

CSCI 210: Computer Architecture

Lecture 29: Pipelining

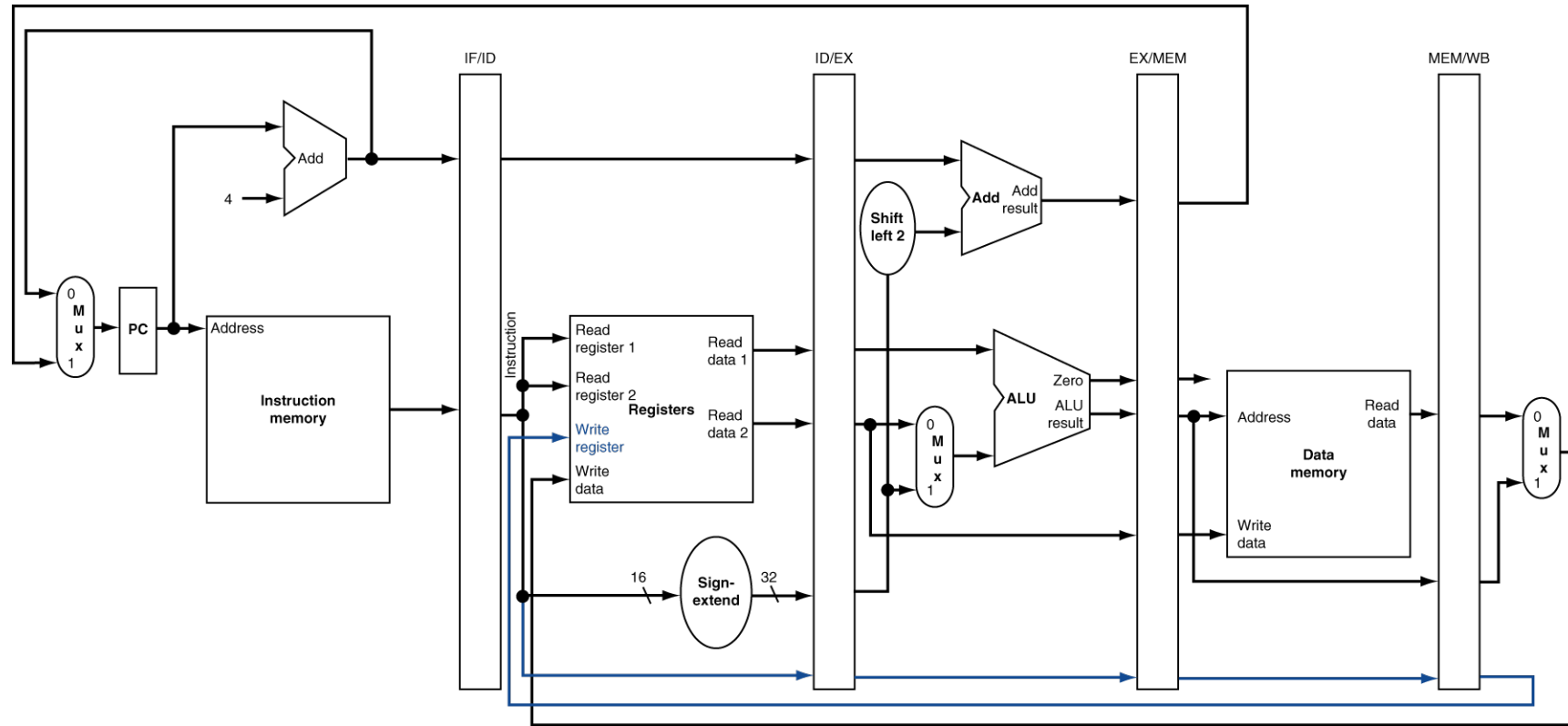
Stephen Checkoway

Slides from Cynthia Taylor

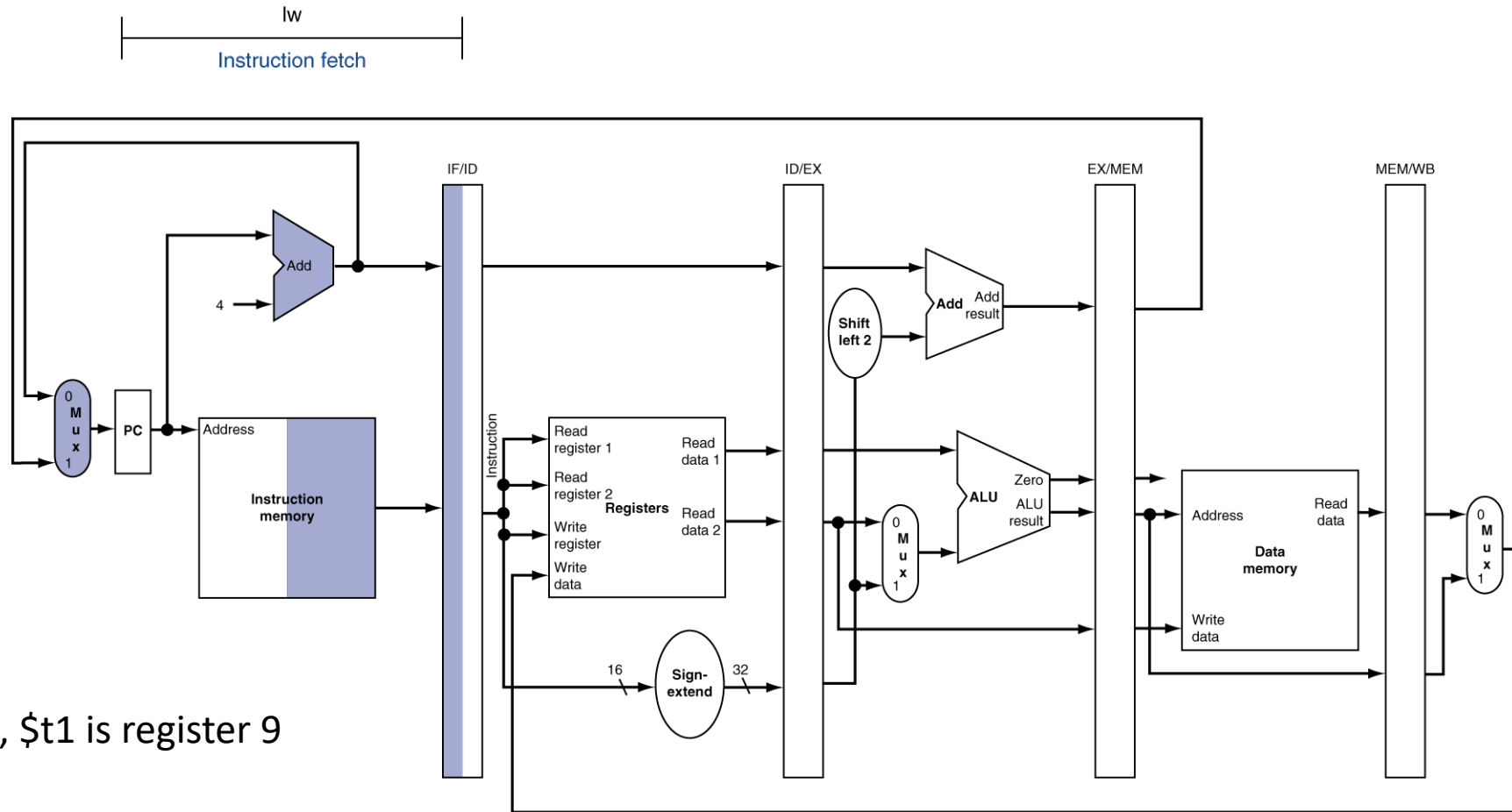
CS History: Berkeley RISC

- Developed by David Patterson at UC Berkeley between 1980 and 1984
- Patterson took a sabbatical to improve DEC's Complex Instruction Set, and instead decided the whole system was bad
- A 1978 Andrew Tannenbaum paper had shown a 10,000 line complex program could be implemented using a simplified ISA with an 8-bit fixed opcode
 - And that 81% of constants were 0, 1 or 2!
 - IBM internally discovered similar results
- First RISC chip came out in 1981
- RISC V is currently in active development as an open-source ISA

