CSCI 210: Computer Architecture
Lecture 29: More Pipelining

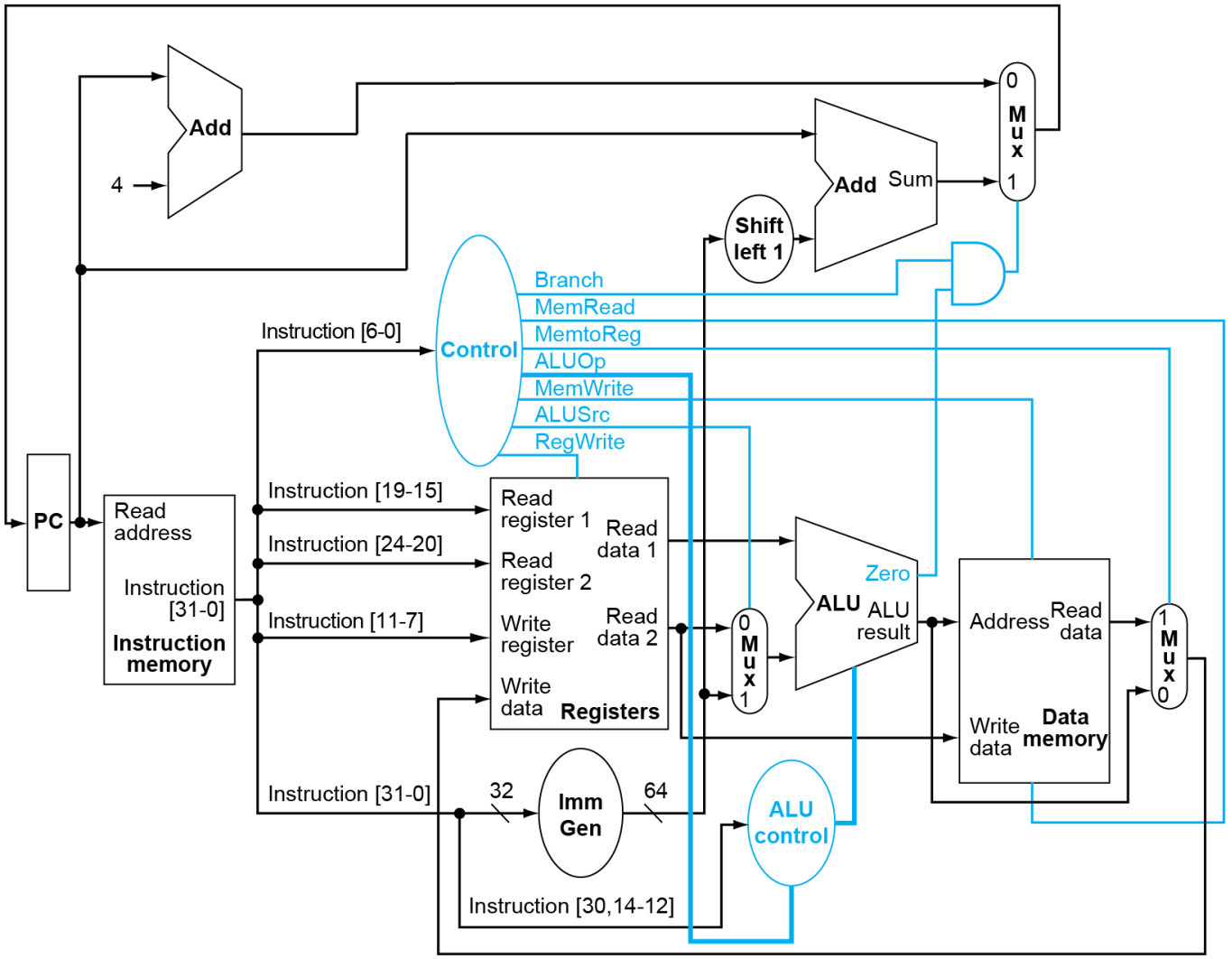
Stephen Checkoway

Slides from Cynthia Taylor

CS History: Berkeley RISC

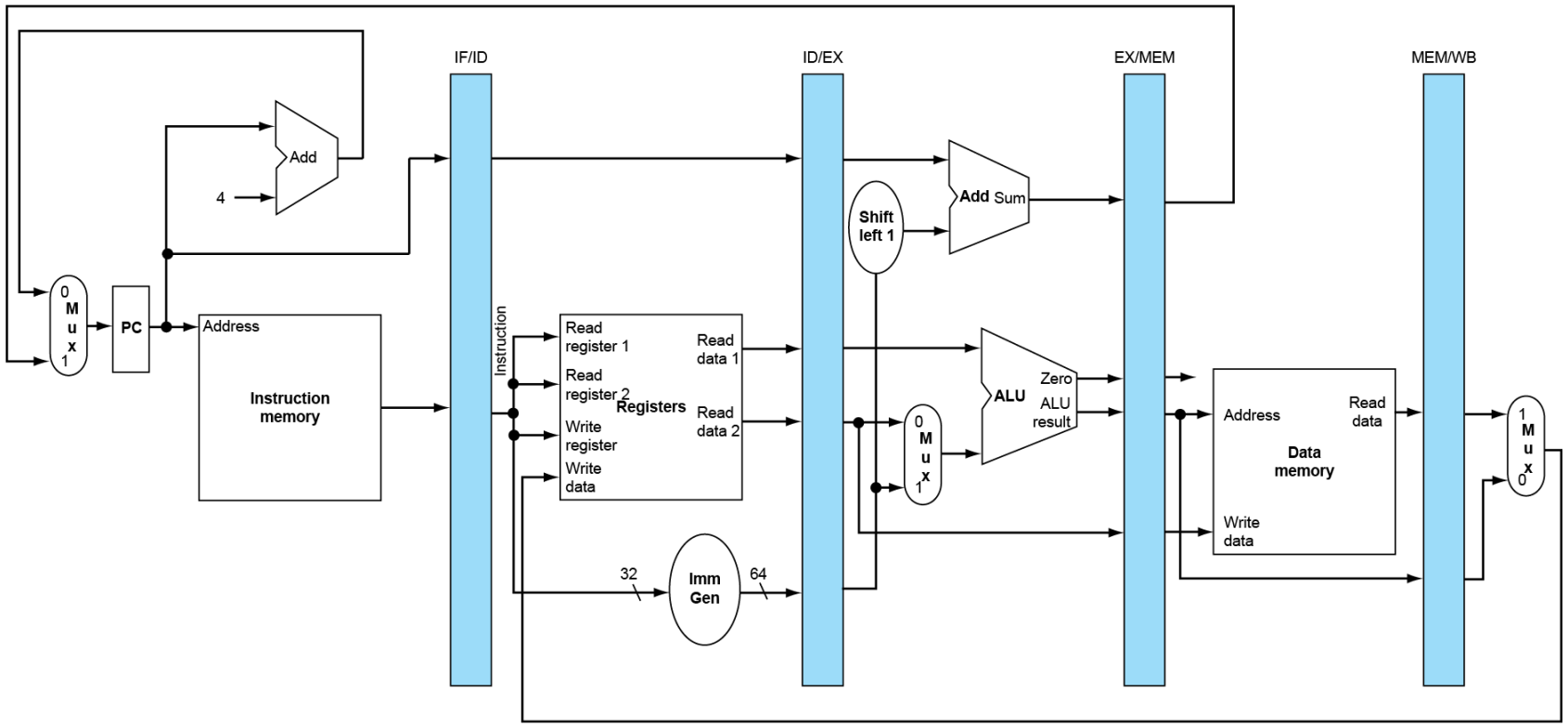
- Developed by David Patterson at UC Berkeley between 1980 and 1984
- Patterson took a sabbatical to improve DEC's Complex Instruction Set, and instead decided the whole system was bad
- A 1978 Andrew Tannenbaum paper had shown a 10,000 line complex program could be implemented using a simplified ISA 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

