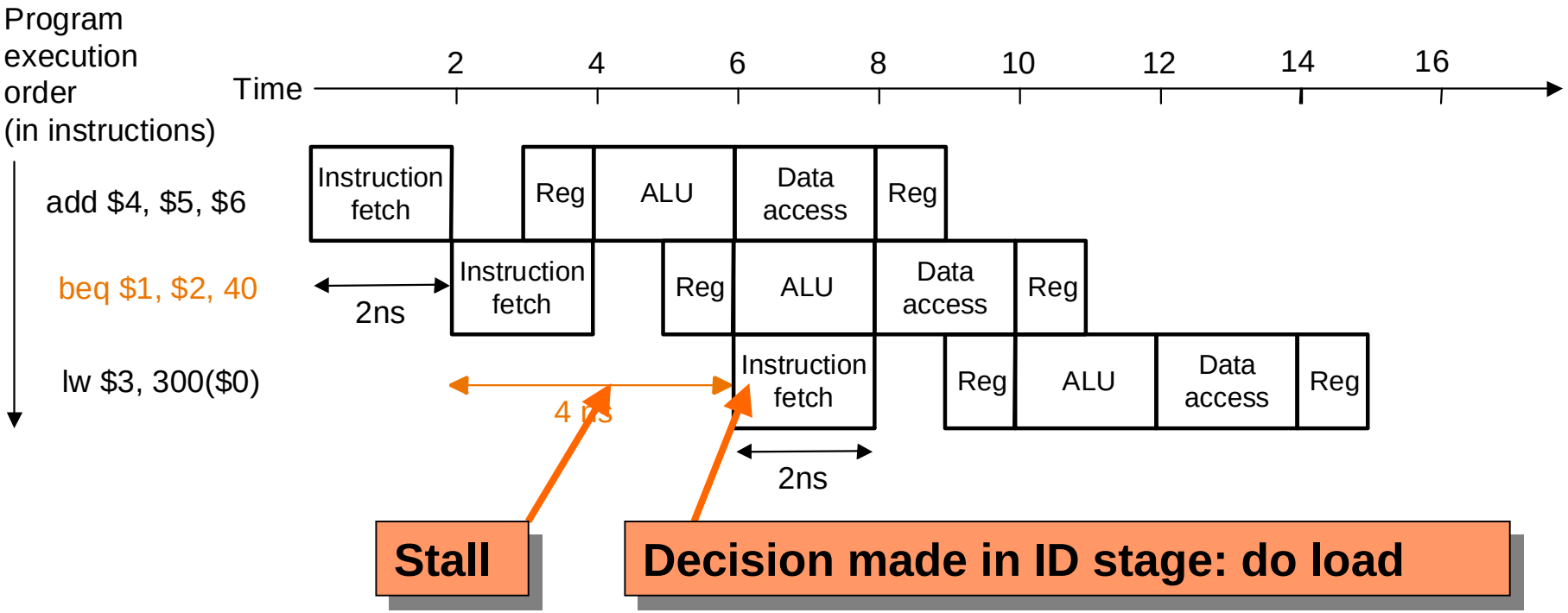
Stall Condition

Source		Destination
IF/ID.$rs IF/ID.$rt	} =	ID/EX.$rt ∧ ID/EX.MemRead=1

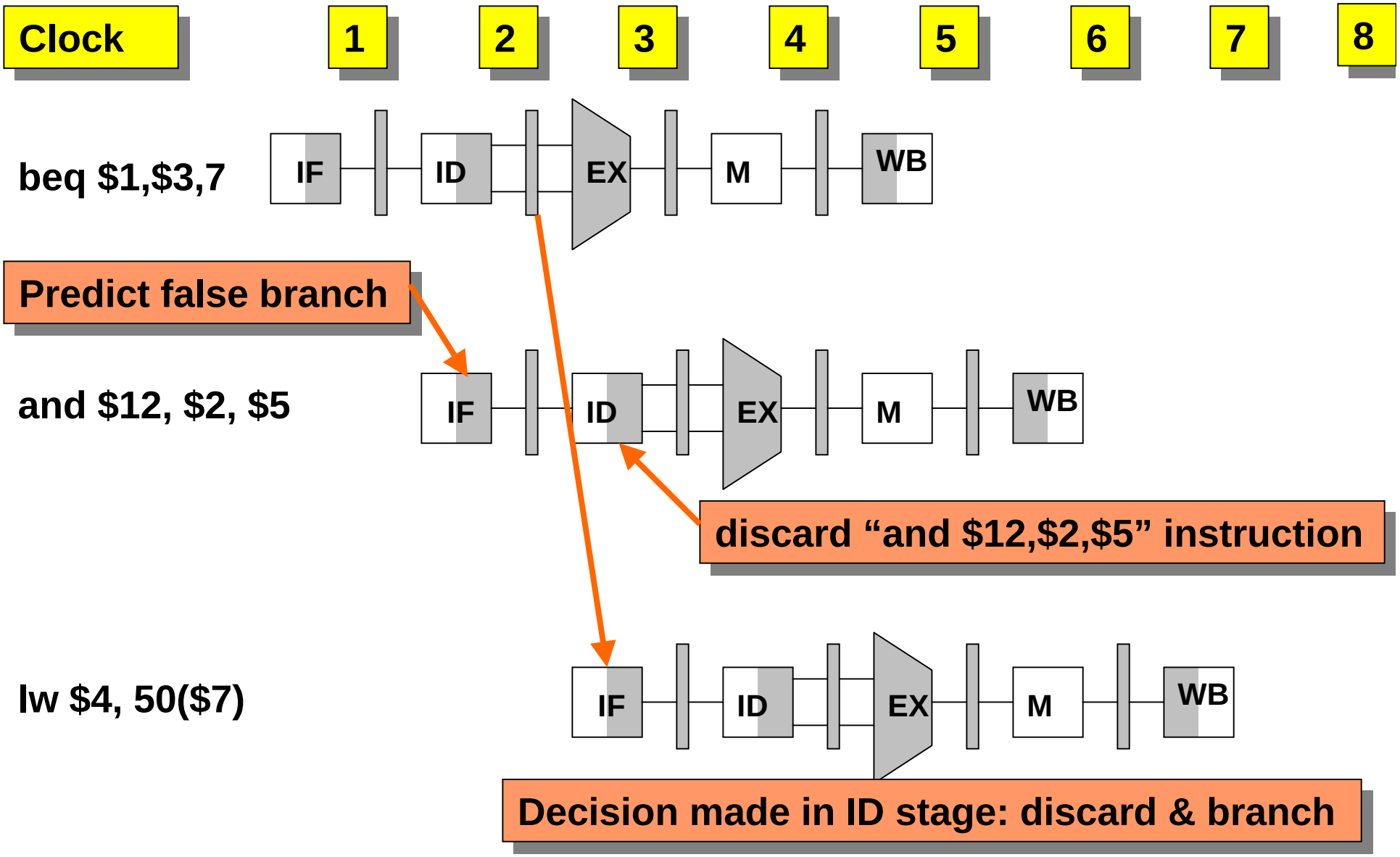
Branch Hazards: Soln #1, Stall until Decision made (fig. 6.4)

```
@3C:    add   $4, $5, $6
@40:    beq   $1, $3, 7
@44:    and   $12, $2, $5
@48:    or    $13, $6, $2
@4C:    add   $14, $2, $2
@50:    lw    $4, 50($7)
```