Pipelined Datapath



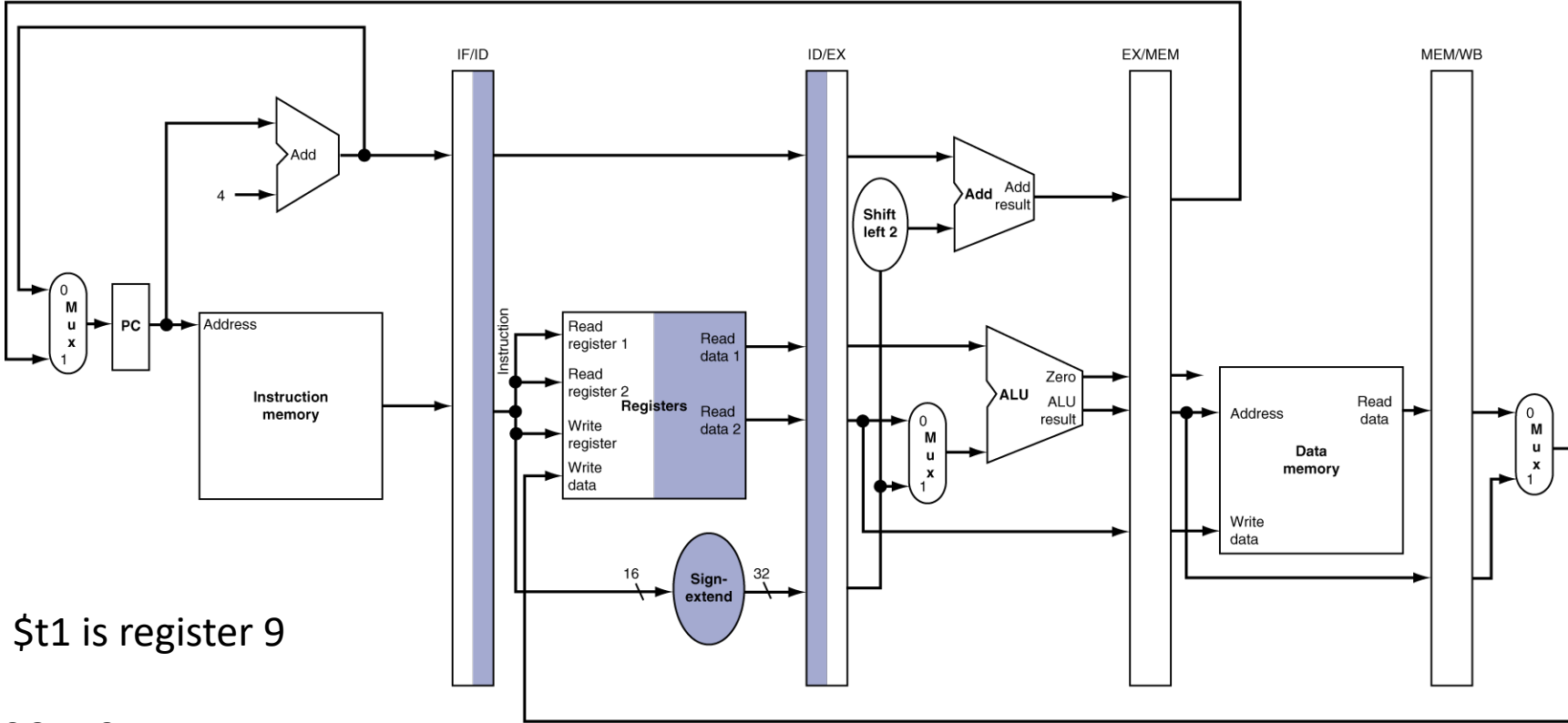
IF for sw \$t0, 4(\$t1)



\$t0 is register 8, \$t1 is register 9
\$t0 holds 5
\$t1 holds 0x4810CAB0
0x4810CAB0 holds 12

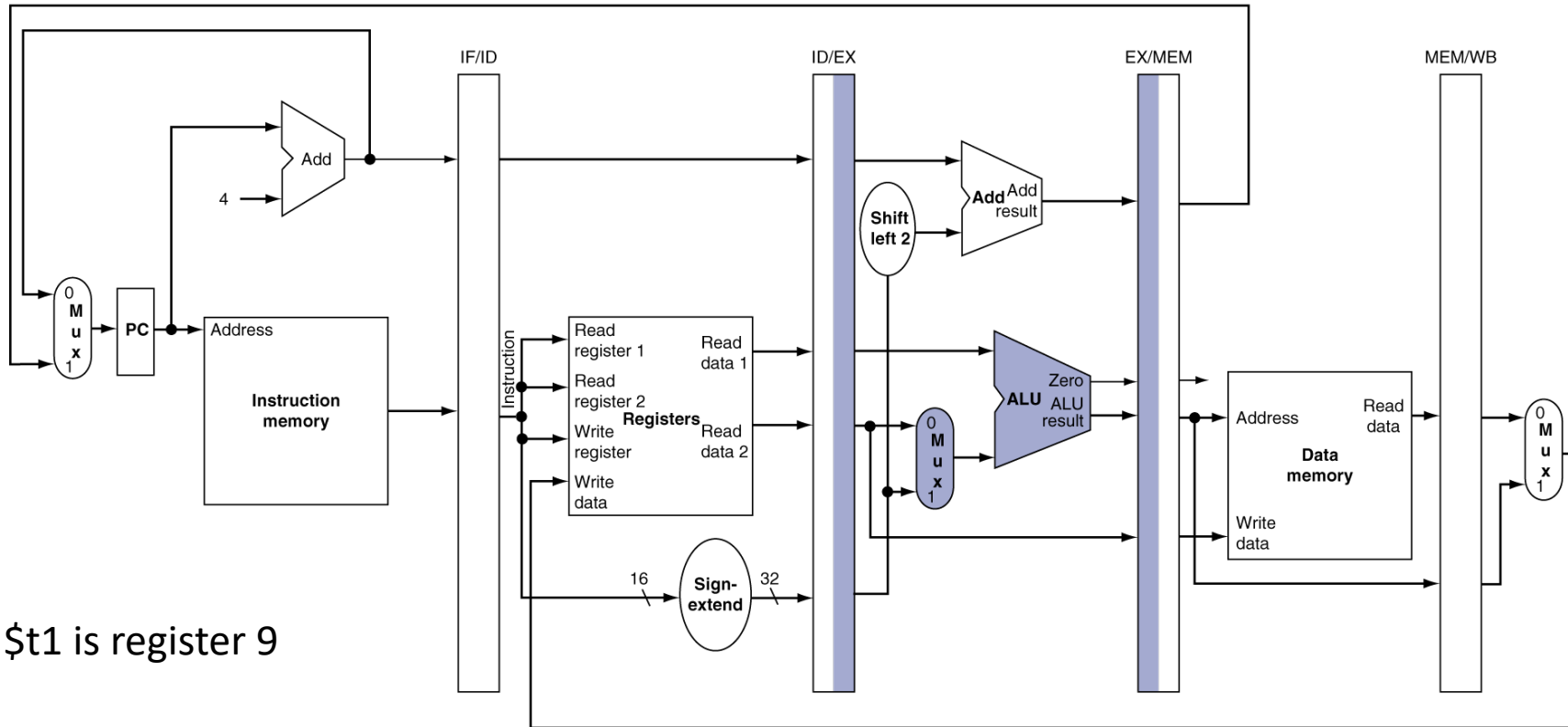
ID for sw \$t0, 4(\$t1)

lw
Instruction decode



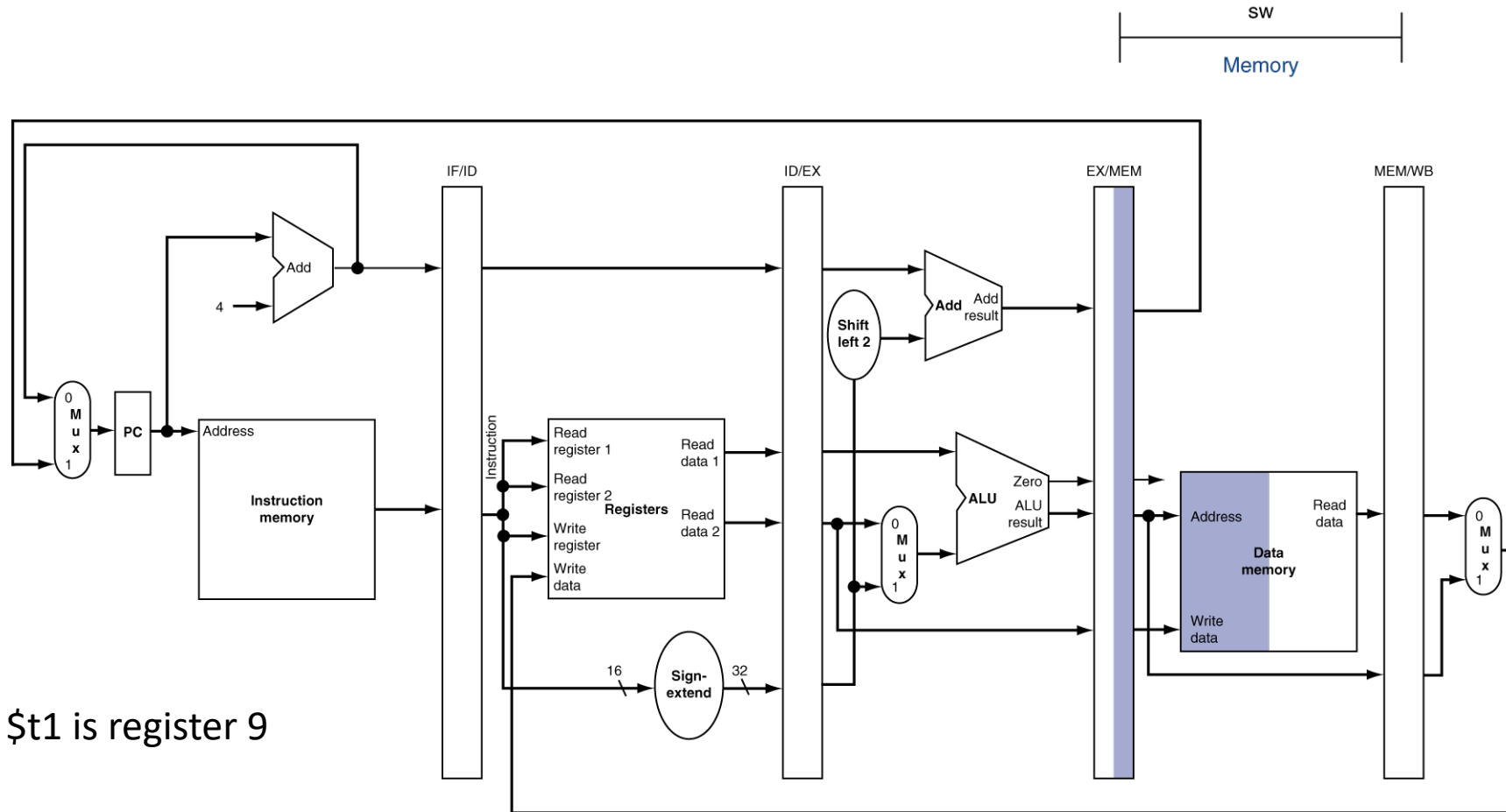
\$t0 is register 8, \$t1 is register 9
 \$t0 holds 5
 \$t1 holds 0x4810CAB0
 0x4810CAB0 holds 12