Single Cycle Data Path

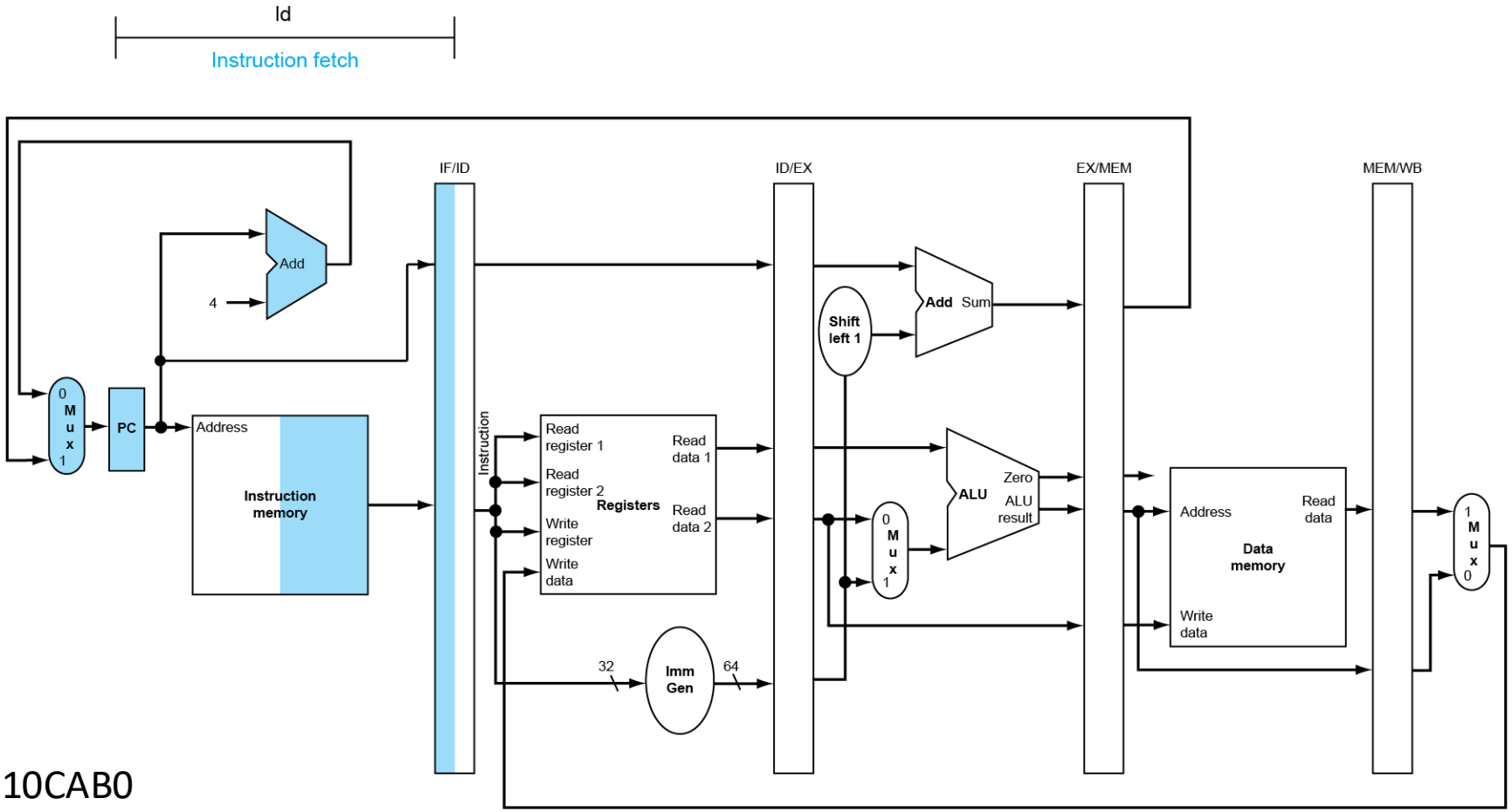


Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle

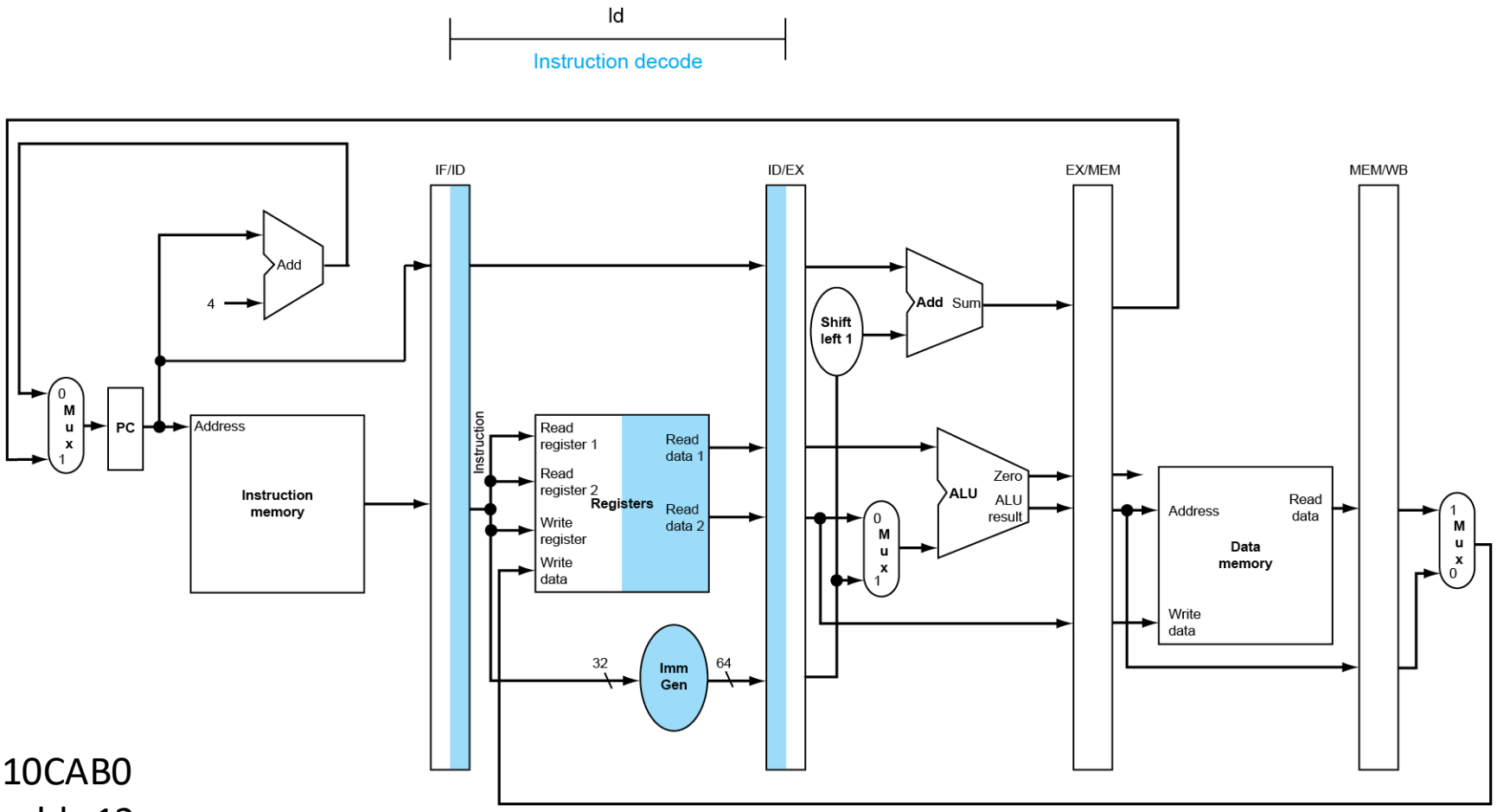


IF for ld x10, 8(x1)



x10 holds 5
x1 holds 0x4810CAB0
0x4810CAB8 holds 12

ID for ld x10, 8(x1)



x10 holds 5
x1 holds 0x4810CAB0
0x4810CAB8 holds 12

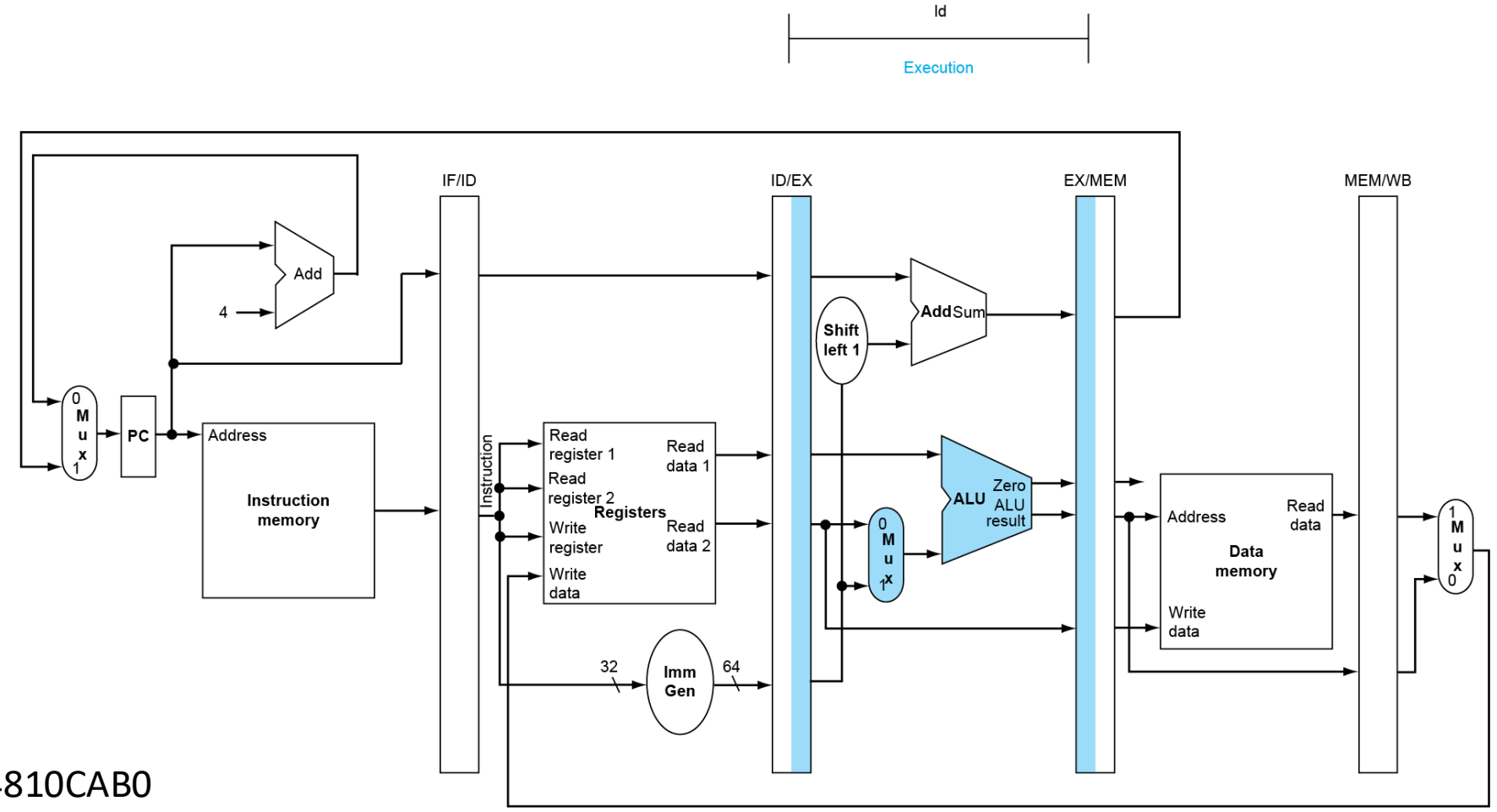
The register file will output data from both read registers, but load will only use one of them. We should