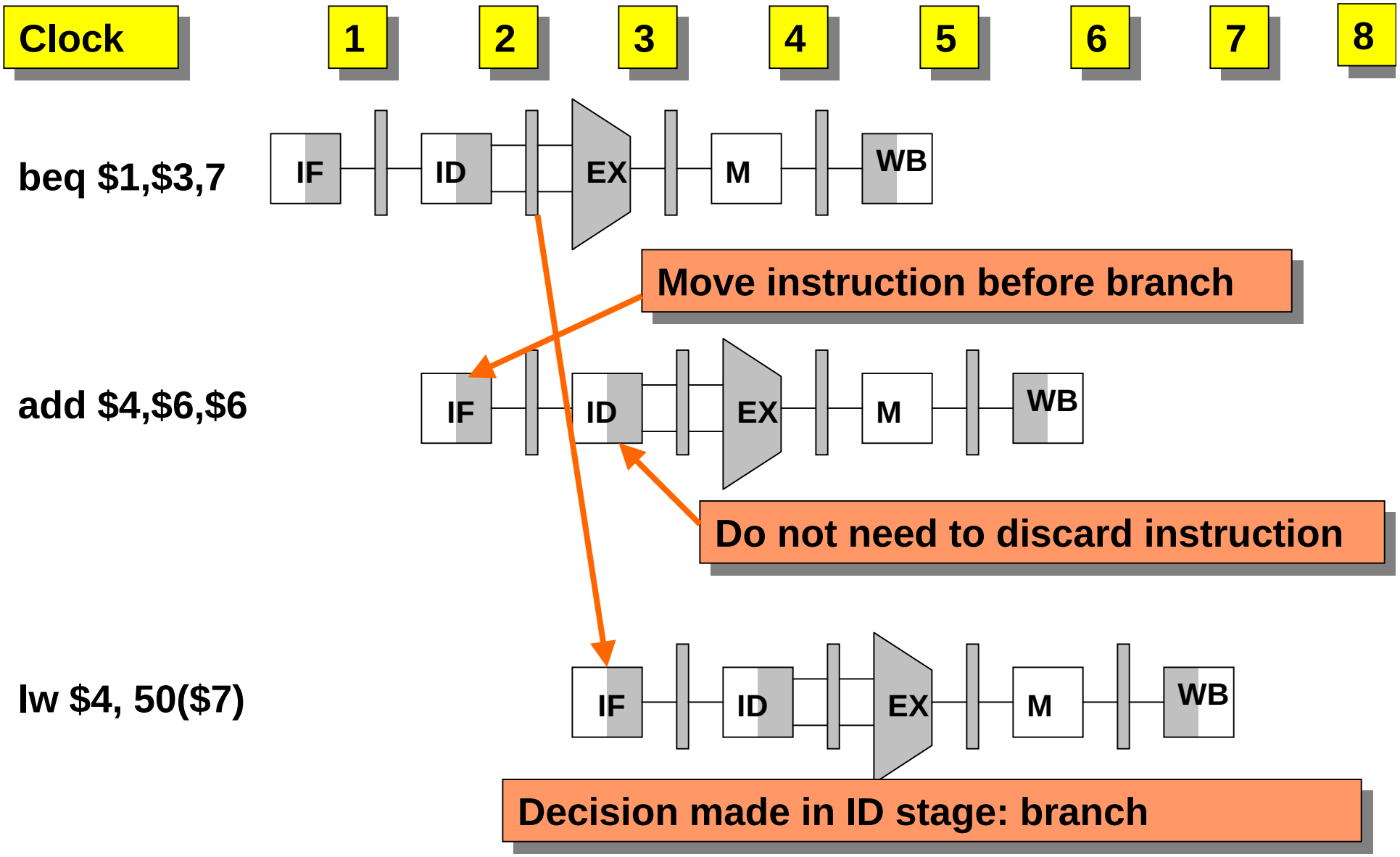
Soln #1: Stall until Decision is made