EX for sw \$t0, 4(\$t1)



\$t0 is register 8, \$t1 is register 9
\$t0 holds 5
\$t1 holds 0x4810CAB0
0x4810CAB0 holds 12

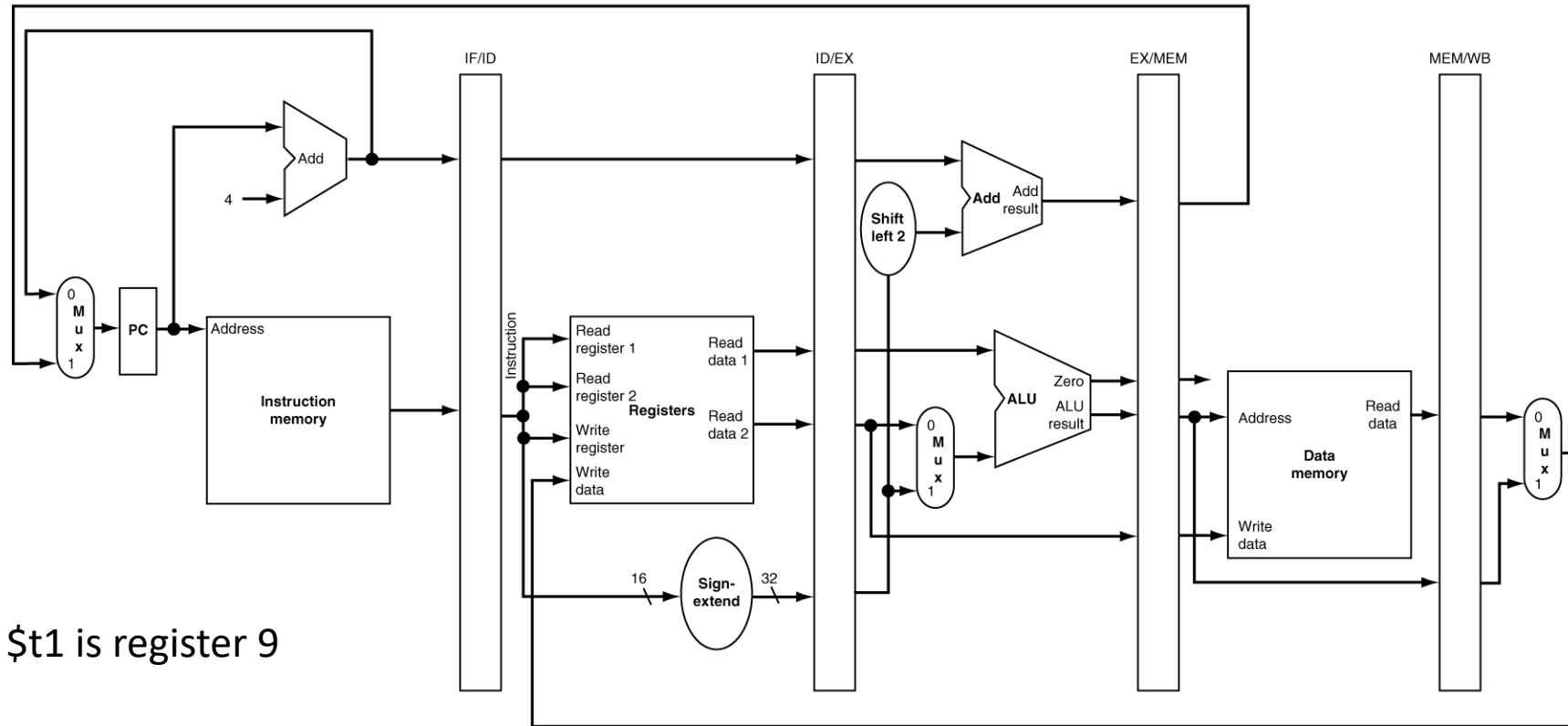
MEM for sw \$t0, 4(\$t1)



\$t0 is register 8, \$t1 is register 9
\$t0 holds 5
\$t1 holds 0x4810CAB0
0x4810CAB0 holds 12

WB for sw \$t0, 4(\$t1)

SW
Write-back



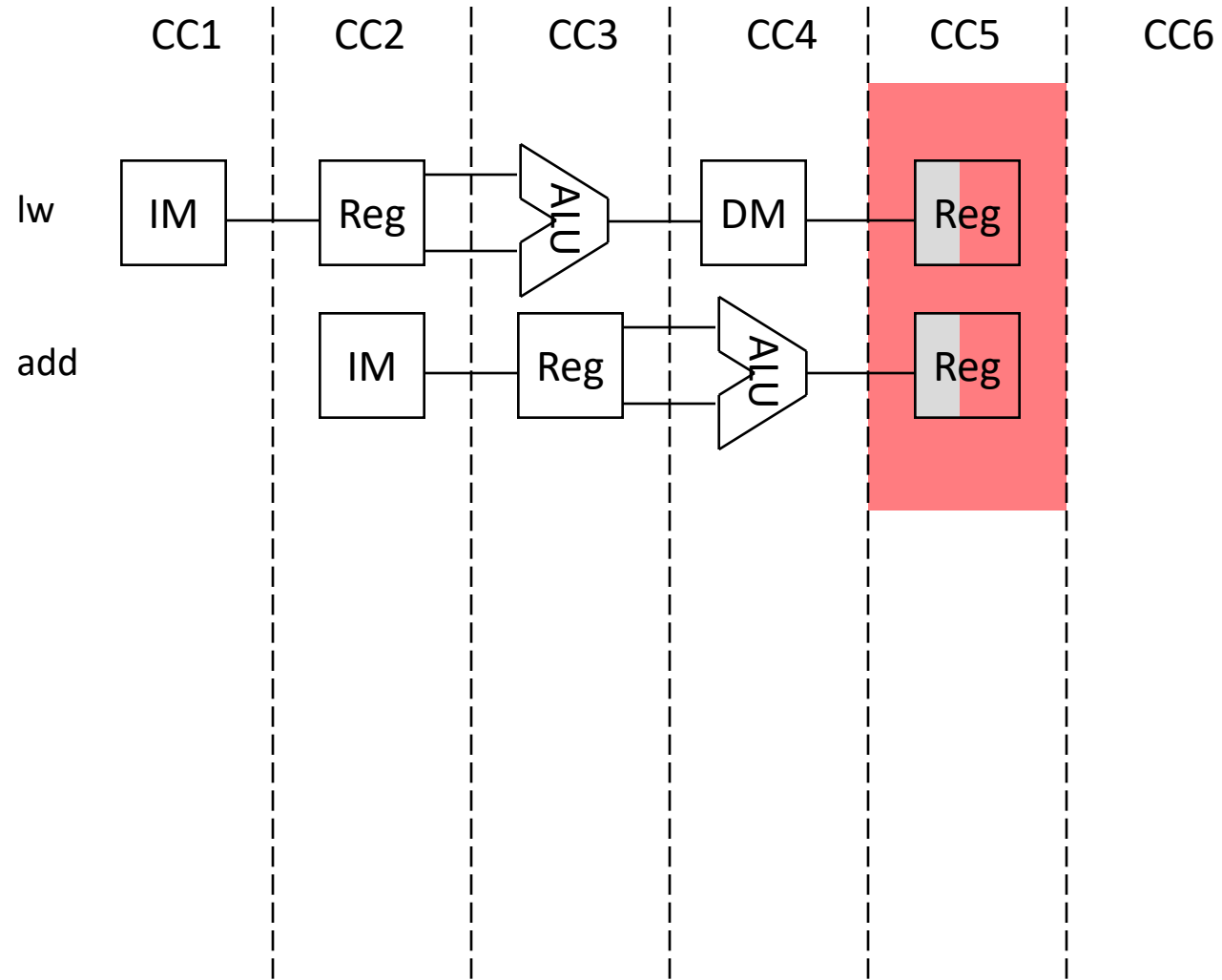
\$t0 is register 8, \$t1 is register 9
\$t0 holds 5
\$t1 holds 0x4810CAB0
0x4810CAB0 holds 12

Pipeline Stages

Should we force every instruction to go through all 5 stages? Can we break it up, with R-type taking 4 cycles instead of 5?