A. Save both of them in ID/EX

B. Only save the one we will use in ID/EX

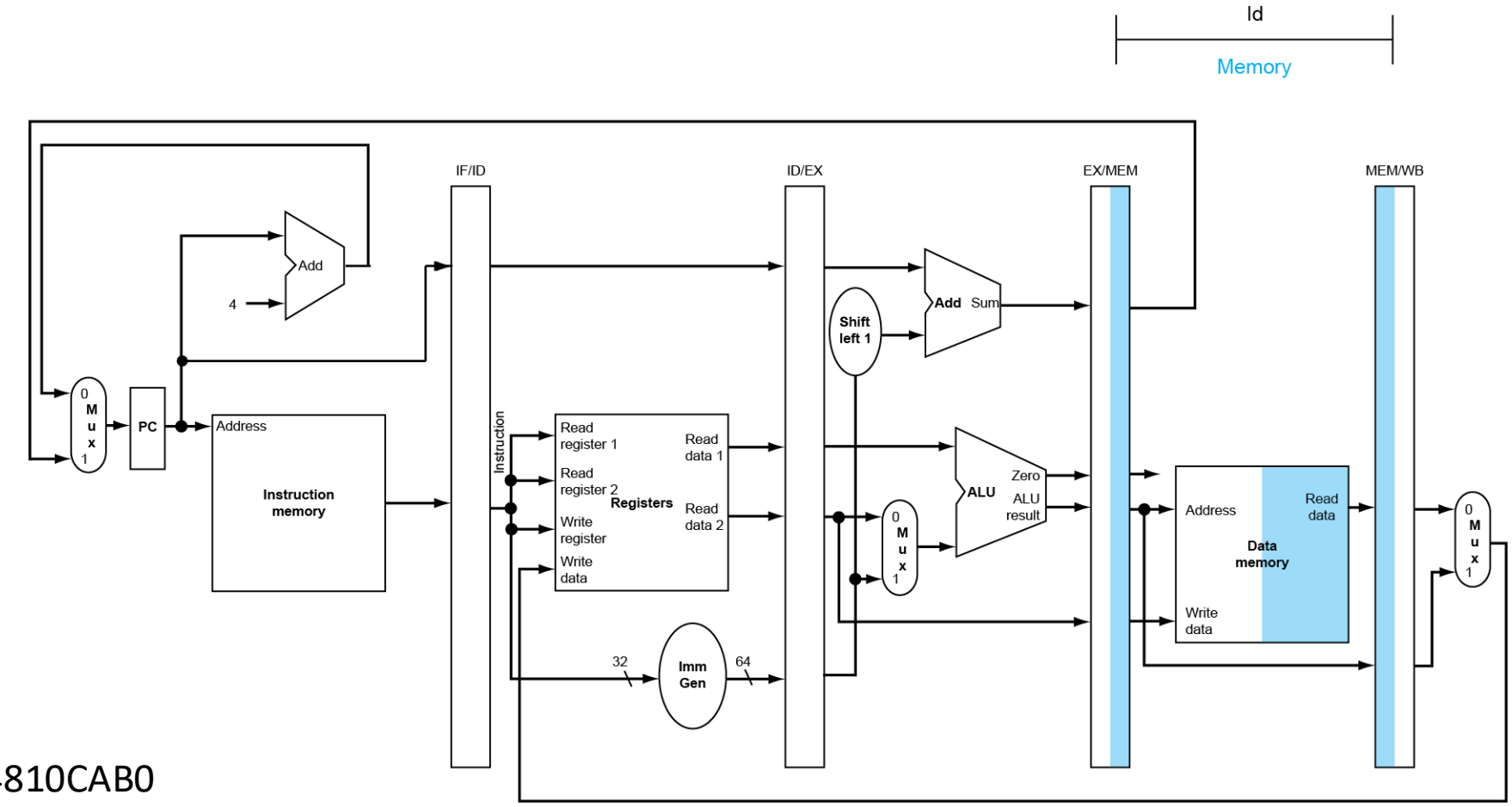
C. Do something else

EX for ld x10, 8(x1)



x10 holds 5
x1 holds 0x4810CAB0
0x4810CAB8 holds 12

MEM for ld x10, 8(x1)



x10 holds 5
x1 holds 0x4810CAB0
0x4810CAB8 holds 12

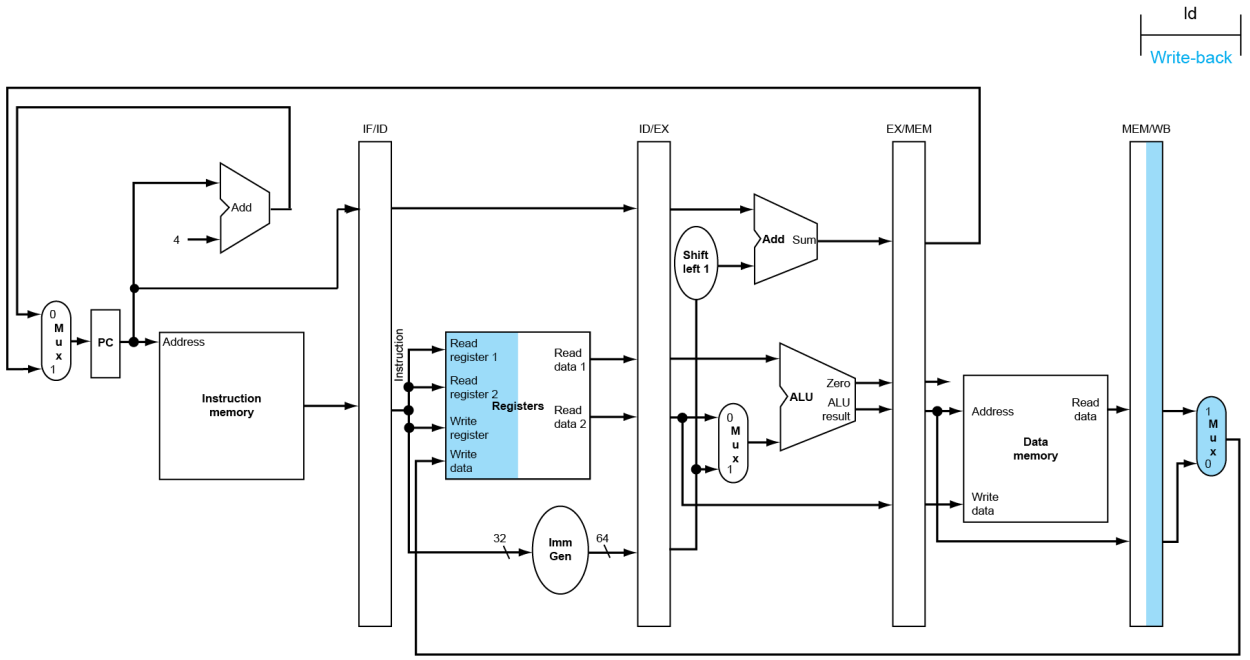
When we do WB for load

A. Everything will be fine

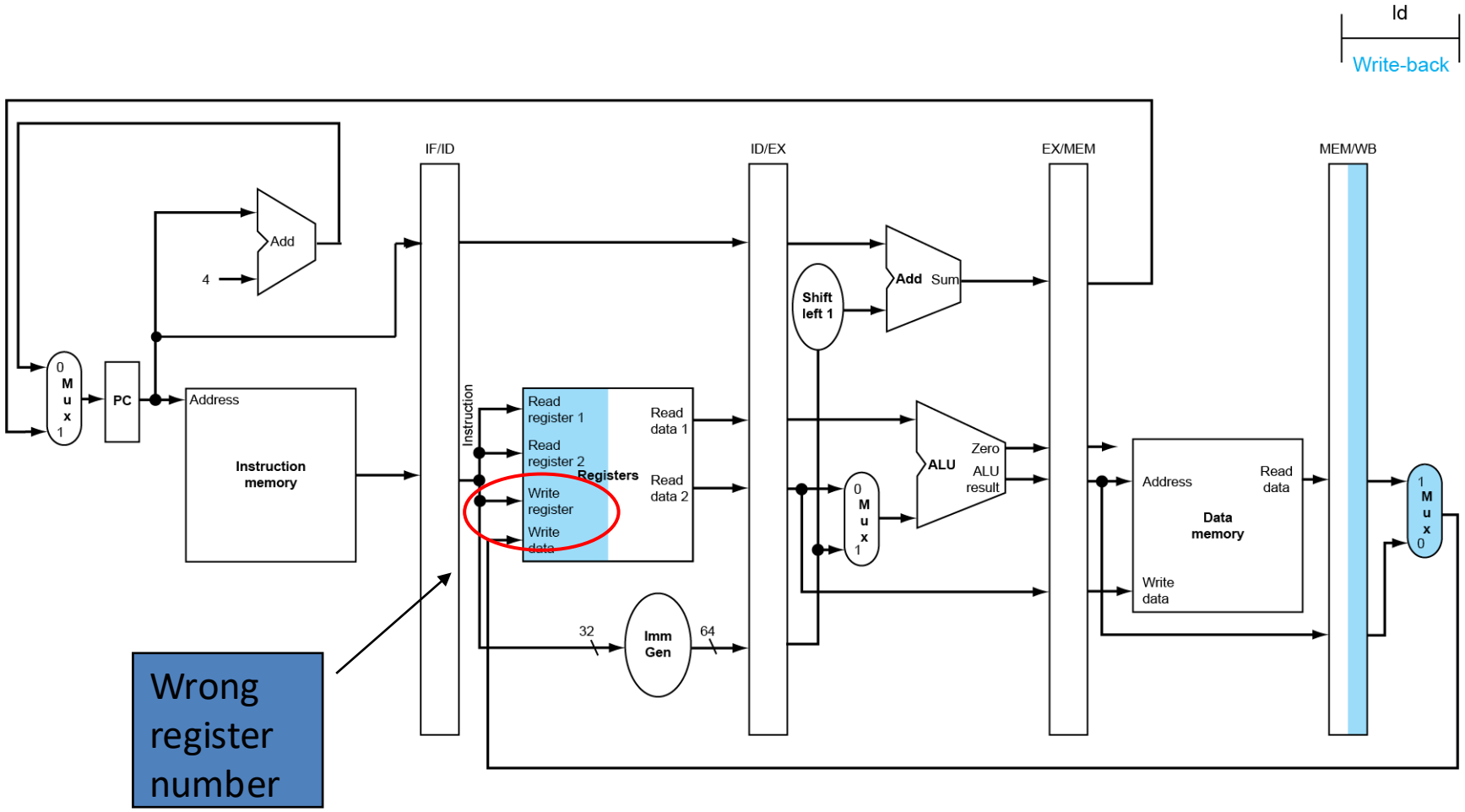
B. The data to write to the register will be wrong