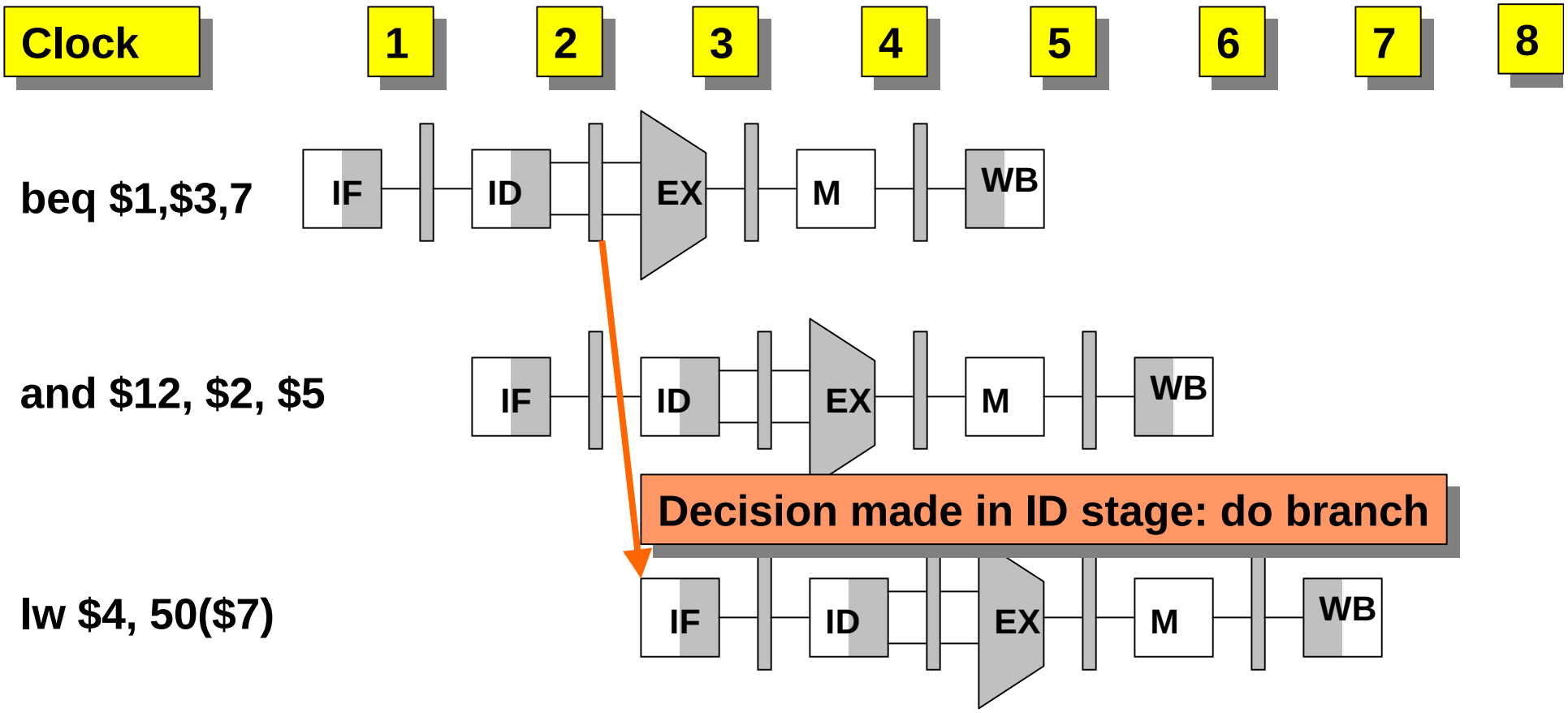
Branch Hazards: Soln #2, Predict until Decision made



