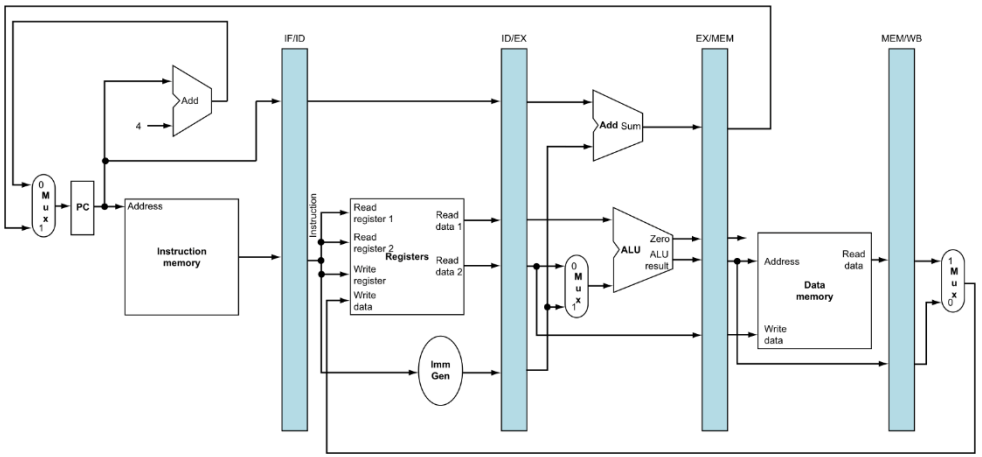
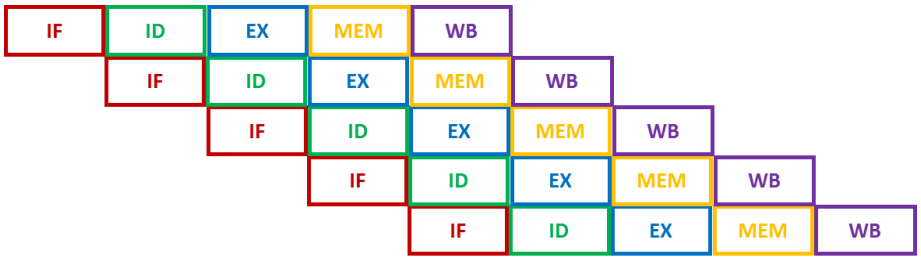
Processor Architecture

Pipelined RISC-V Implementation (Part I)



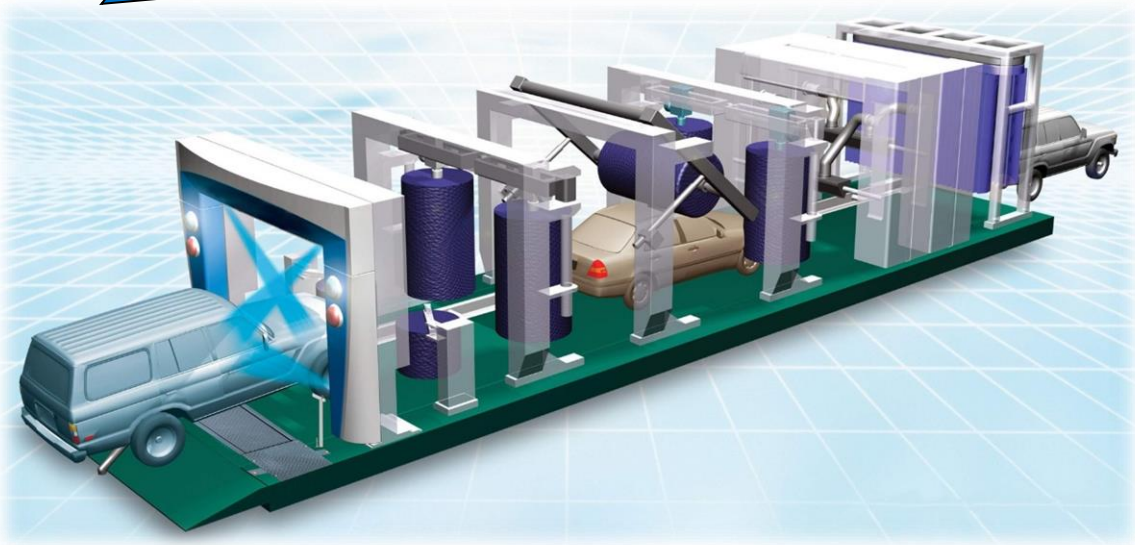
Module Outline

- From a Single-Cycle to a Pipelined Implementation
- A Pipelined Datapath
- A Pipelined Control Path
- Module Summary