C. The register number to write to will be wrong

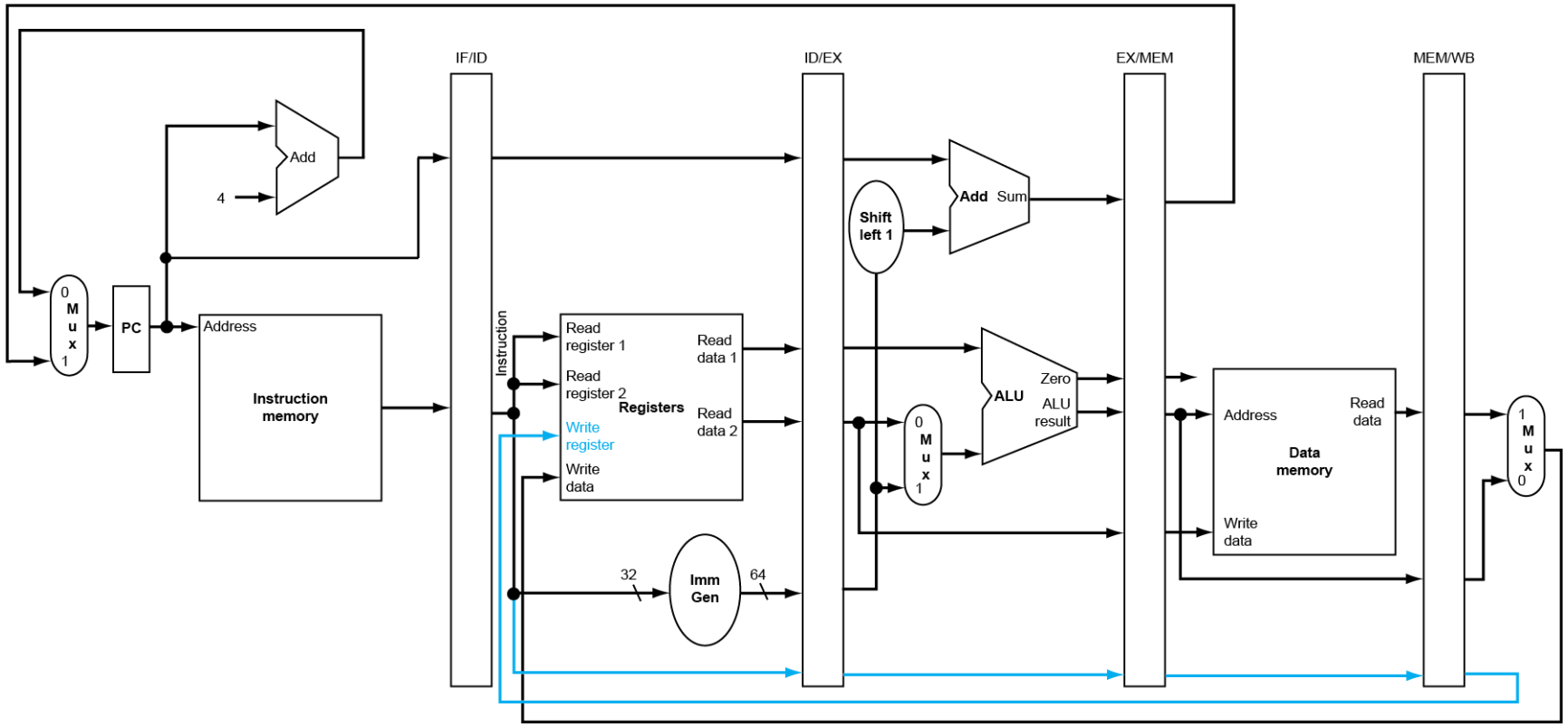
D. Both the data and the register number will be wrong



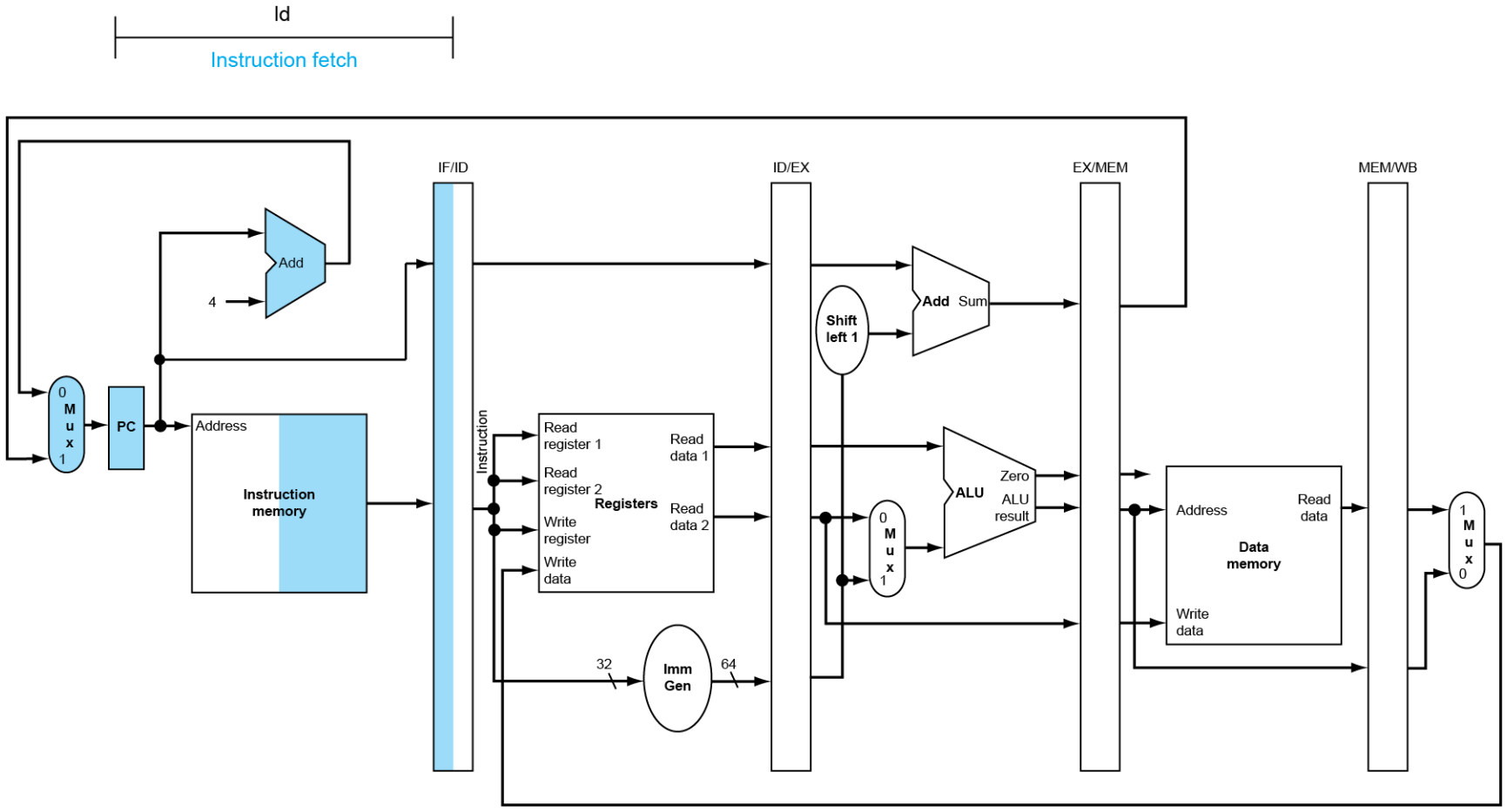
WB for Load


ld
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
32
Imm Gen
64
Wrong register number