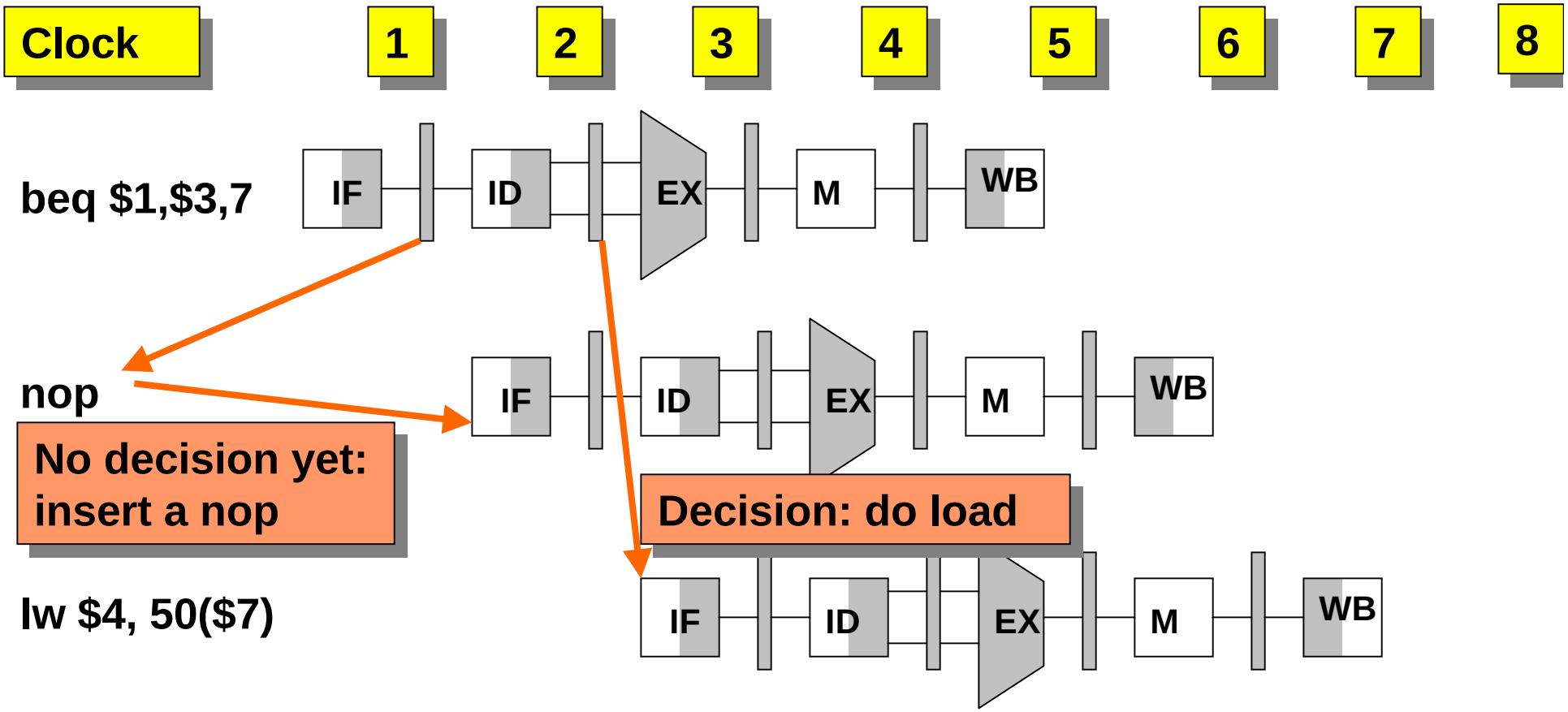
Branch Hazards: Soln #3, Delayed Decision



Branch Hazards: Soln #3, Delayed Decision



Branch Hazards: Decision made in the ID stage (figure 6.4)