From a Single-Cycle to a Pipelined Implementation

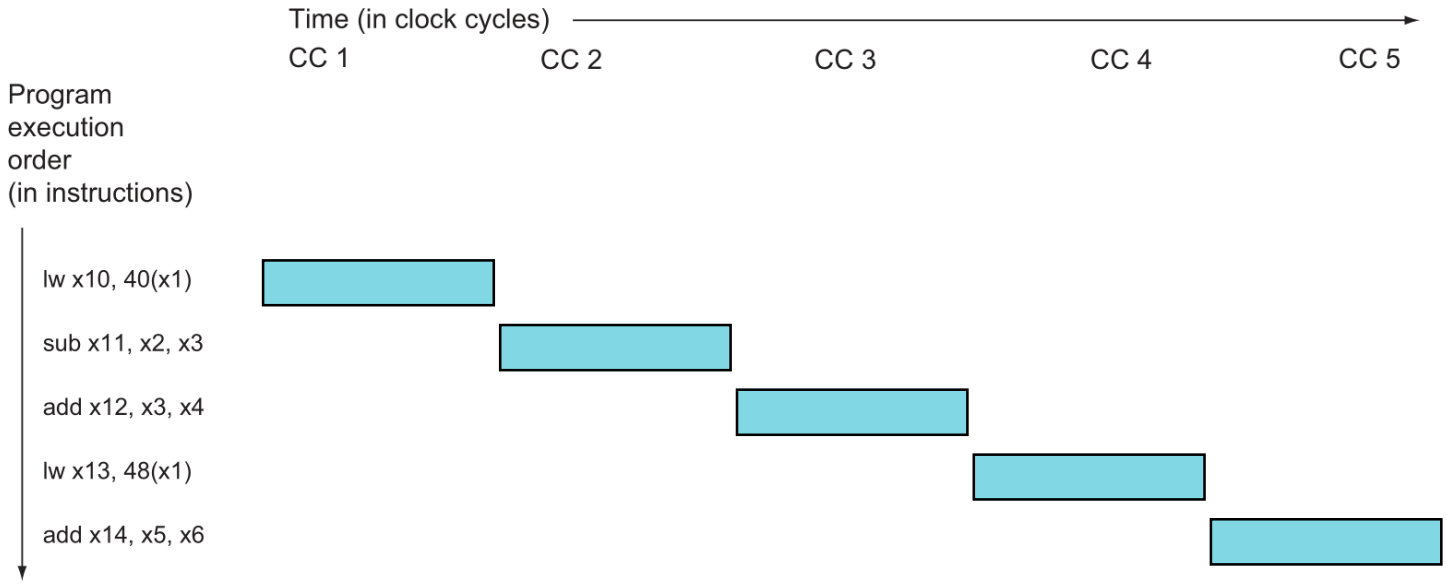
From Single-Cycle to Pipelining



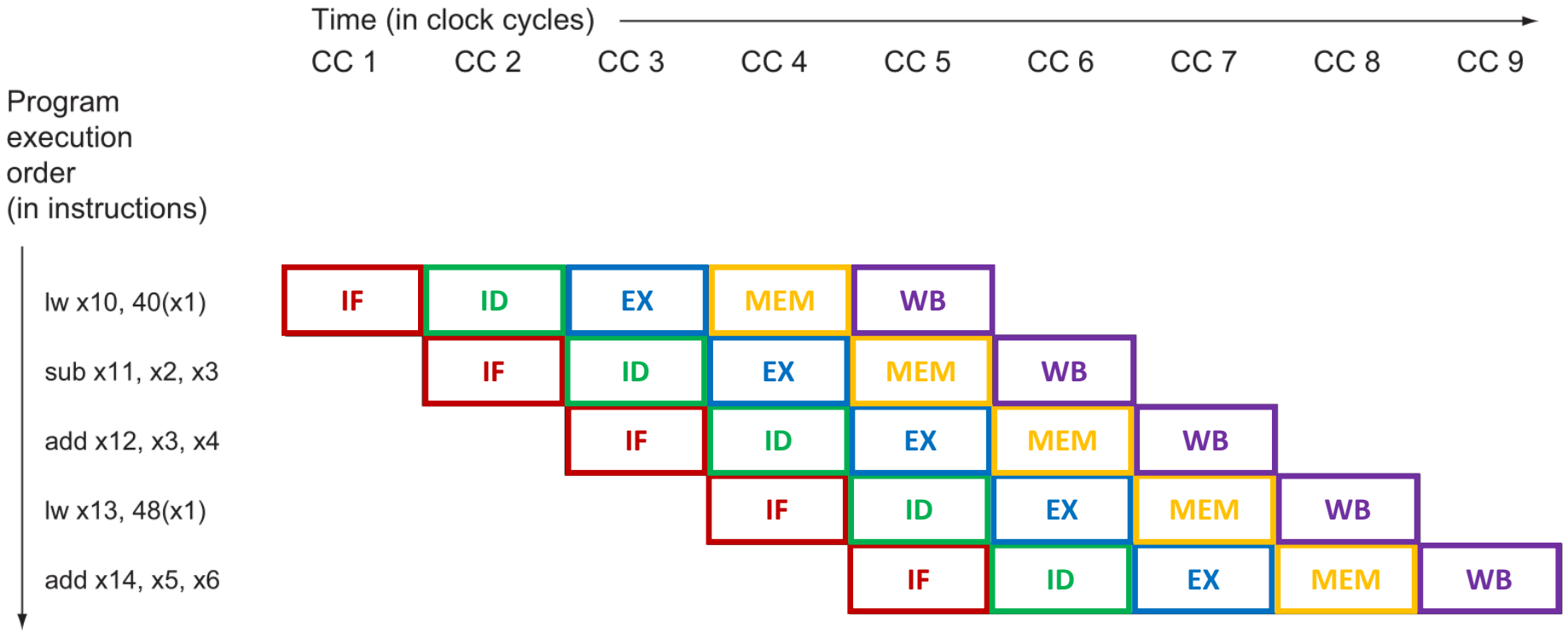
From Single-Cycle Execution

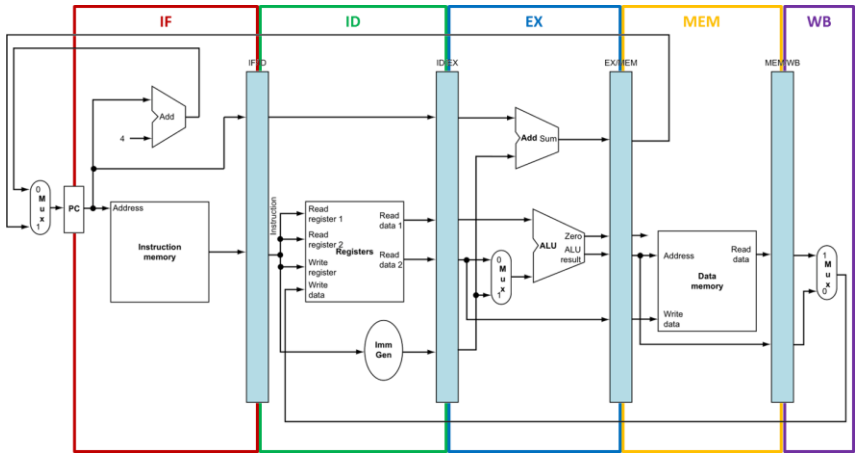
- Instruction sequence

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



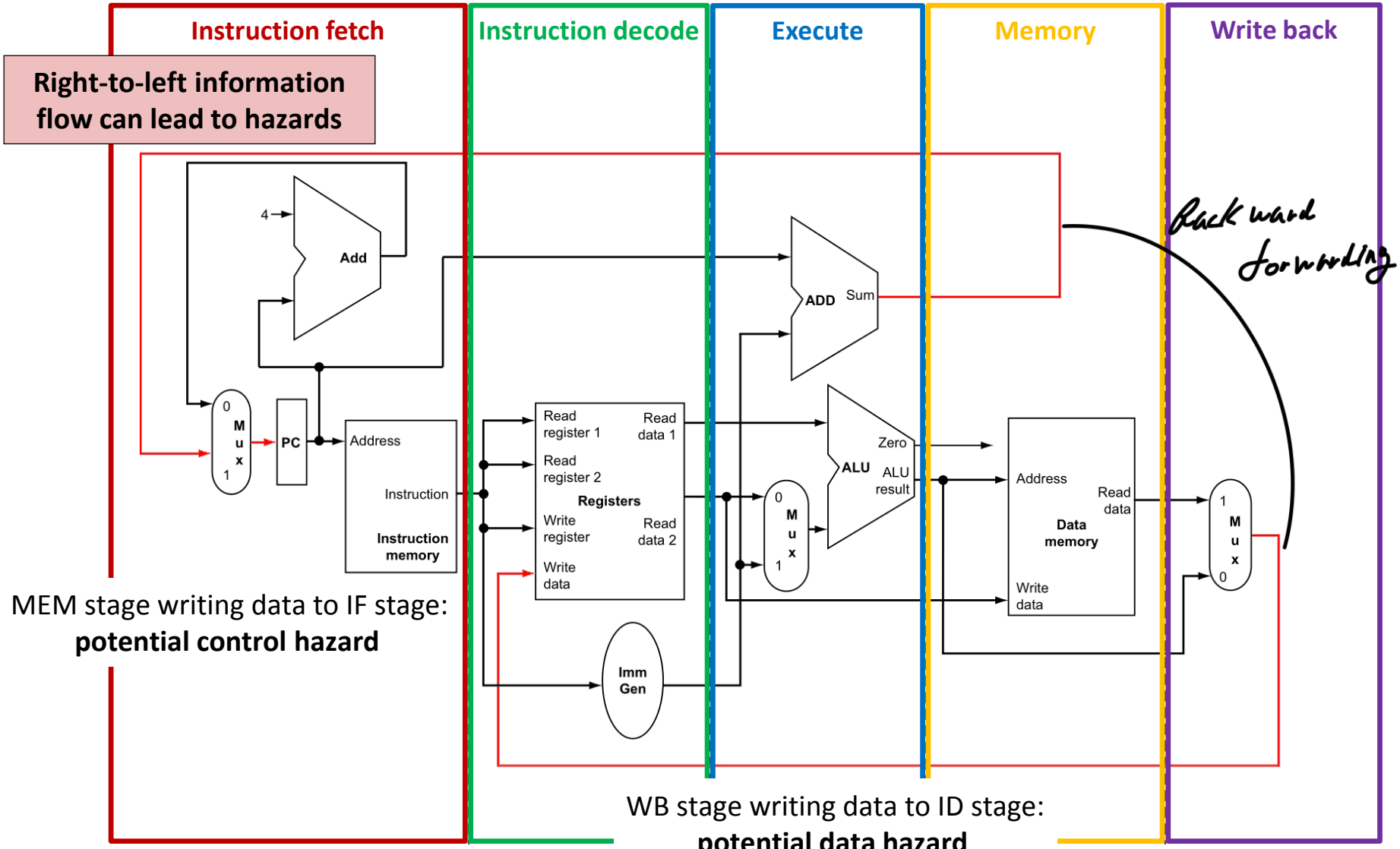
Pipelined Execution





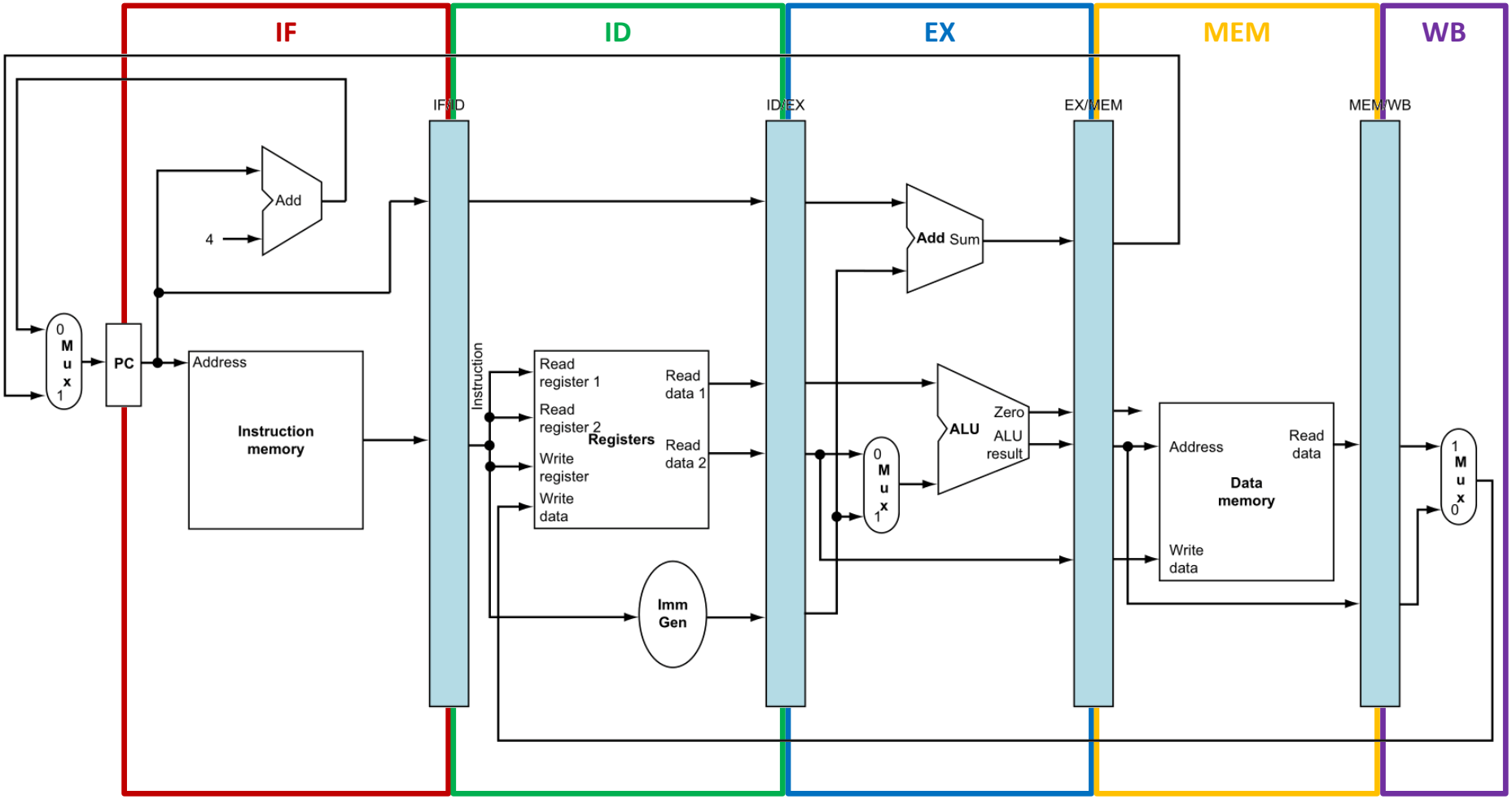
A Pipelined Datapath