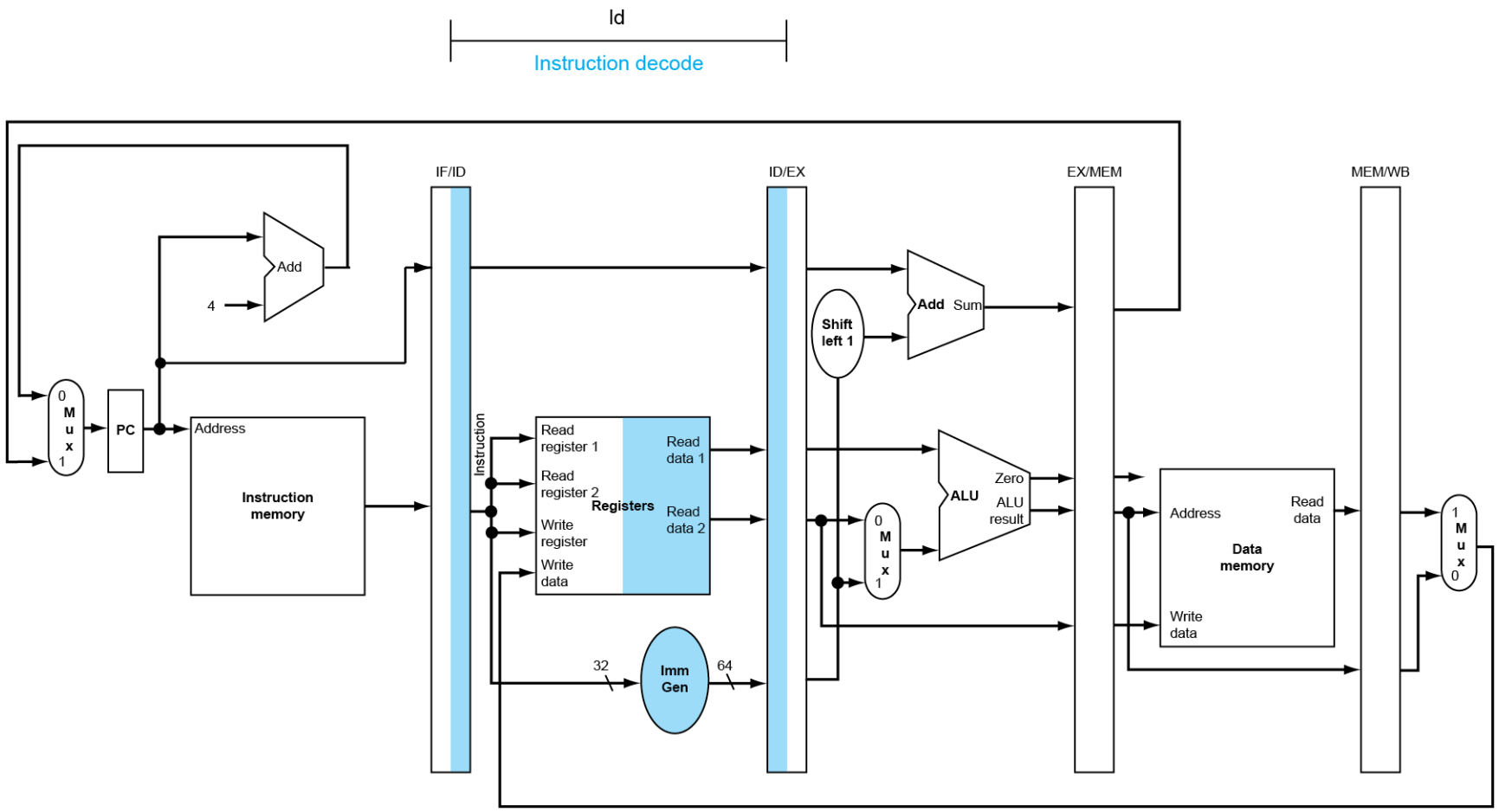
Corrected Datapath for Load



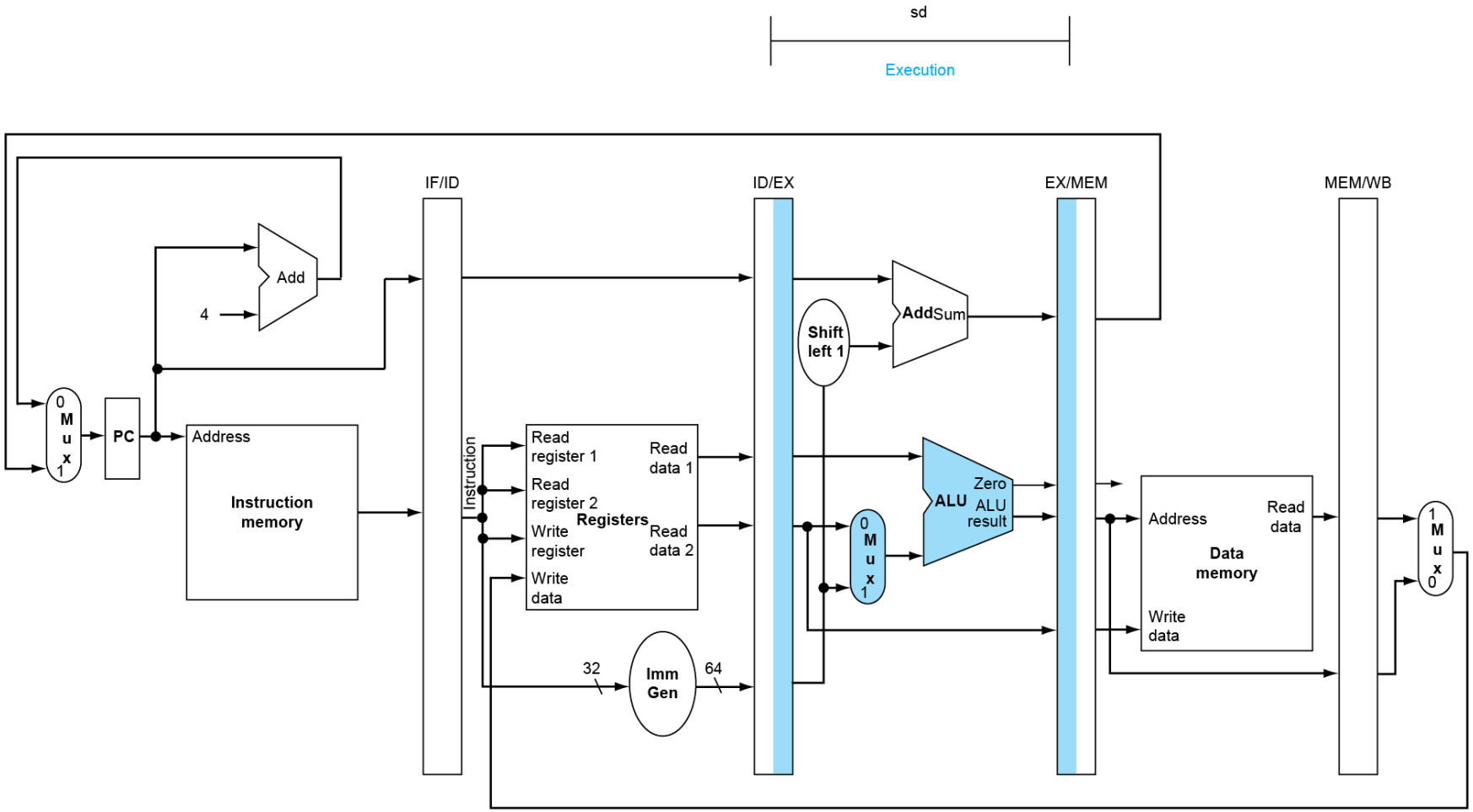
IF for sd


ld
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0

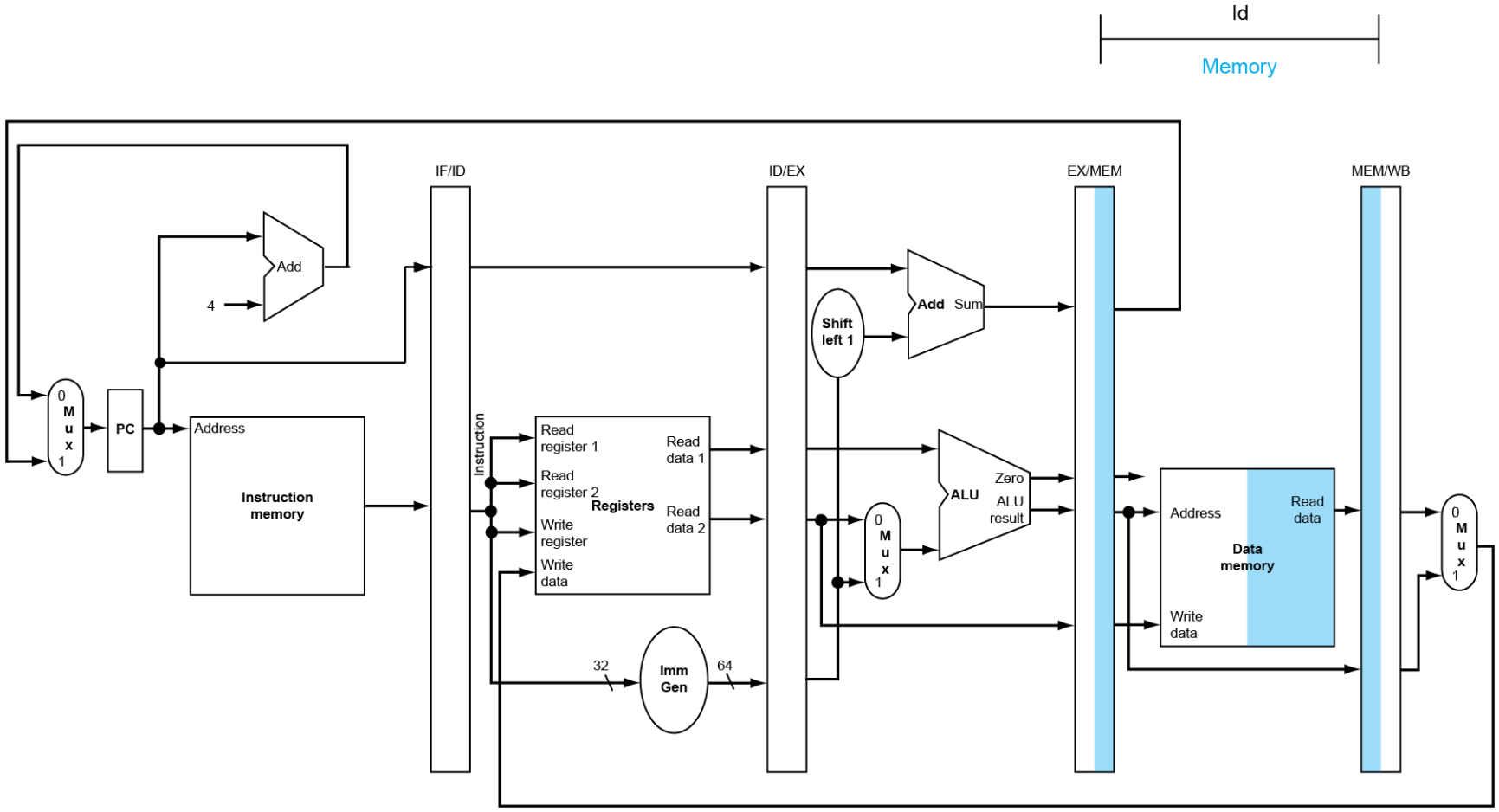
ID for sd


ld
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0

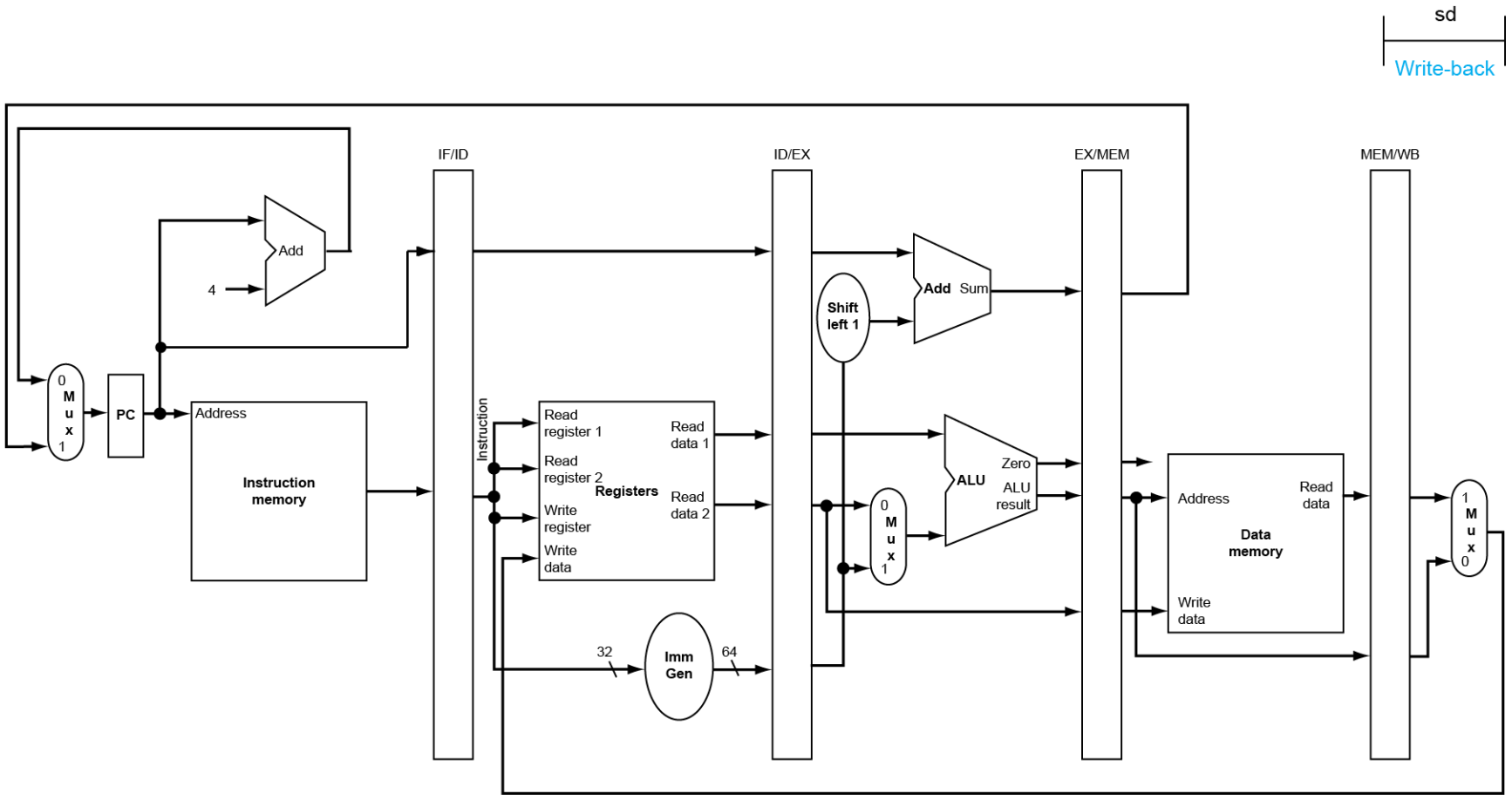
EX for sd


sd
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
AddSum
ALU
Zero
ALU result
32
Imm Gen
64