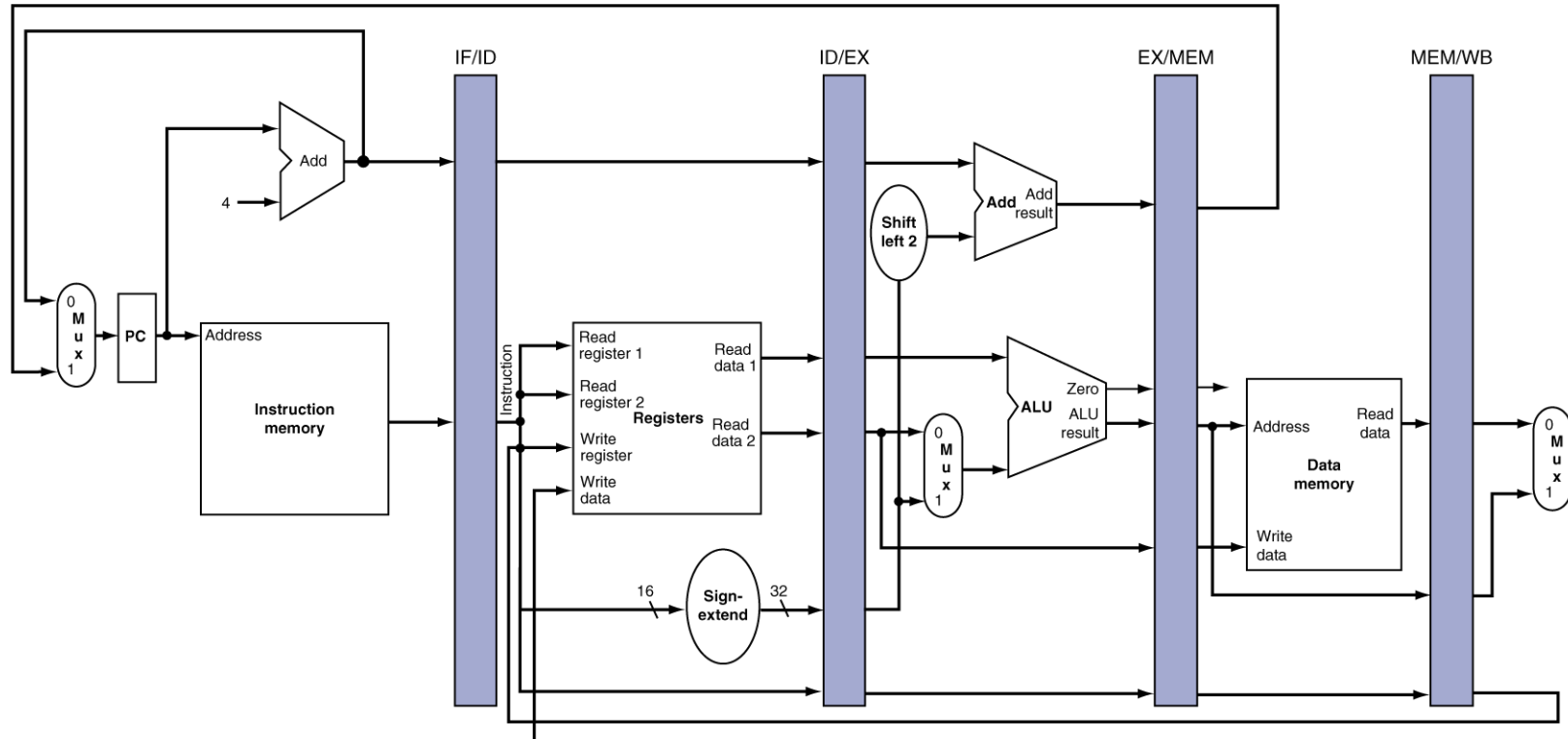
Selection	Yes/No	Reason (Choose BEST answer)
A	Yes	Decreasing R-type to 4 cycles improves instruction throughput
B	Yes	Decreasing R-type to 4 cycles improves instruction latency
C	No	Decreasing R-type to 4 cycles causes hazards
D	No	Decreasing R-type to 4 cycles causes hazards and doesn't impact throughput
E	No	Decreasing R-type to 4 cycles causes hazards and doesn't impact latency

Mixed Instructions in the Pipeline

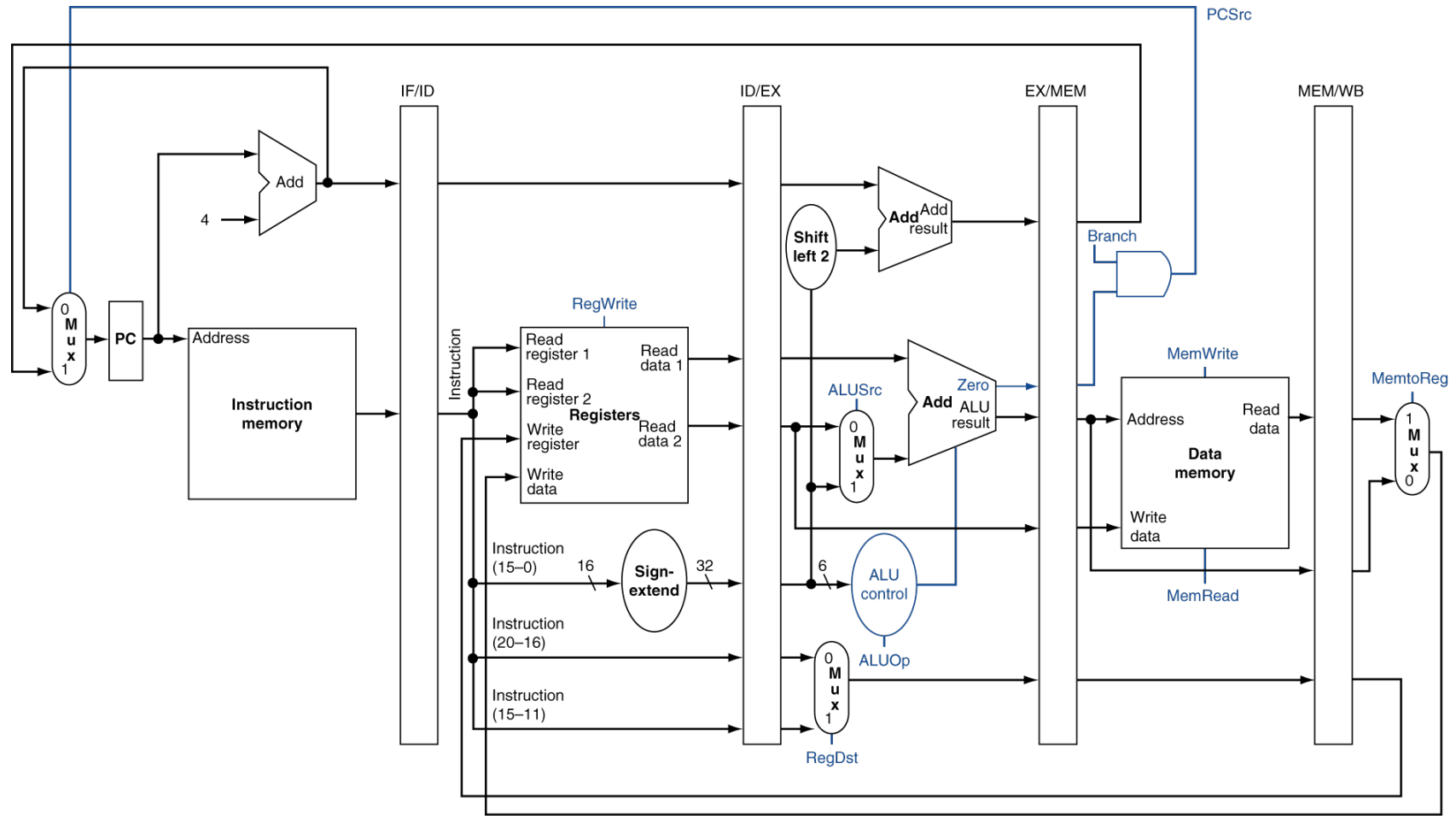


State of pipeline in a given cycle

add \$14, \$5, \$6	lw \$13, 24(\$1)	add \$12, \$3, \$4	sub \$11, \$2, \$3	lw \$10, 20(\$1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



Pipelined Control



How do we control our pipelined CPU?