RISC-V Pipelined Datapath



Pipeline Registers

- Pipeline registers hold information produced in previous cycle



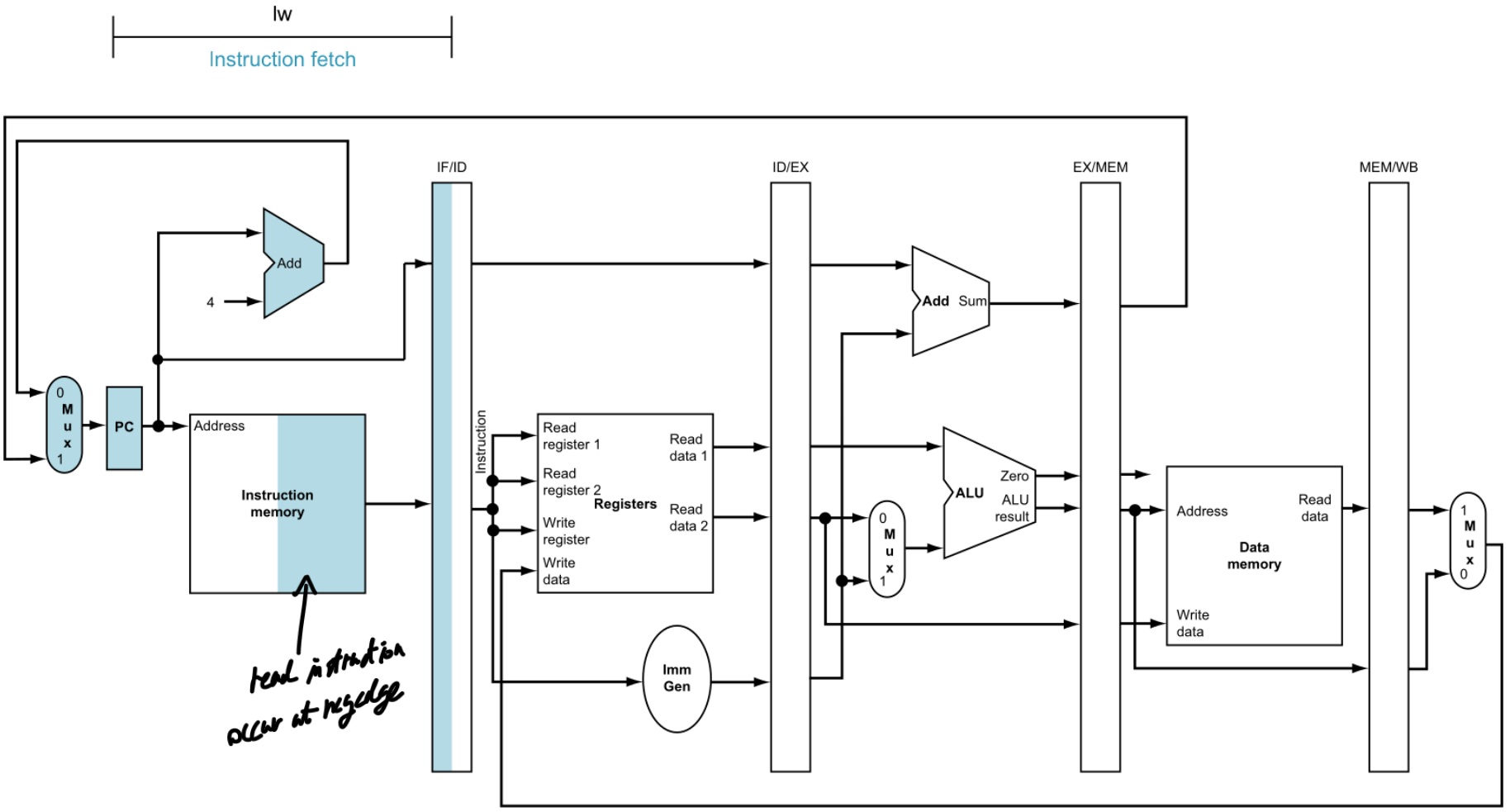
Pipeline Operation

- Cycle-by-cycle flow of instructions through the pipelined datapath

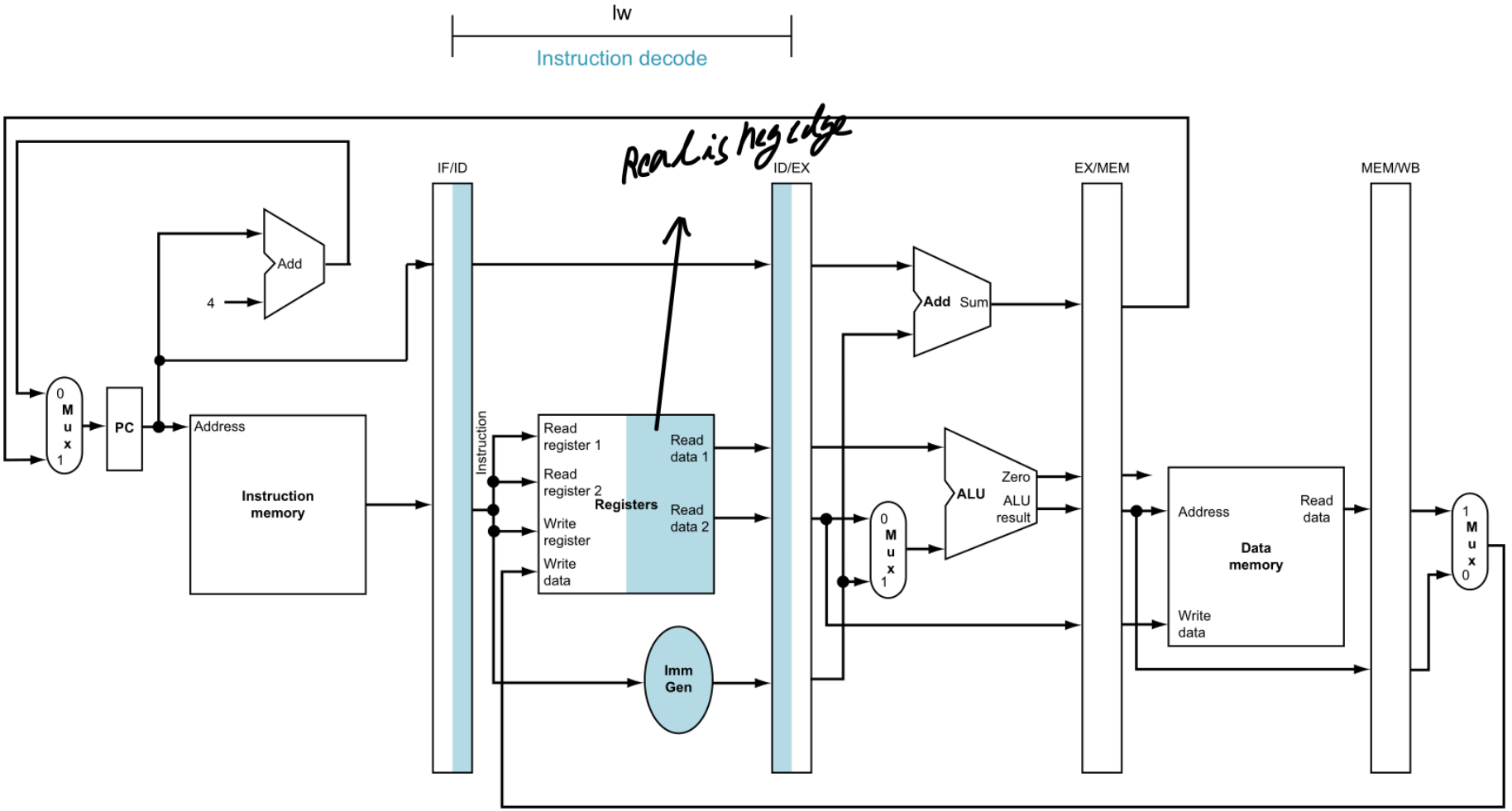
- "Single-clock-cycle" pipeline diagram
  - Shows pipeline usage in a single cycle
  - Highlight resource used
  - (cf.) "multi-clock-cycle" diagram: graph of operation over time