Address
Data memory
Read data
Write data
1
M
u
x
0

MEM for sd


ld
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
Add
Sum
ALU
Zero
ALU result
32
Imm Gen
64
Address
Data memory
Read data
Write data

WB for sd

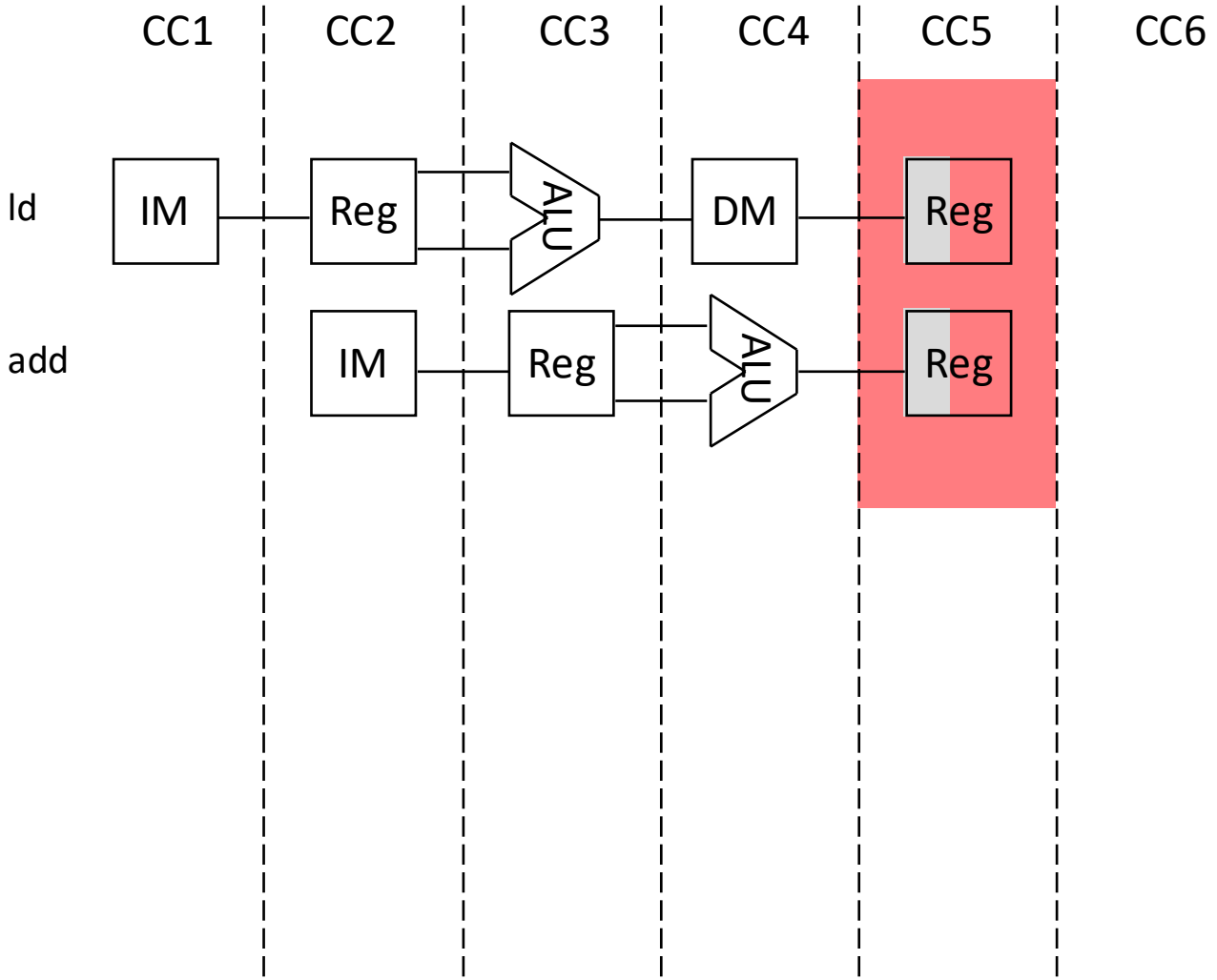

sd
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Imm Gen
64
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
1
M
u
x
0

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?