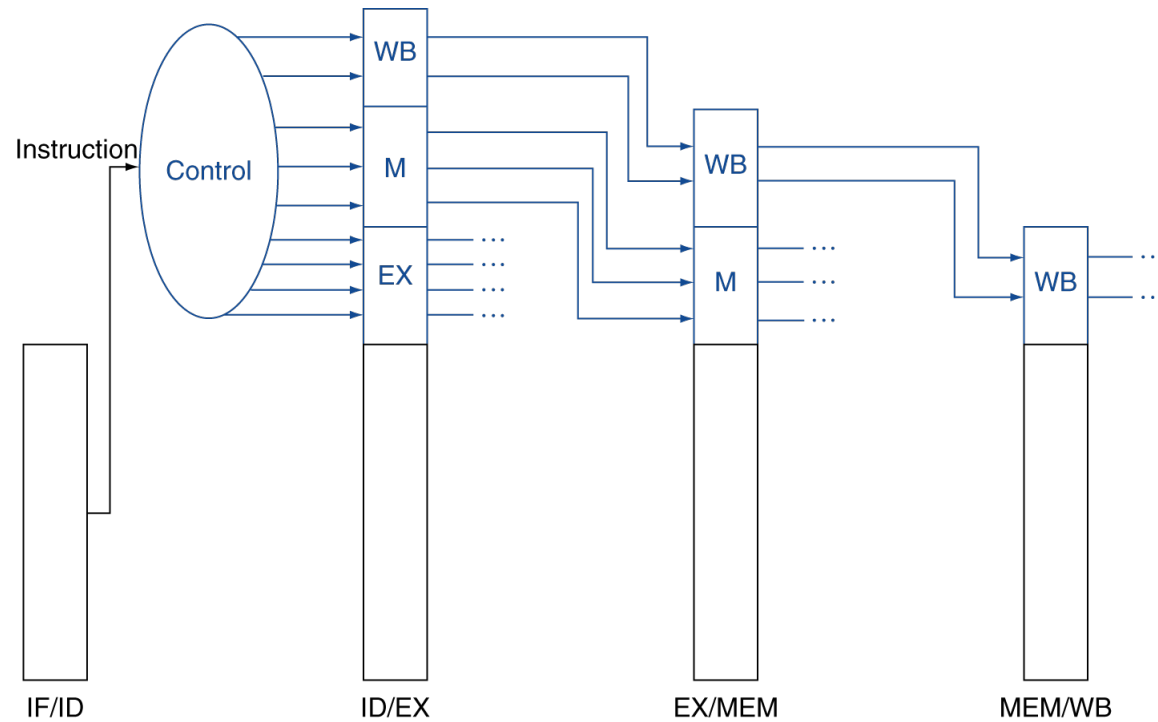
- A. We need to add new control signals.
- B. We need to forward the control values to the correct stage.
- C. We don't need to do anything special; it will work the way it is.

Pipeline Control

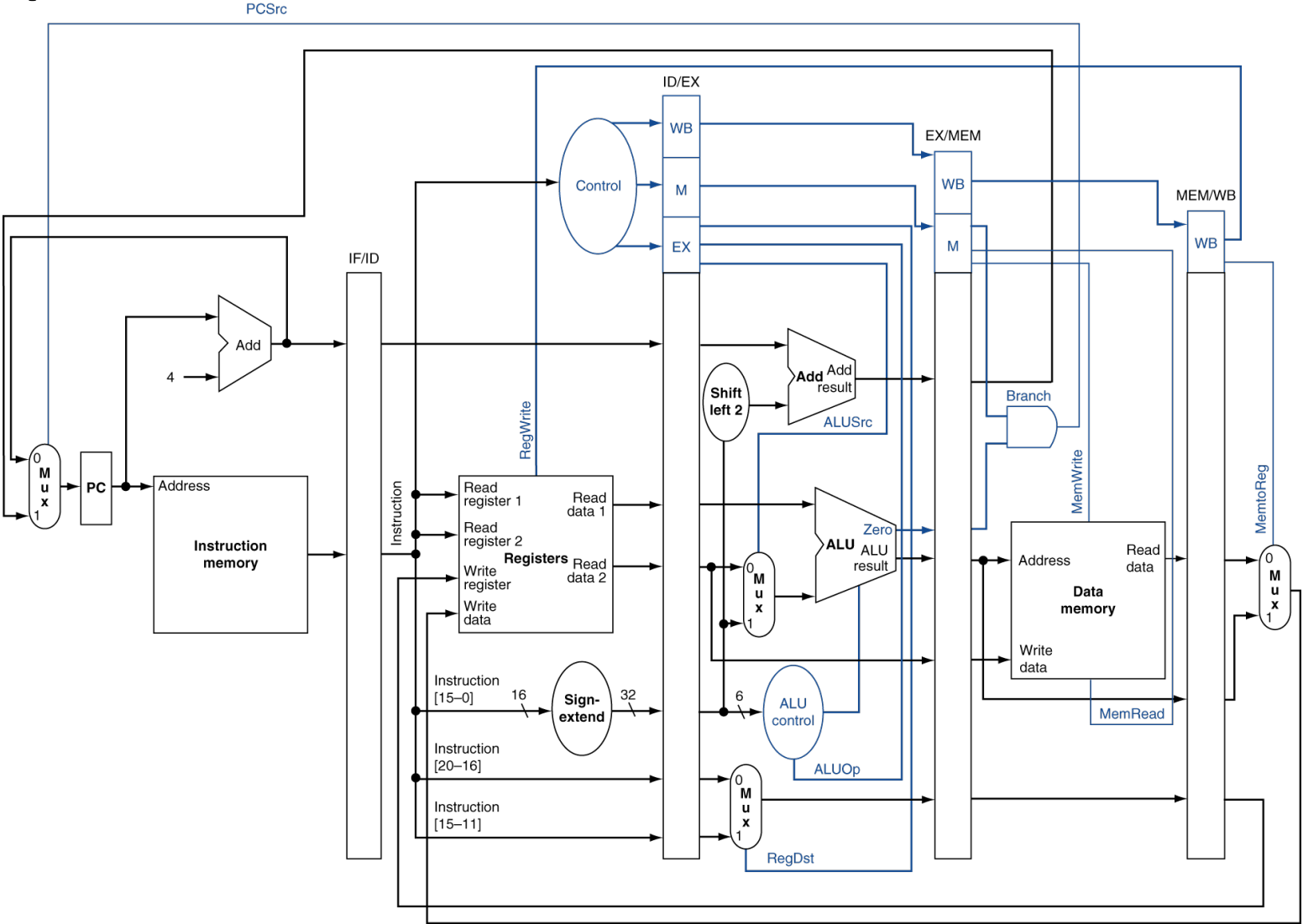
- IF Stage: read Instr Memory (always) and write PC + 4
- ID Stage: compute all control signals for subsequent stages
- EX, MEM, and WB stages have control signals
 - The pipeline registers will need to store the control signals