- Let's have a look at "single-clock-cycle" diagrams for load & store

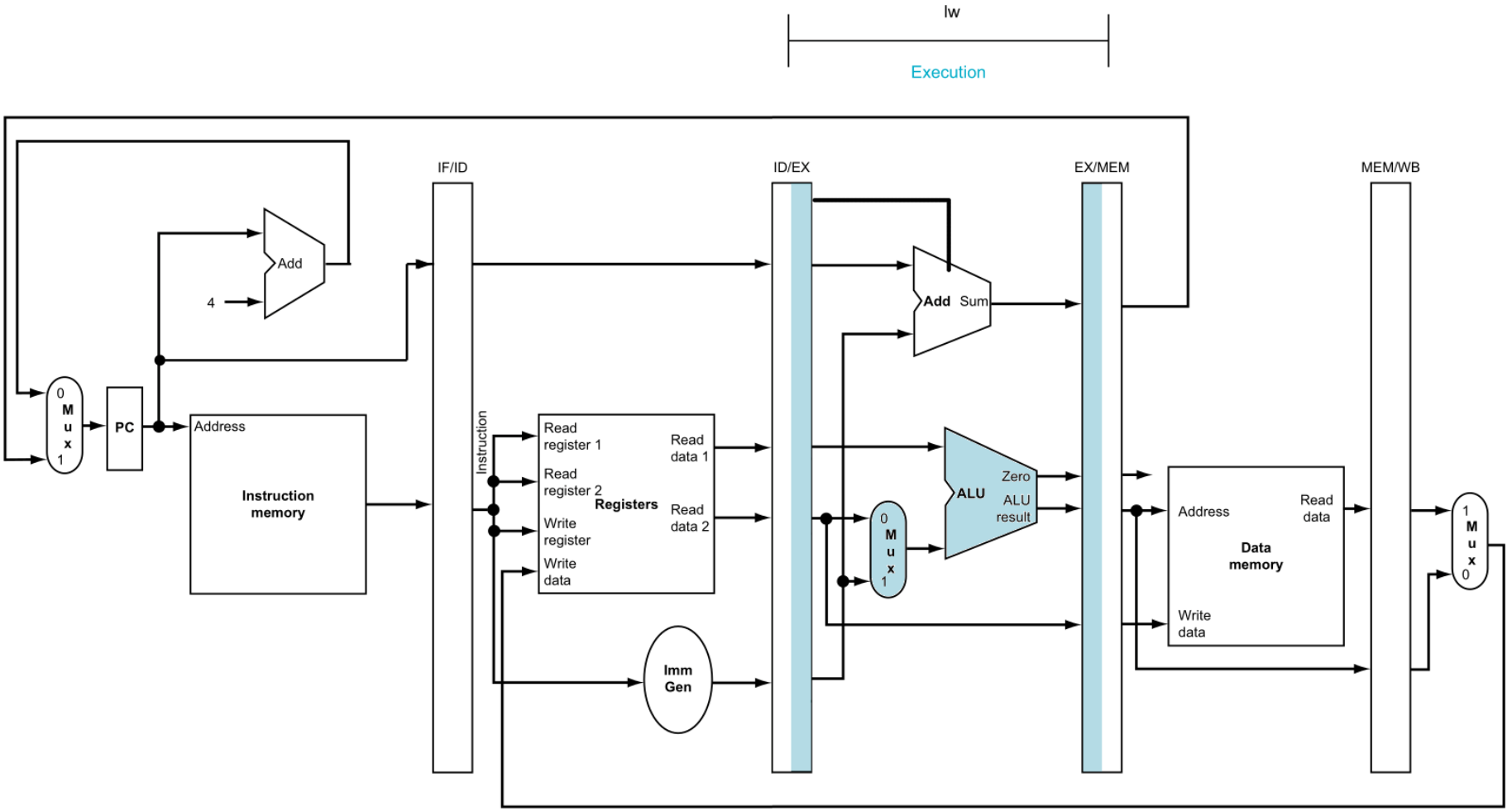
IF for Load



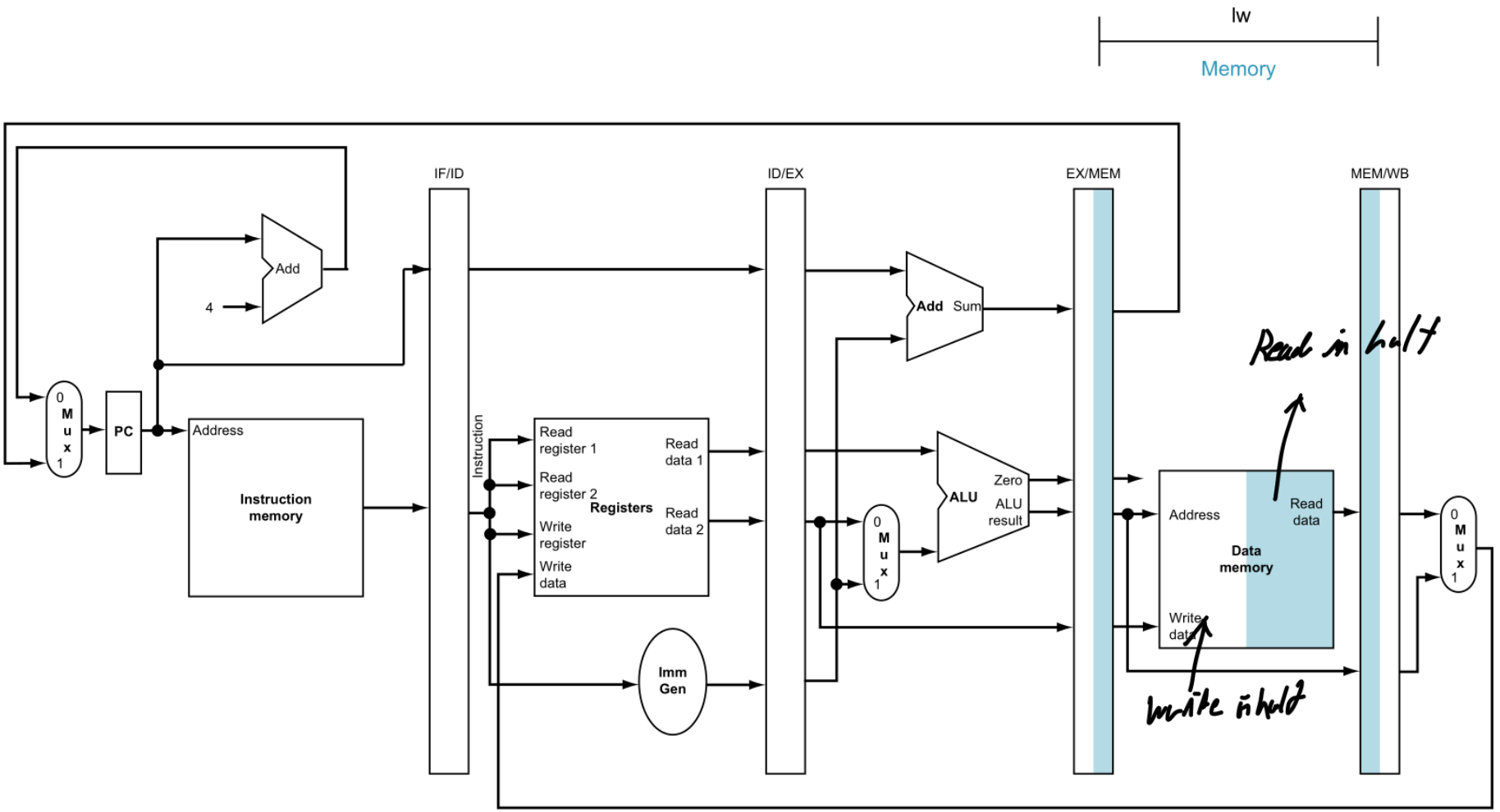
ID for Load



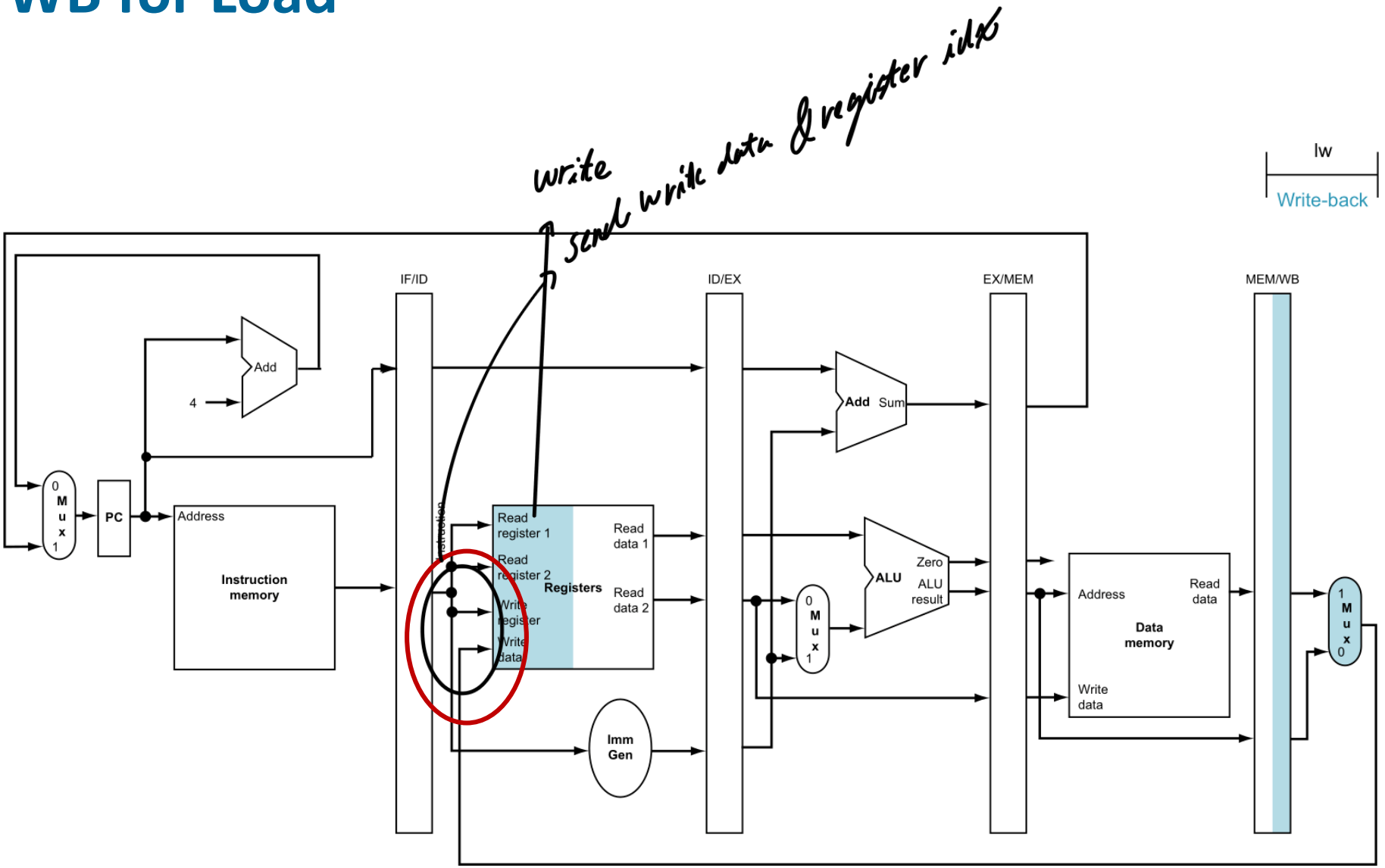