Branch Hazards: Soln #2, Predict until Decision made

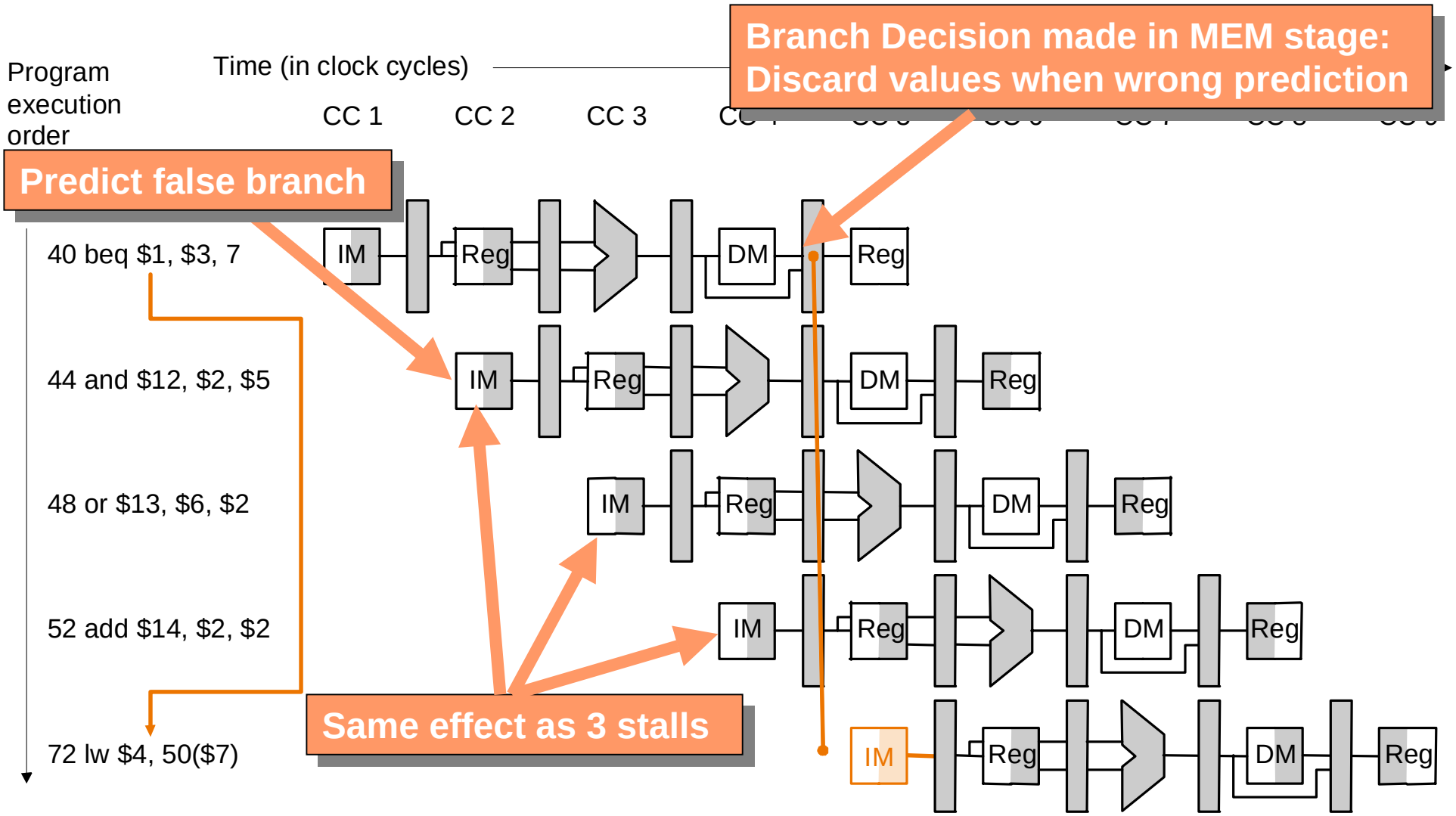
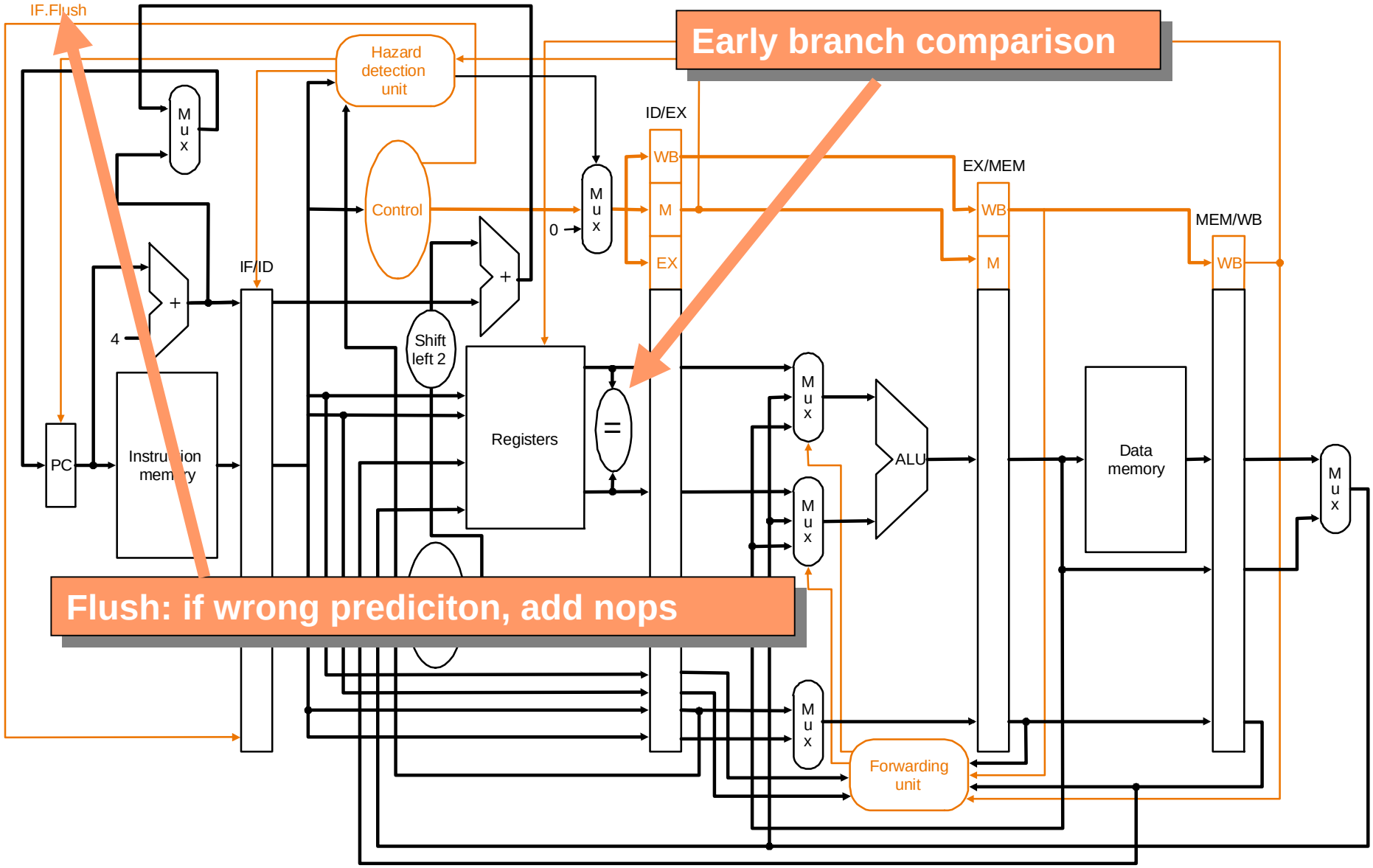