Pipelined Control

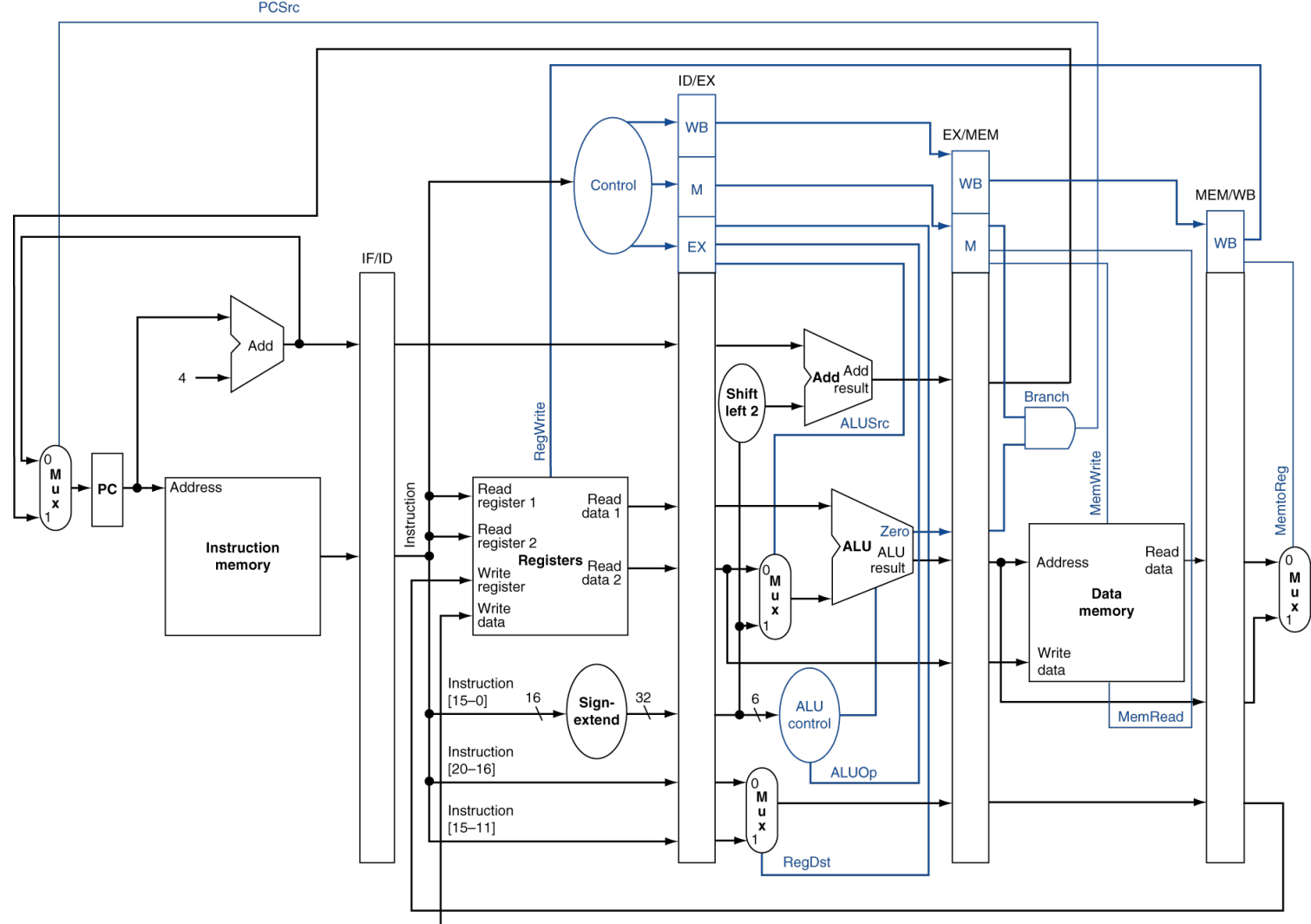
Control signals derived from instruction



Pipelined Control: add \$t0, \$t1, \$t2



\$t1 holds 5
\$t2 holds 6



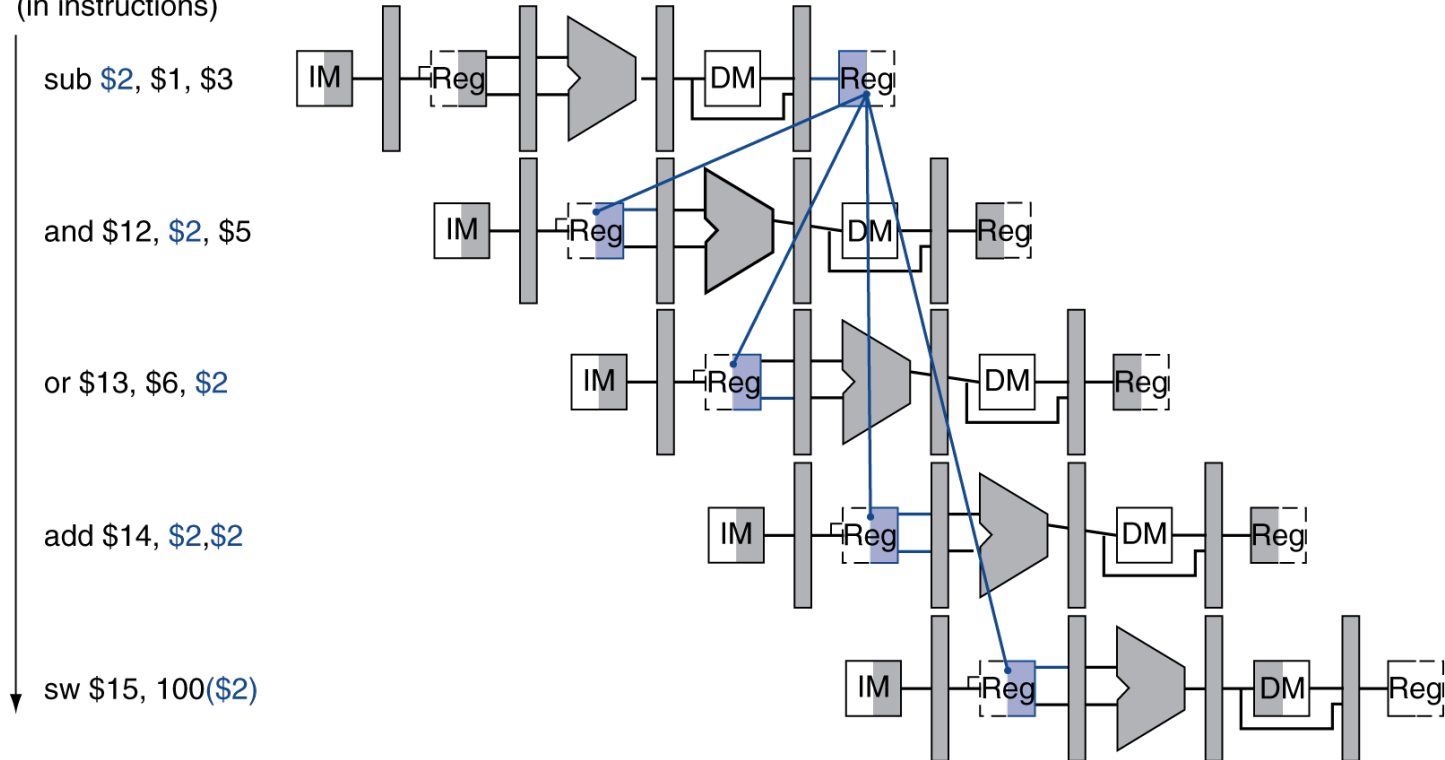
	EX Stage				MEM Stage			WB Stage	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Brch	MemRead	MemWrite	RegWrite	Mem toReg
R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Questions on Pipeline Control?

Dependencies & Forwarding

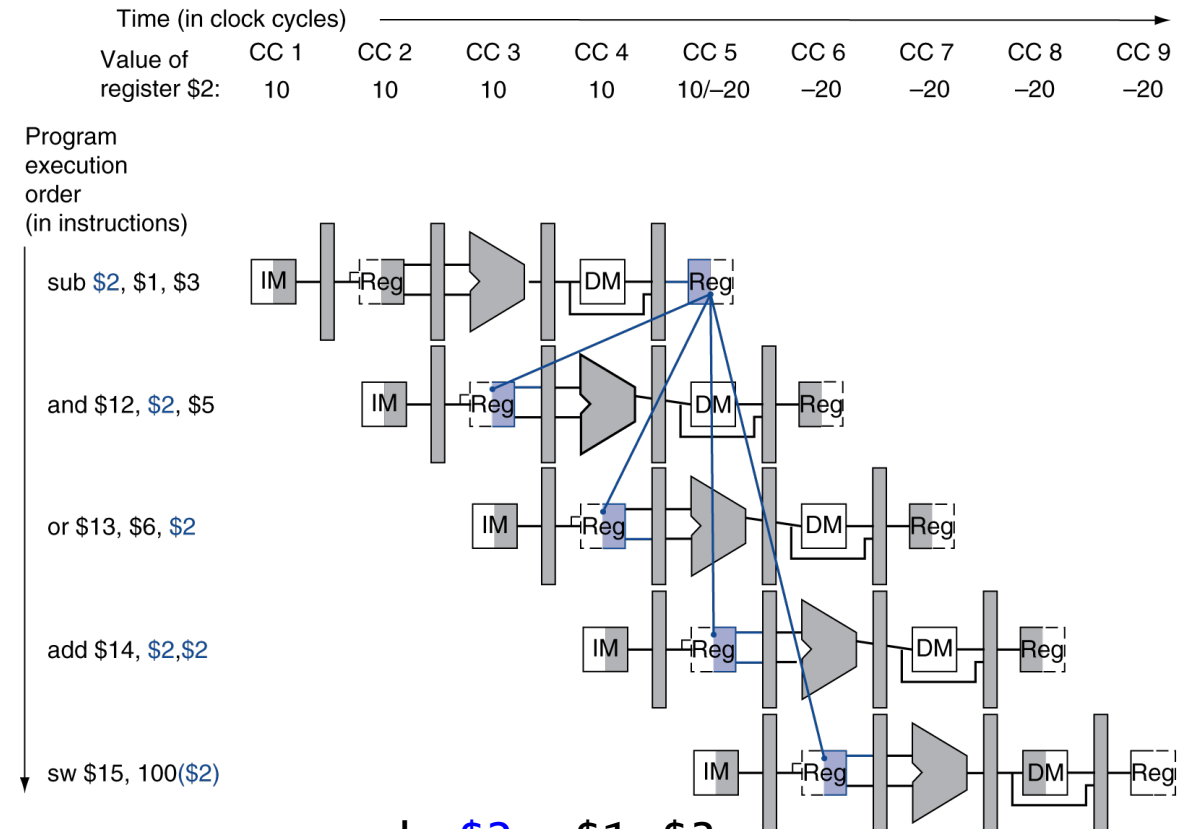
	Time (in clock cycles) →								
Value of register \$2:	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
	10	10	10	10	10/-20	-20	-20	-20	-20