EX for Load



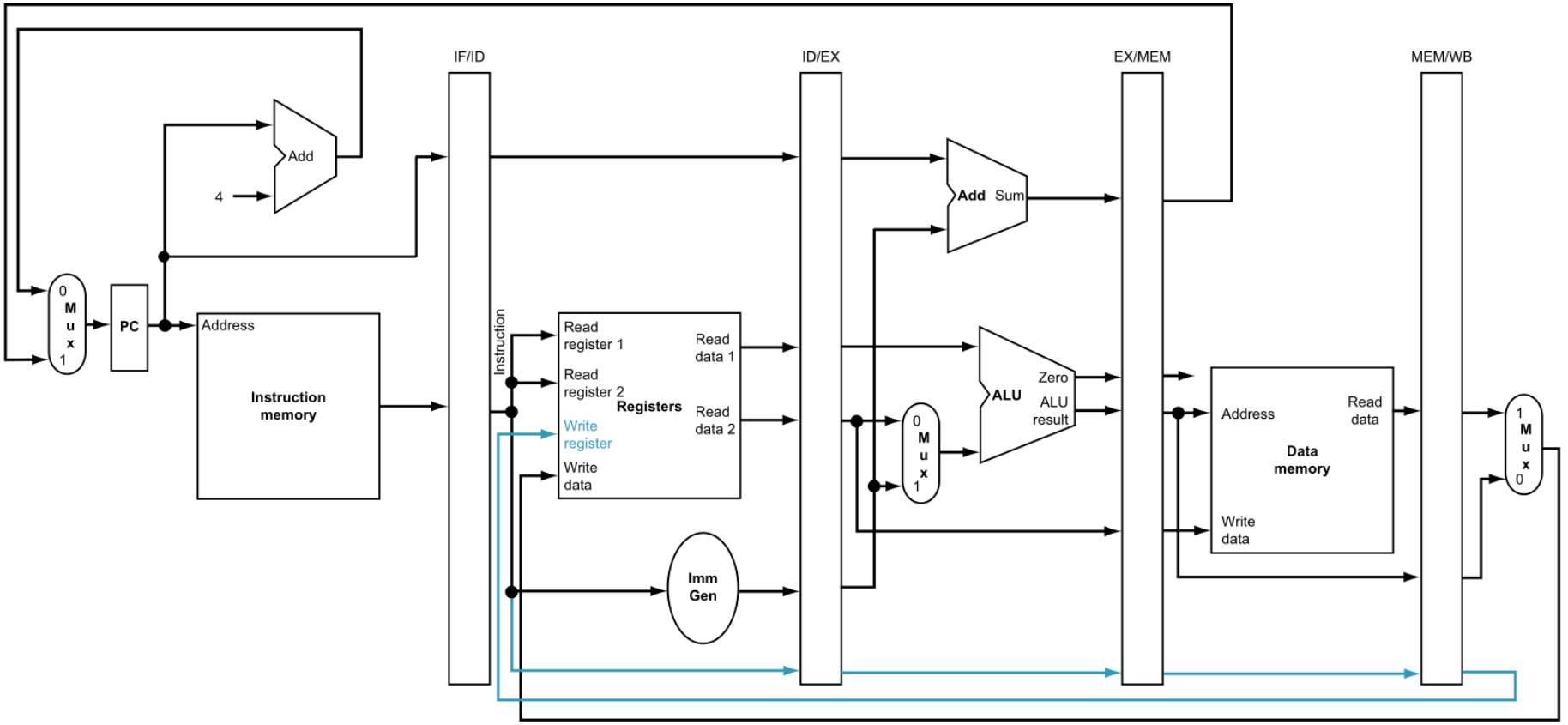
MEM for Load



WB for Load

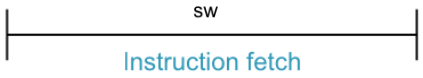


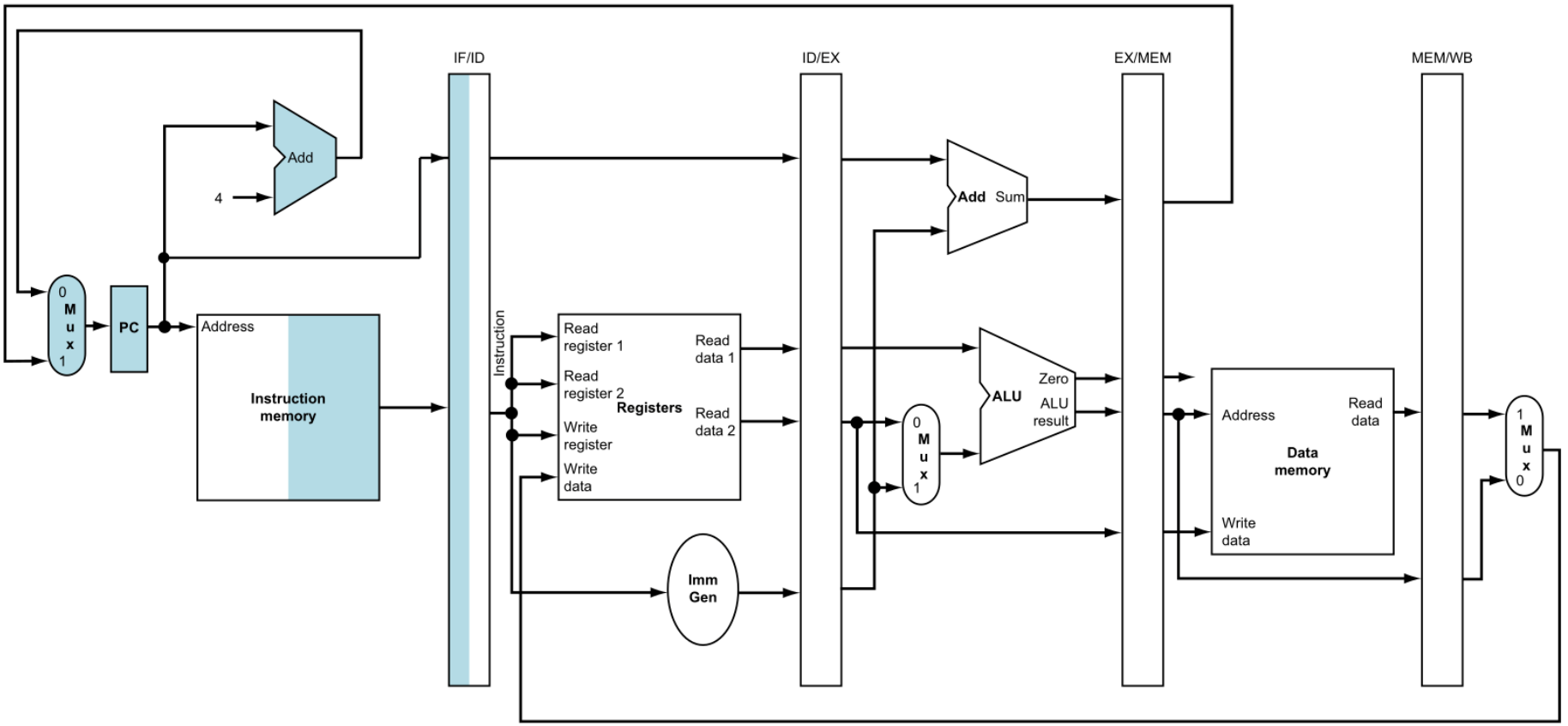
Corrected Datapath for Load