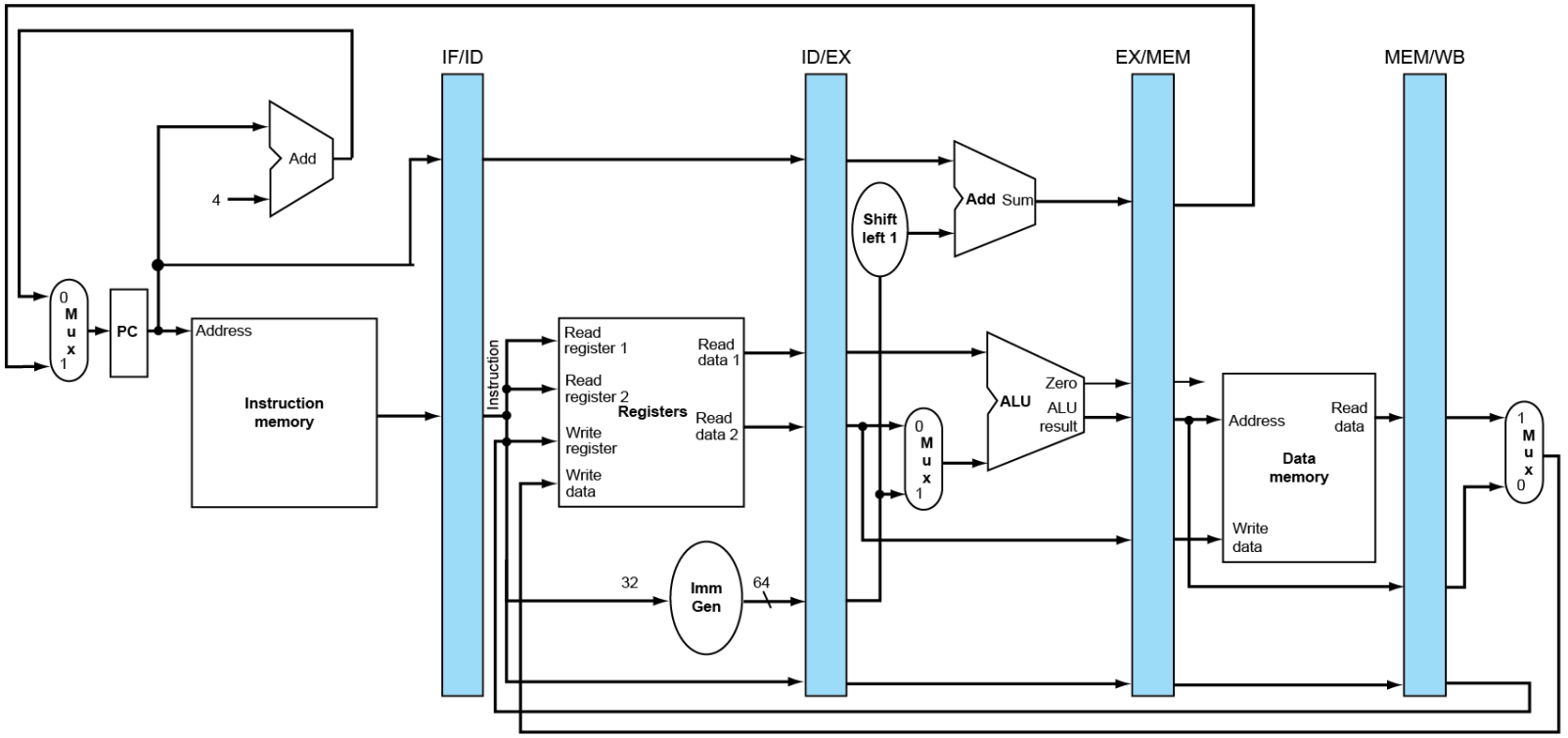
Selection	Yes/No	Reason (Choose BEST answer)
A	Yes	Decreasing R-type to 4 cycles improves instruction throughput
B	Yes	Decreasing R-type to 4 cycles improves instruction latency
C	No	Decreasing R-type to 4 cycles causes hazards
D	No	Decreasing R-type to 4 cycles causes hazards and doesn't impact throughput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

Mixed Instructions in the Pipeline

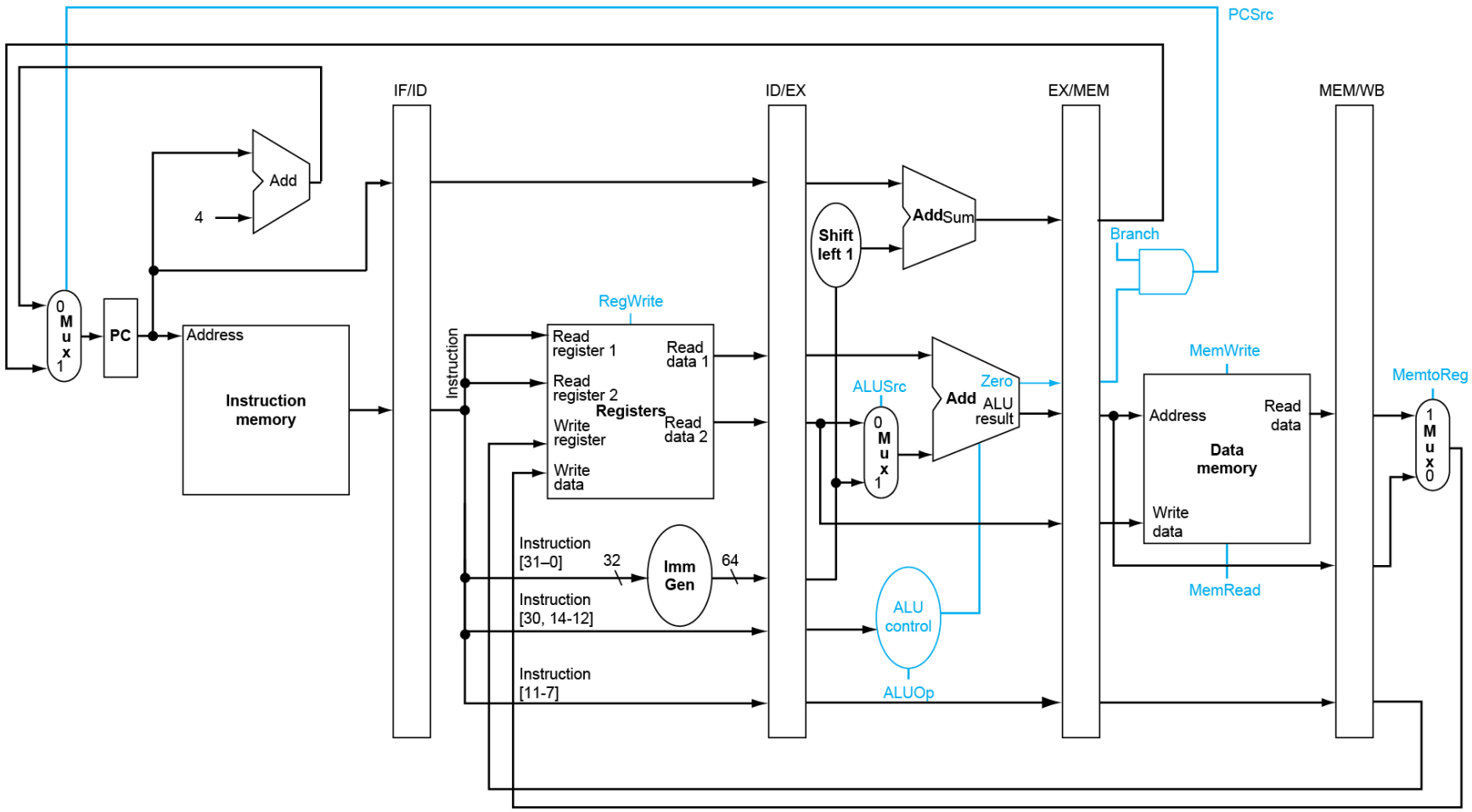


State of pipeline in a given cycle

add x14, x5, x6	ld x13, 48(x1)	add x12, x3, x4	sub x11, x2, x3	ld x10, 40(x1)
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.

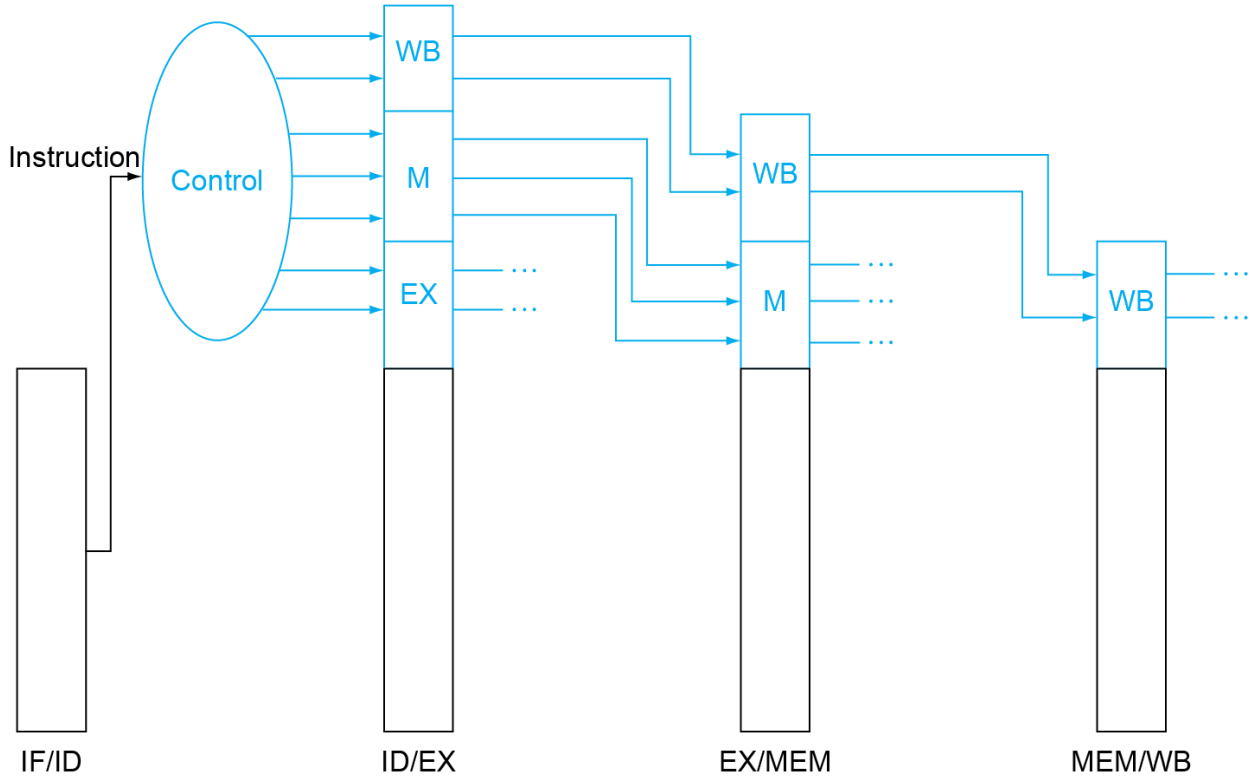
C. We don’t need to do anything special; it will work the way it is.