Program execution order (in instructions)



We can best solve **these** data hazards

- A. By stalling.
- B. By forwarding.
- C. By combining forwards and stalls.
- D. By doing something else.



```

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

Data Hazards in ALU Instructions

- Consider this sequence:

```
sub $2, $1, $3
```

```
and $12, $2, $5
```

```
or $13, $6, $2
```

```
add $14, $2, $2
```

```
sw $15, 100($2)
```

- We can resolve hazards with forwarding
 - How do we detect when to forward?
 - How do we actually perform the forward?

Forwarding

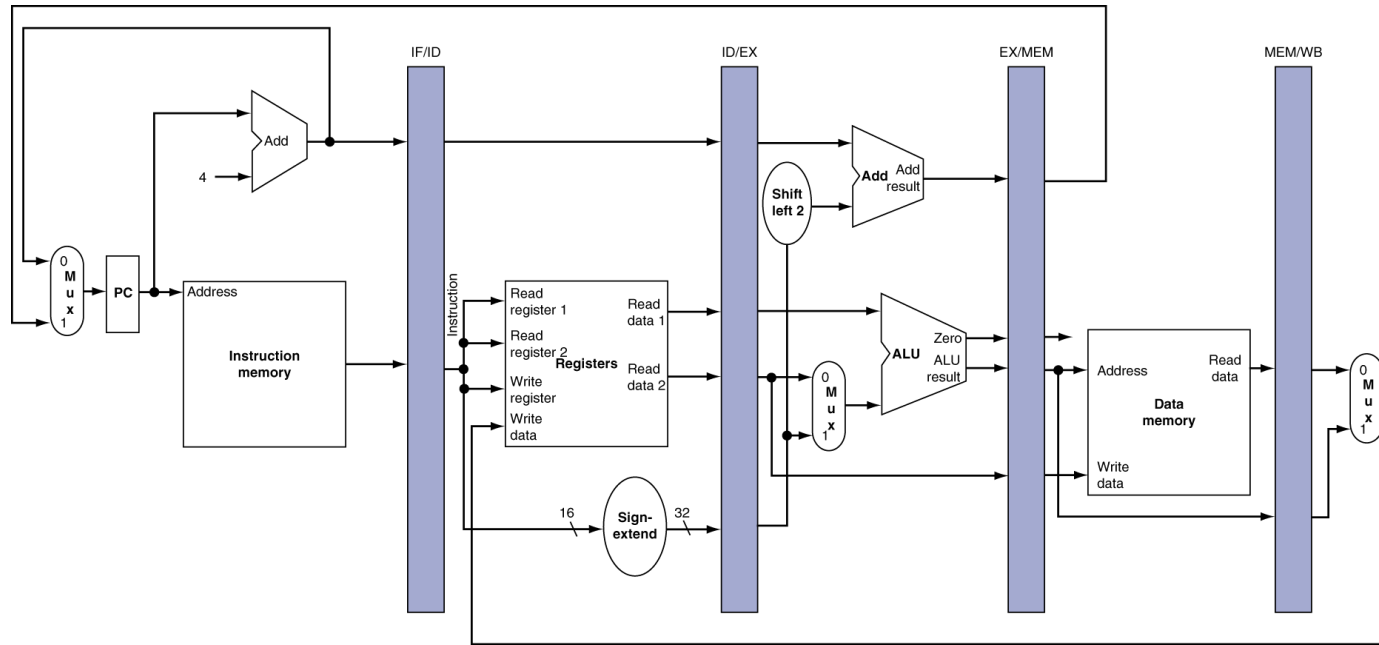
Data path

- Connect the outputs of EX and MEM stages to both ALU inputs controlled by multiplexers

Control path

- Pass rs, rt, and rd register numbers through the pipeline registers
- Add a forwarding unit to control the multiplexers
 - Depends on RegWrite and rs/rt/rd from various stages

Detecting the Need to Forward



- Data hazards when
 - 1a. $EX/MEM.RegisterRd = ID/EX.RegisterRs$