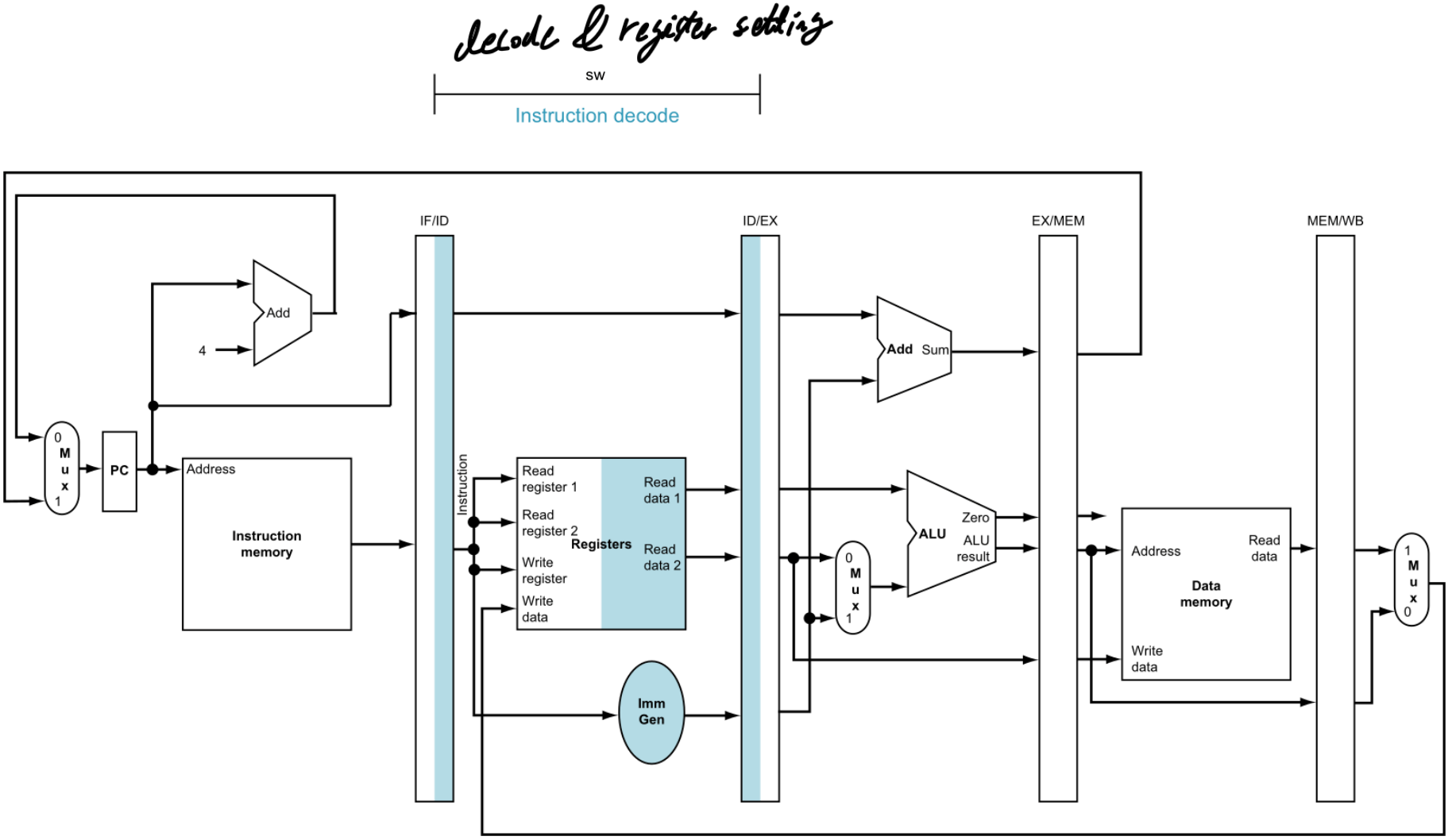
IF for Store

read instruction

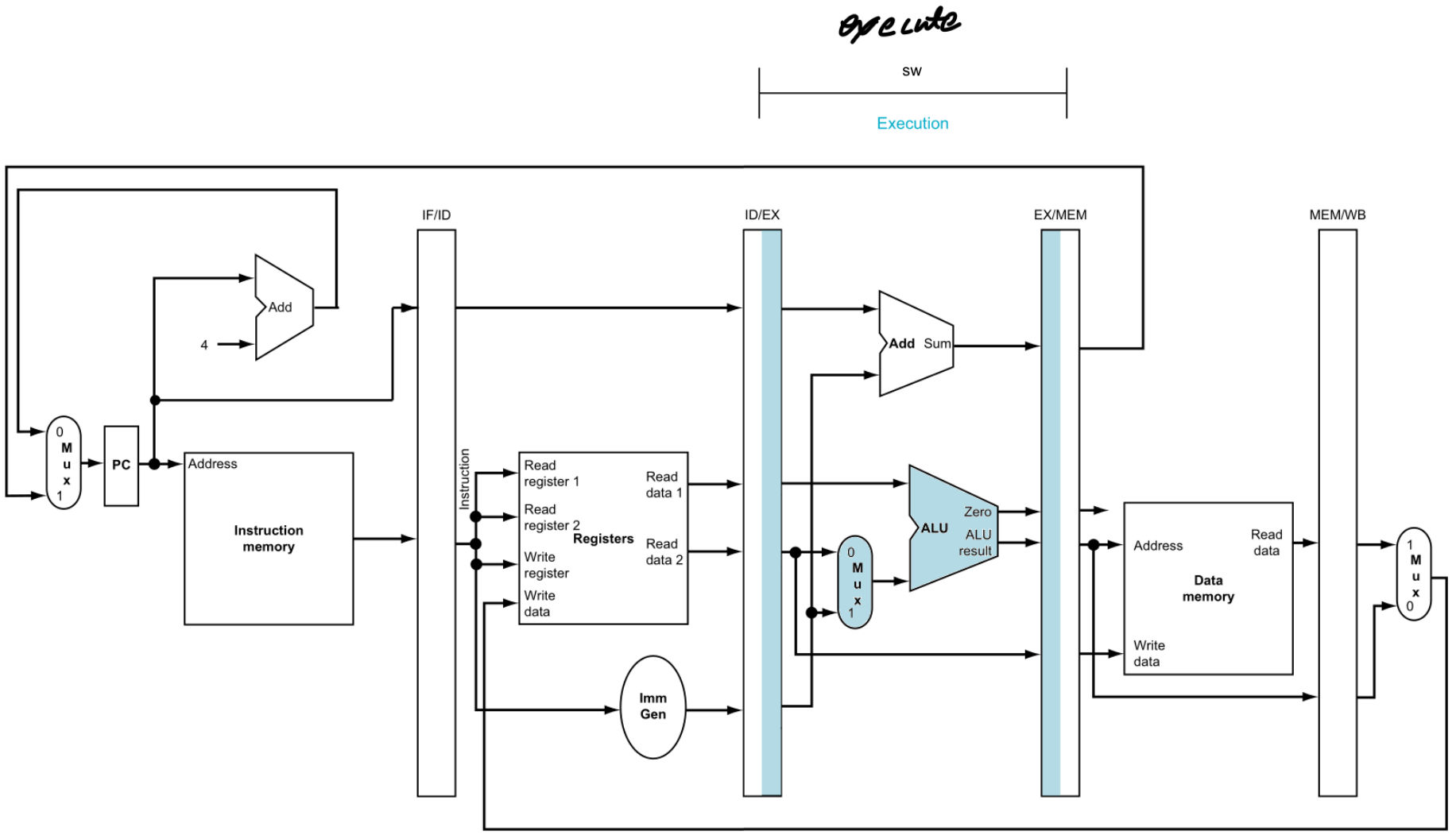




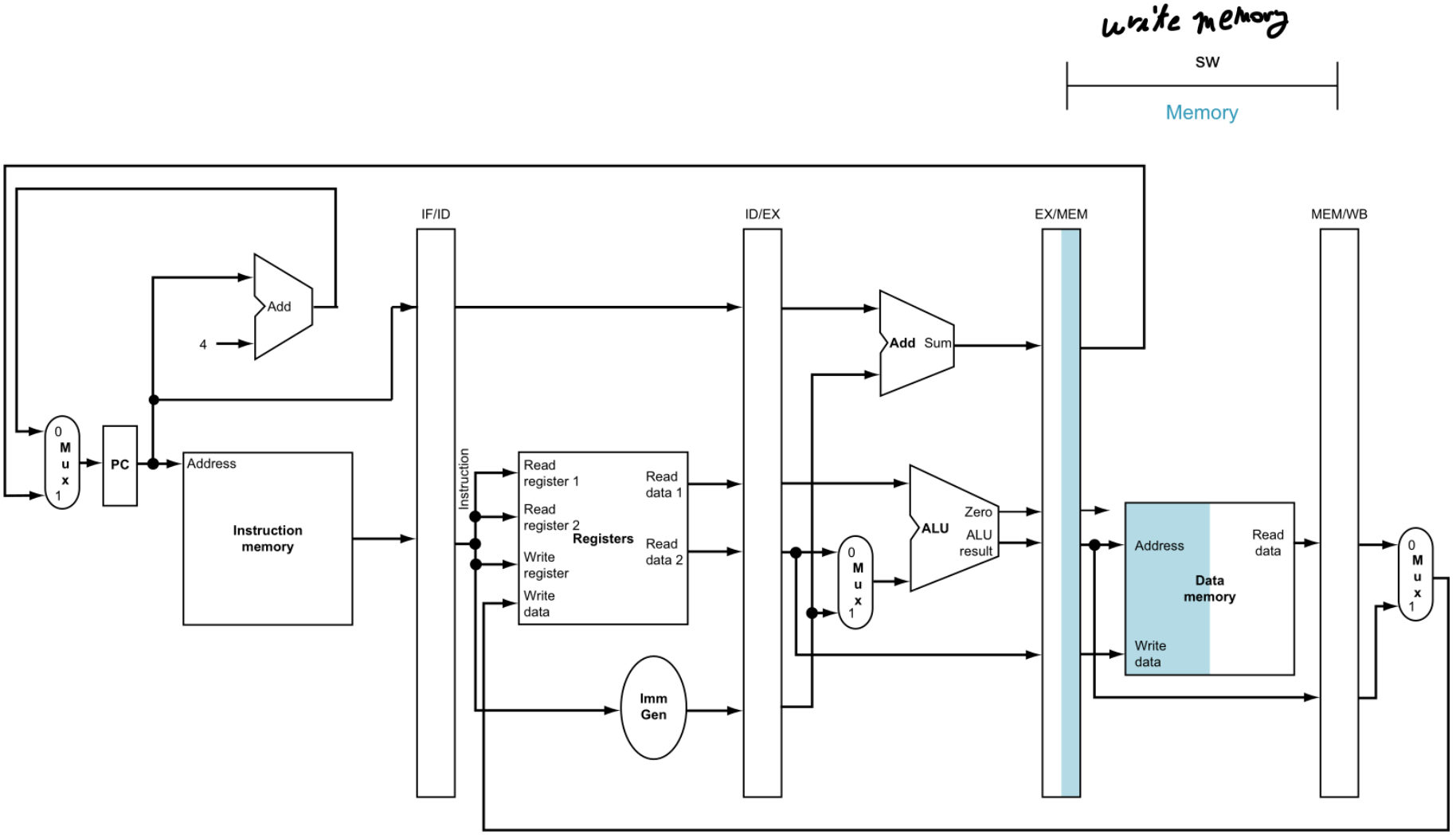
ID for Store



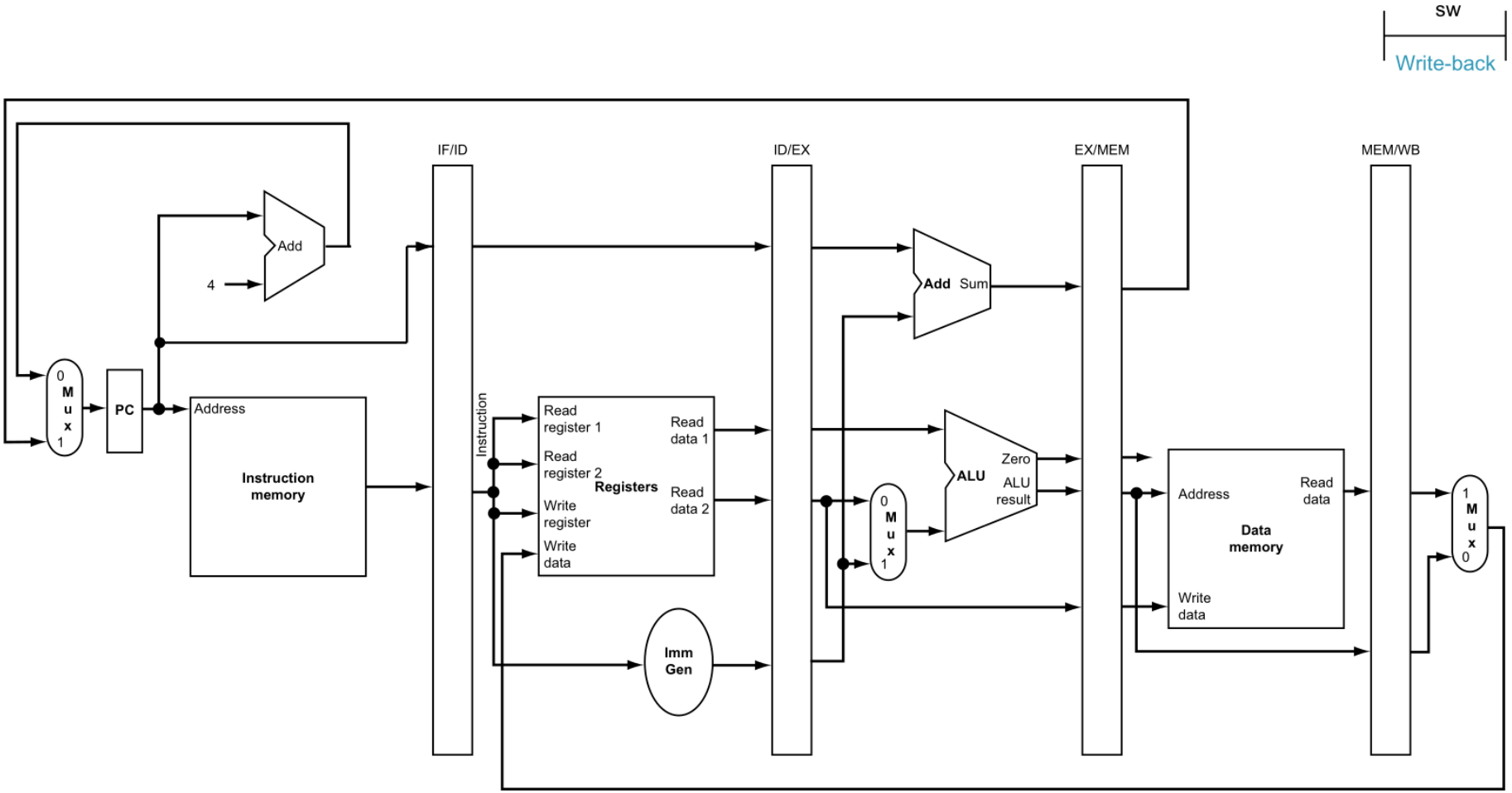
EX for Store