Figure 6.50

Figure 6.51


IF.Flush
Early branch comparison
Hazard detection unit
M u x
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
0
IF/ID
+
4
Shift left 2
PC
Instruction memory
Registers
=
ALU
Data memory
Flush: if wrong prediciton, add nops
Forwarding unit

Performance

load: assume half of the instructions are immediately followed by an instruction that uses it (i.e. data dependency)

load instruction time = 50%*(1 clock) + 50%*(2 clocks)=1.5

Jump: assume that jumps always pay 1 full clock cycle delay (stall). Jump instruction time = 2

Branch: the branch delay of misprediction is 1 clock cycle

that 25% of the branches are mispredicted.

branch time = 75%*(1 clocks) + 25%*(2 clocks) = 1.25

Performance, page 504

Also known as the instruction latency with in a pipeline

Pipeline throughput

Instruction	Single-Cycle	Multi-Cycle Clocks	Pipeline Cycles	Instruction Mix
loads	1	5	1.5 (50% dependancy)	23%
stores	1	4	1	13%
arithmetic	1	4	1	43%
branches	1	3	1.25 (25% dependancy)	19%
jumps	1	3	2	2%
Clock speed	125 Mhz 8 ns	500 Mhz 2 ns	500 Mhz 2 ns	
CPI	1	4.02	1.18	= Σ Cycles*Mix
MIPS	125 MIPS	125 MIPS	424 MIPS	= Clock/CPI

load instruction time = 50%*(1 clock) + 50%*(2 clocks)=1.5

branch time = 75%*(1 clocks) + 25%*(2 clocks) = 1.25