 - 1b. $EX/MEM.RegisterRd = ID/EX.RegisterRt$
 - 2a. $MEM/WB.RegisterRd = ID/EX.RegisterRs$
 - 2b. $MEM/WB.RegisterRd = ID/EX.RegisterRt$

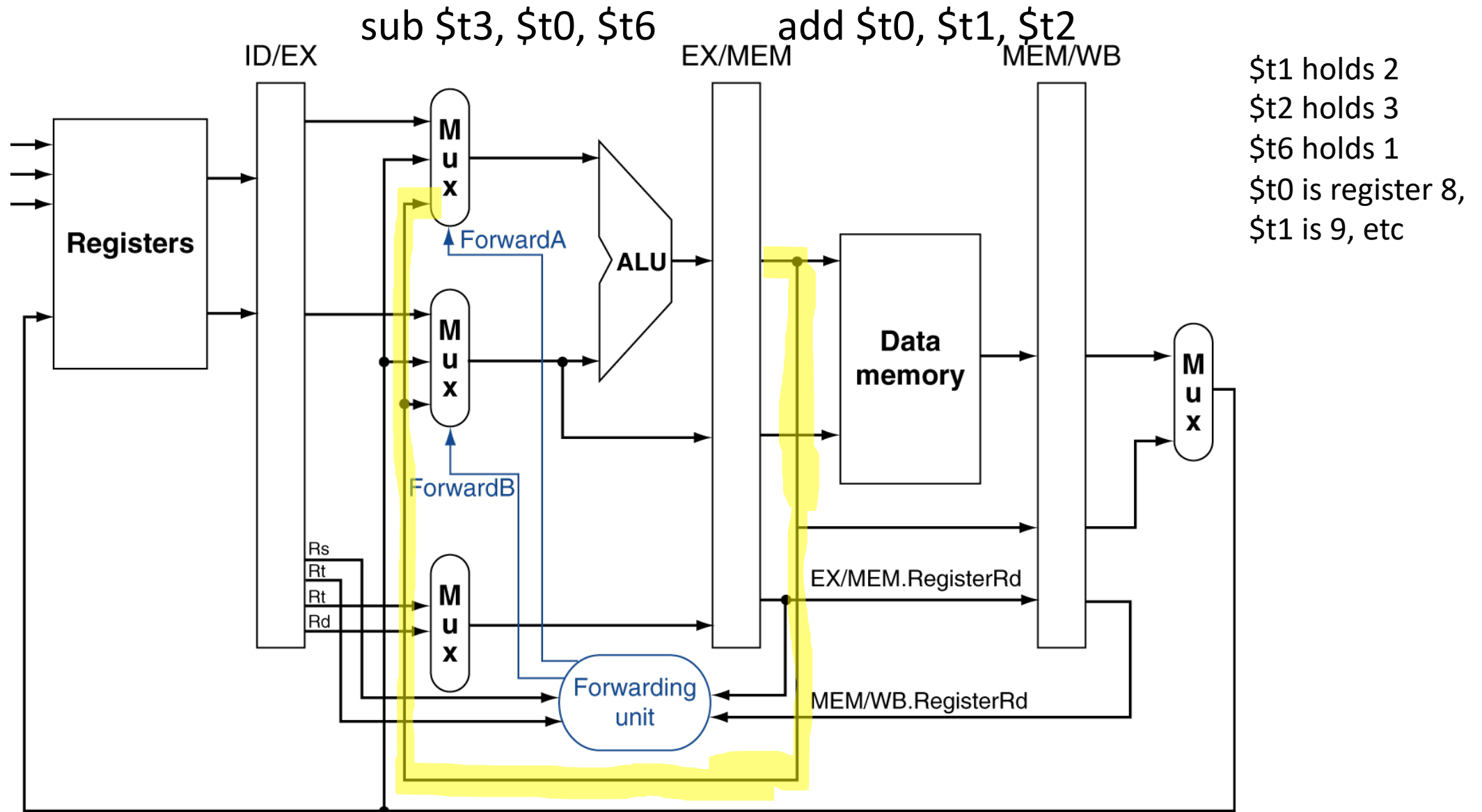
Fwd from
EX/MEM
pipeline reg

Fwd from
MEM/WB
pipeline reg

Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
 - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not \$zero
 - EX/MEM.RegisterRd \neq 0,
MEM/WB.RegisterRd \neq 0

Forwarding Paths

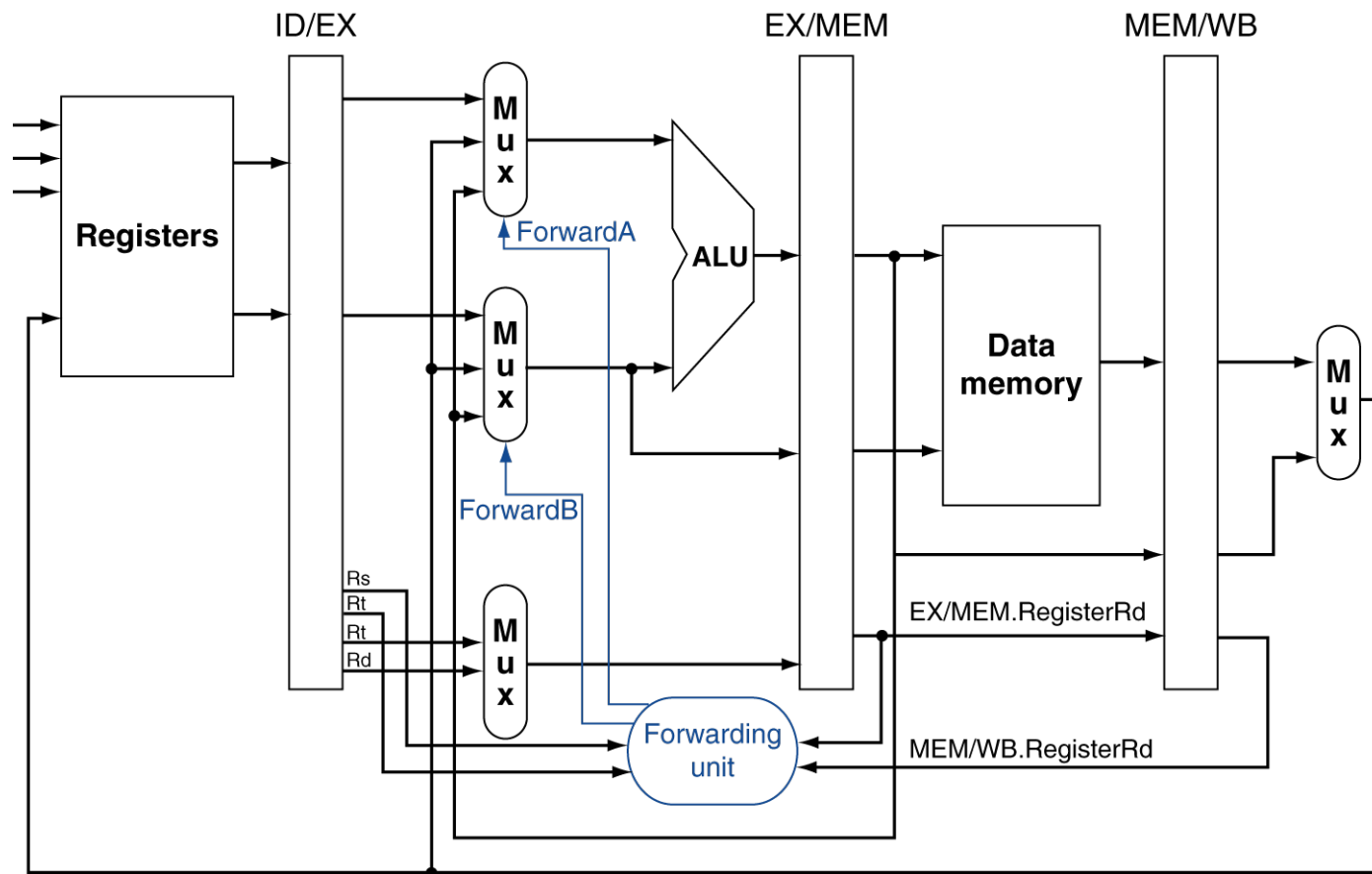


b. With forwarding

If EX/MEM.RegisterRd = MEM/WB.RegisterRd = rs (i.e., both pipeline registers contain a value that will be written to the same register that's about to be used for the ALU), which value should be used by the ALU?

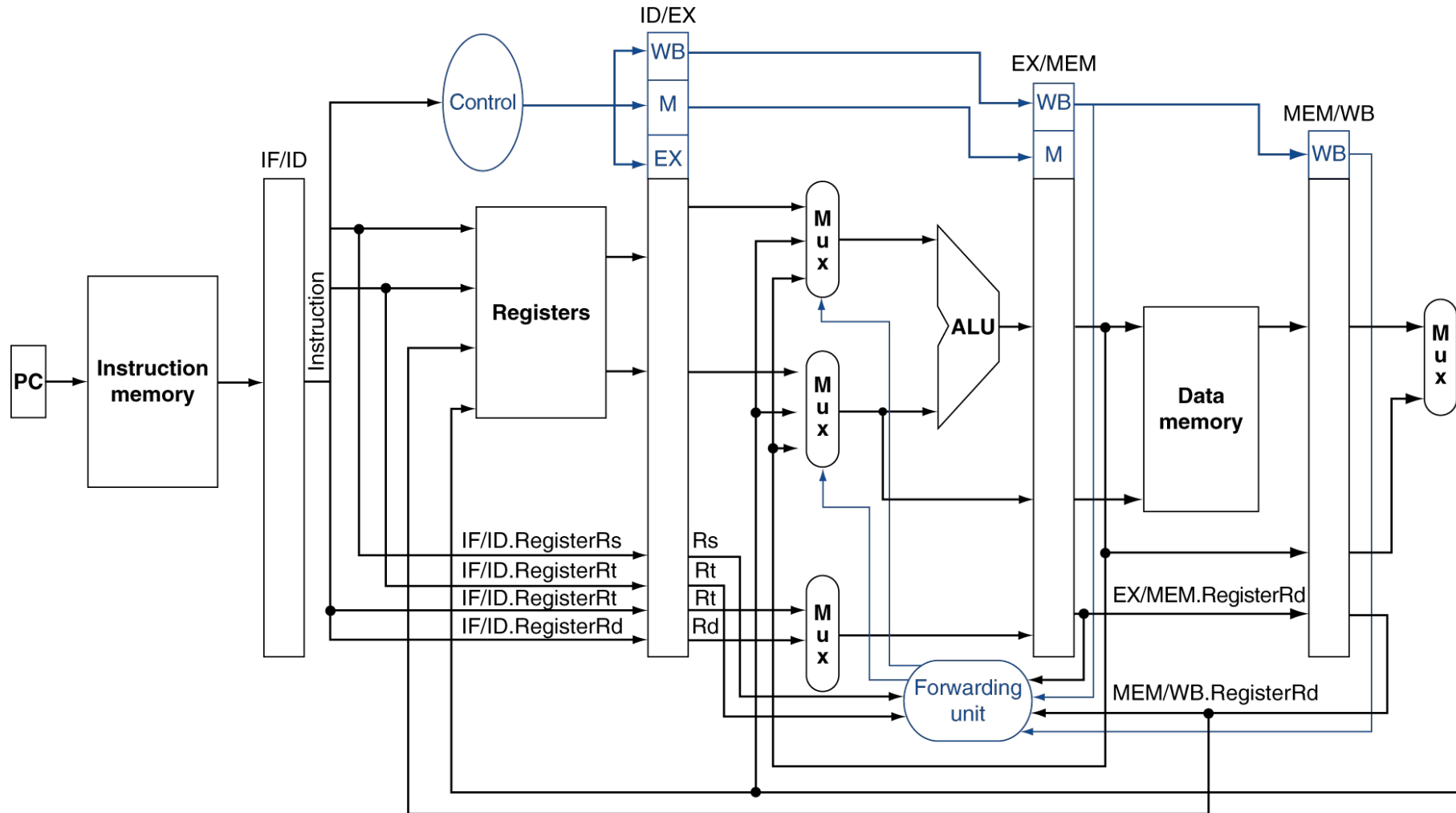
```
add $t1, $t0, $t2
sub $t1, $t1, $t6
add $t8, $t1, $t7
```

- A. The one in EX/MEM
- B. The one in MEM/WB
- C. Either works since both write to rs
- D. The rs value from the register file



b. With forwarding

Datapath with Forwarding



Reading

- Next lecture: Pipelined Datapath
 - Section 5.7