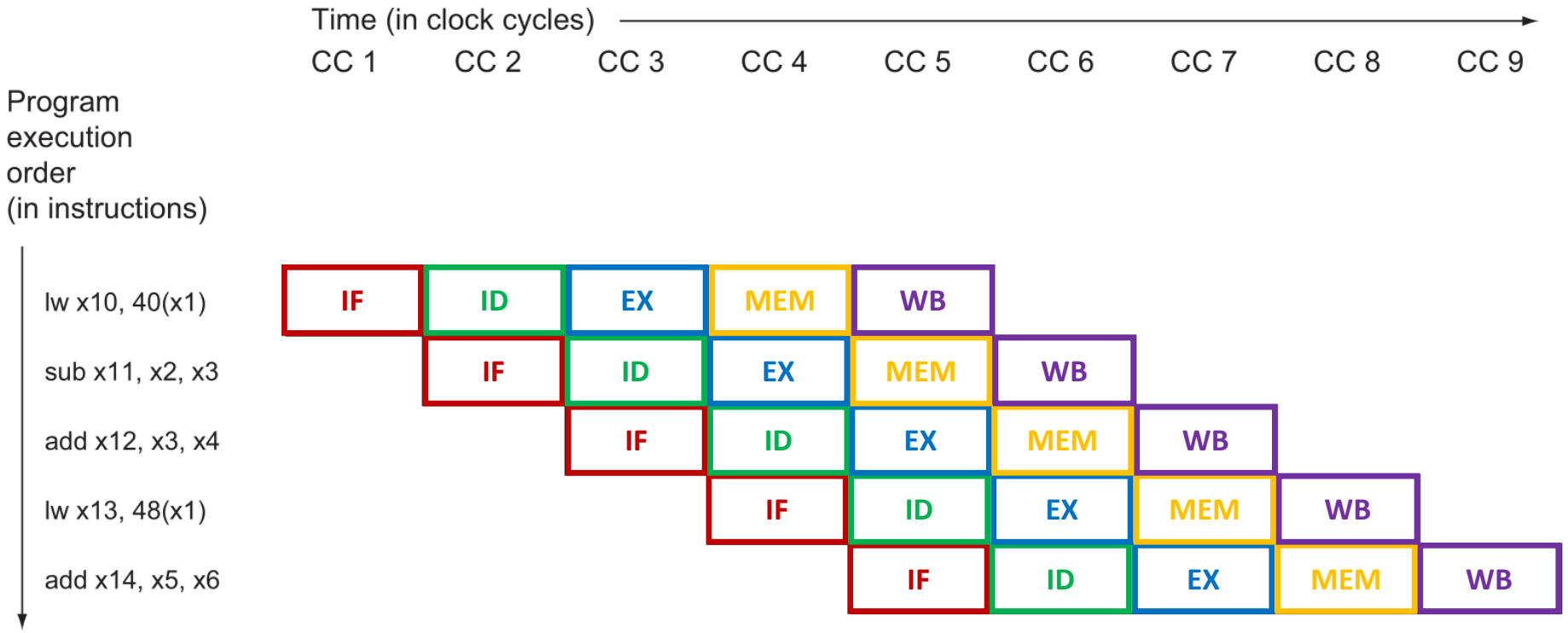
MEM for Store



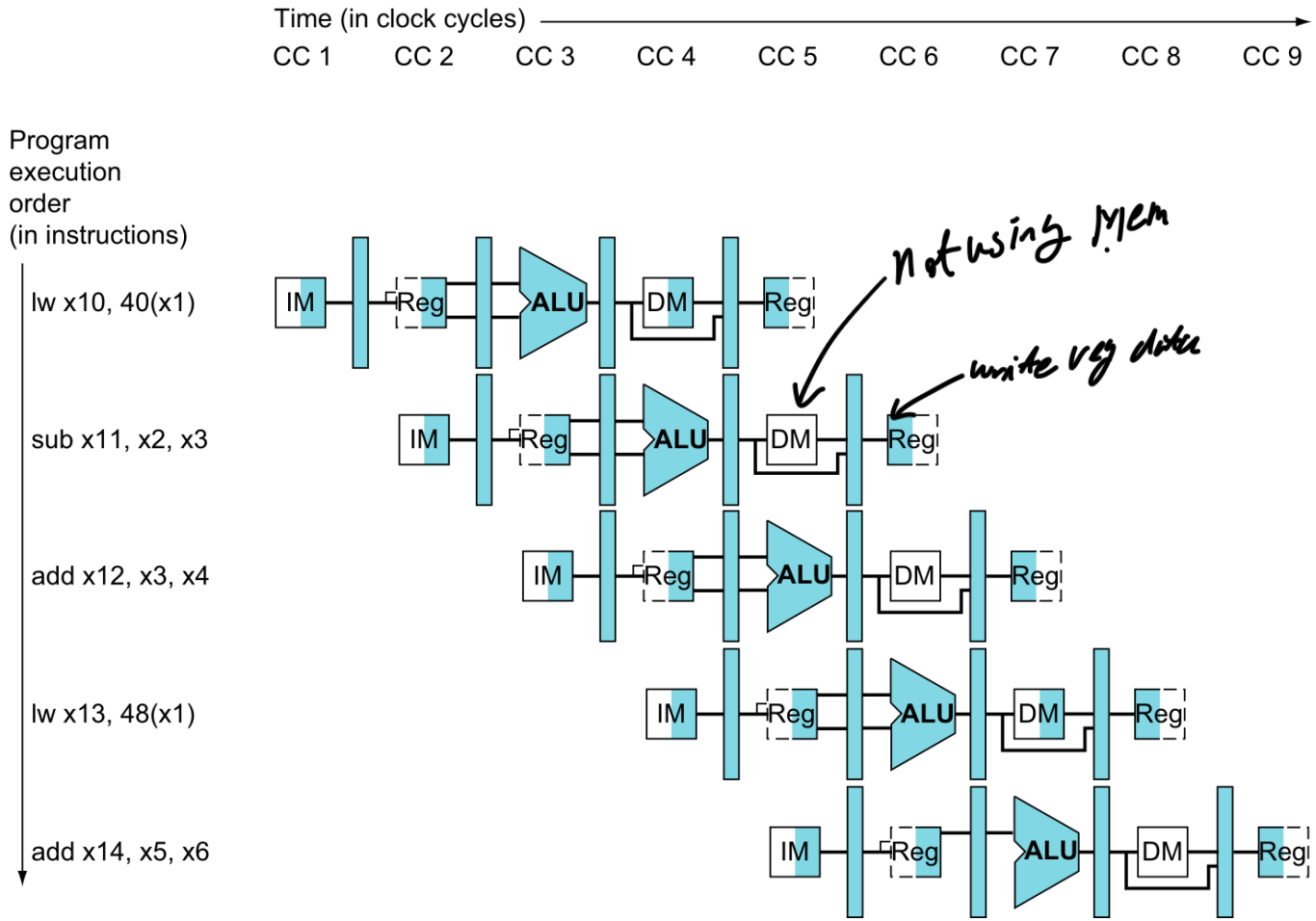
WB for Store



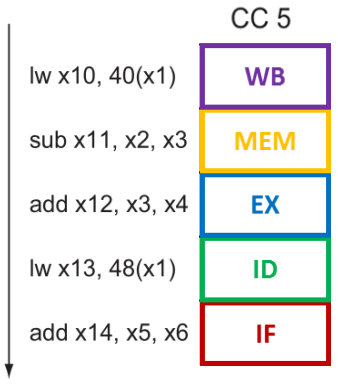
Pipelined Execution



Pipelined Execution