Pipeline Control

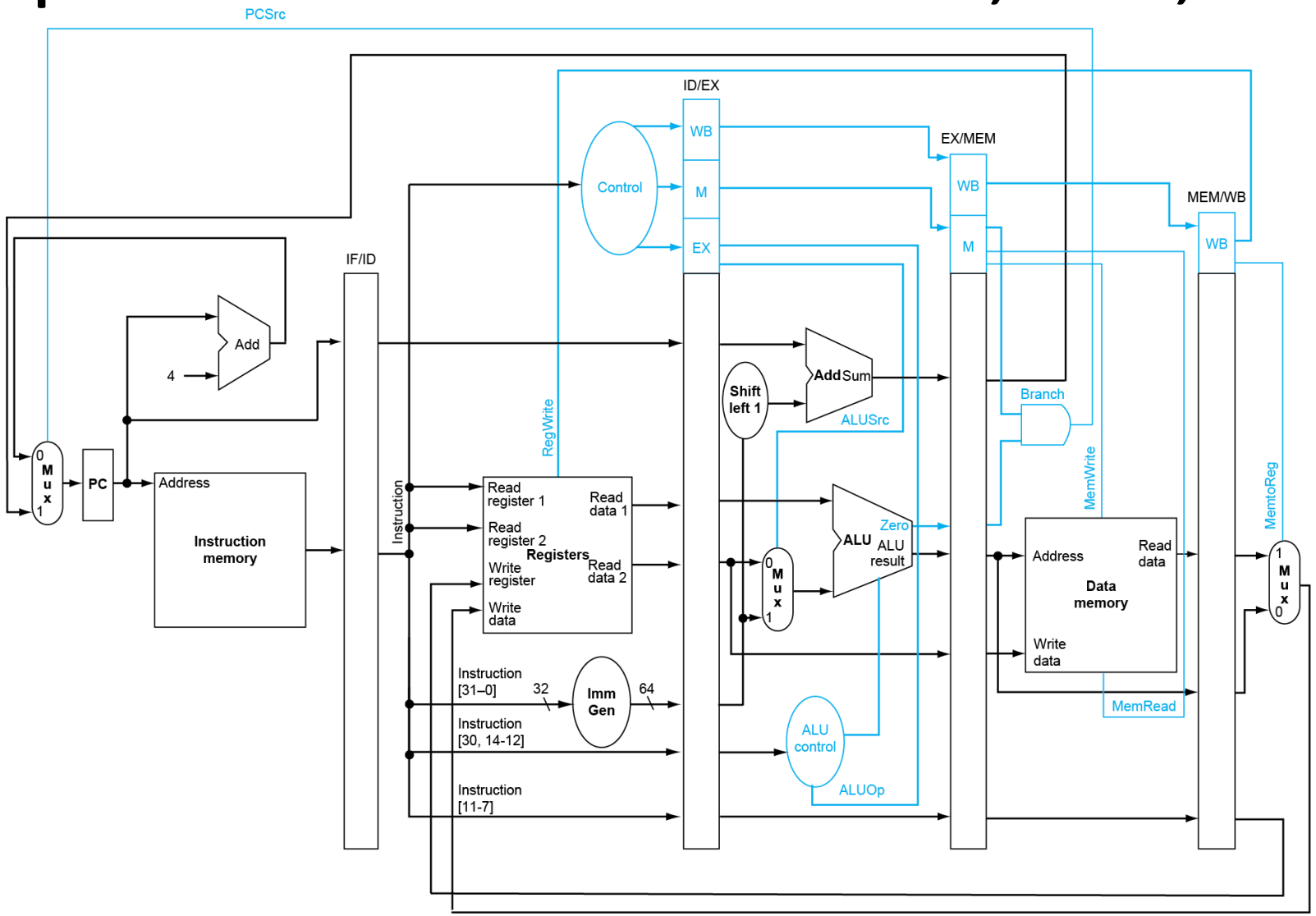
- IF Stage: read instruction memory (always) and write PC to the pipeline register
- ID Stage: compute all control signals for subsequent stages
- EX, MEM, and WB stages have control signals
  - The pipeline registers will need to store the control signals

Pipelined Control

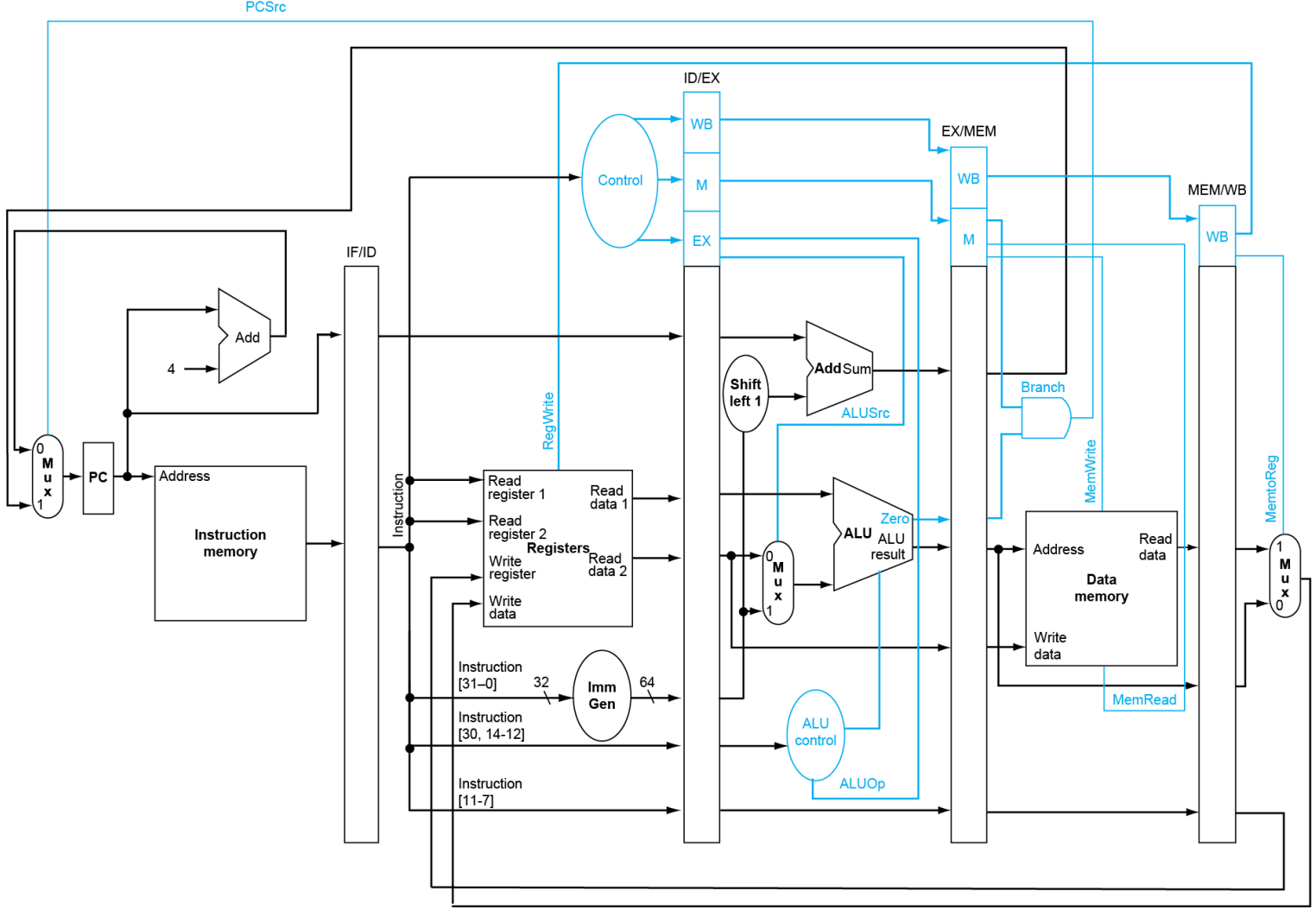
Control signals derived from instruction



Pipelined Control: add x10, x11, x12



x11 holds 5
x12 holds 6



	EX Stage			MEM Stage			WB Stage	
	ALUOp1	ALUOp0	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
R-type	1	0	0	0	0	0	1	0
`ld`	0	0	1	0	1	0	1	1
`sd`	0	0	1	0	0	1	0	X
`beq`	0	1	0	1	0	0	0	X

Questions on Pipeline Control?

Reading

- Next lecture: Dealing with data hazards
- Reading: § 4.7