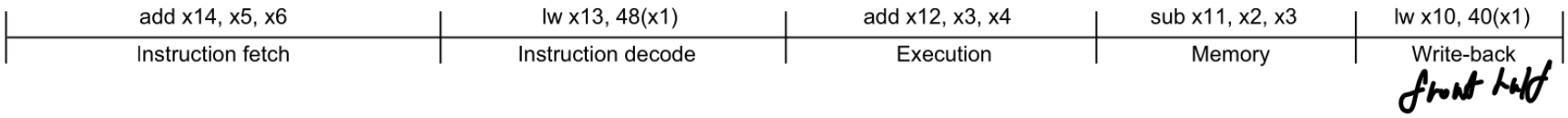


Single-Cycle Pipeline Diagram

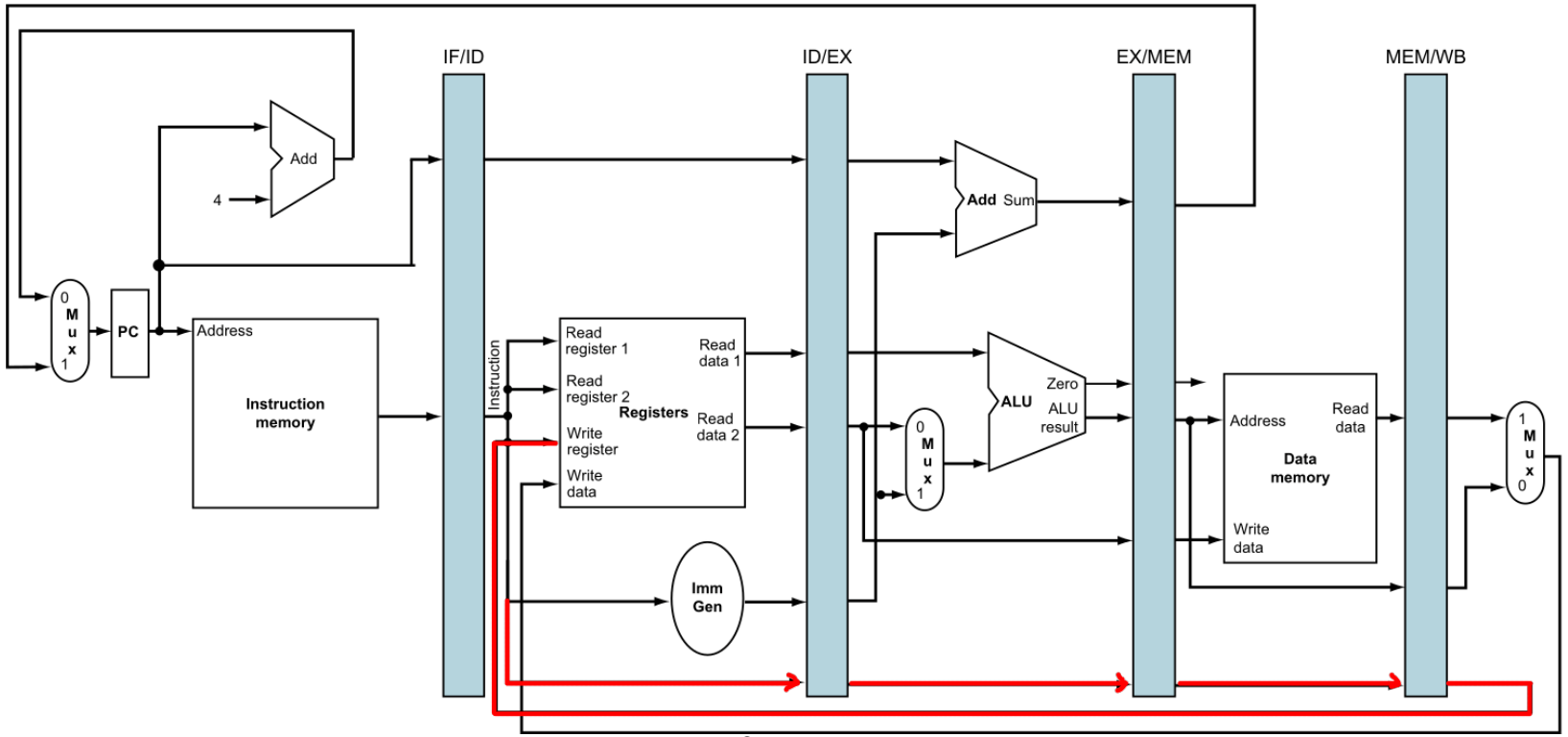


- State of pipeline in a given cycle

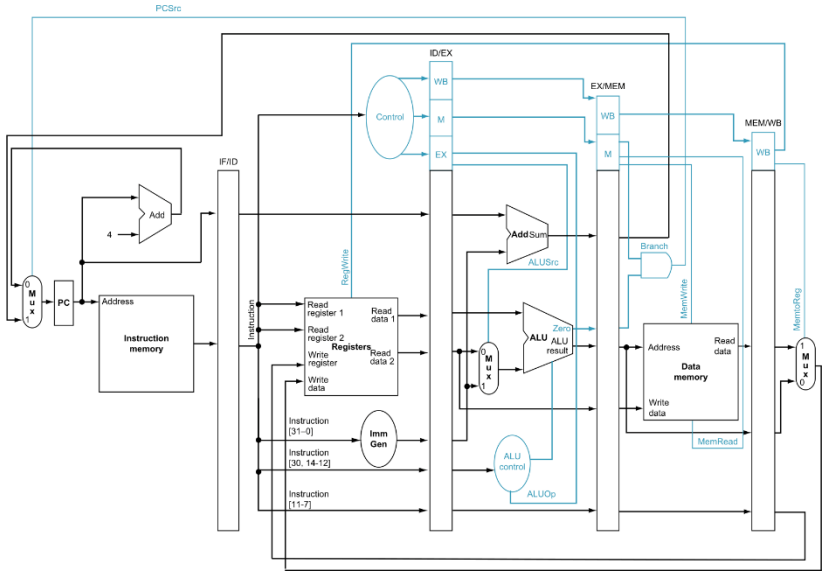
backward half



front half

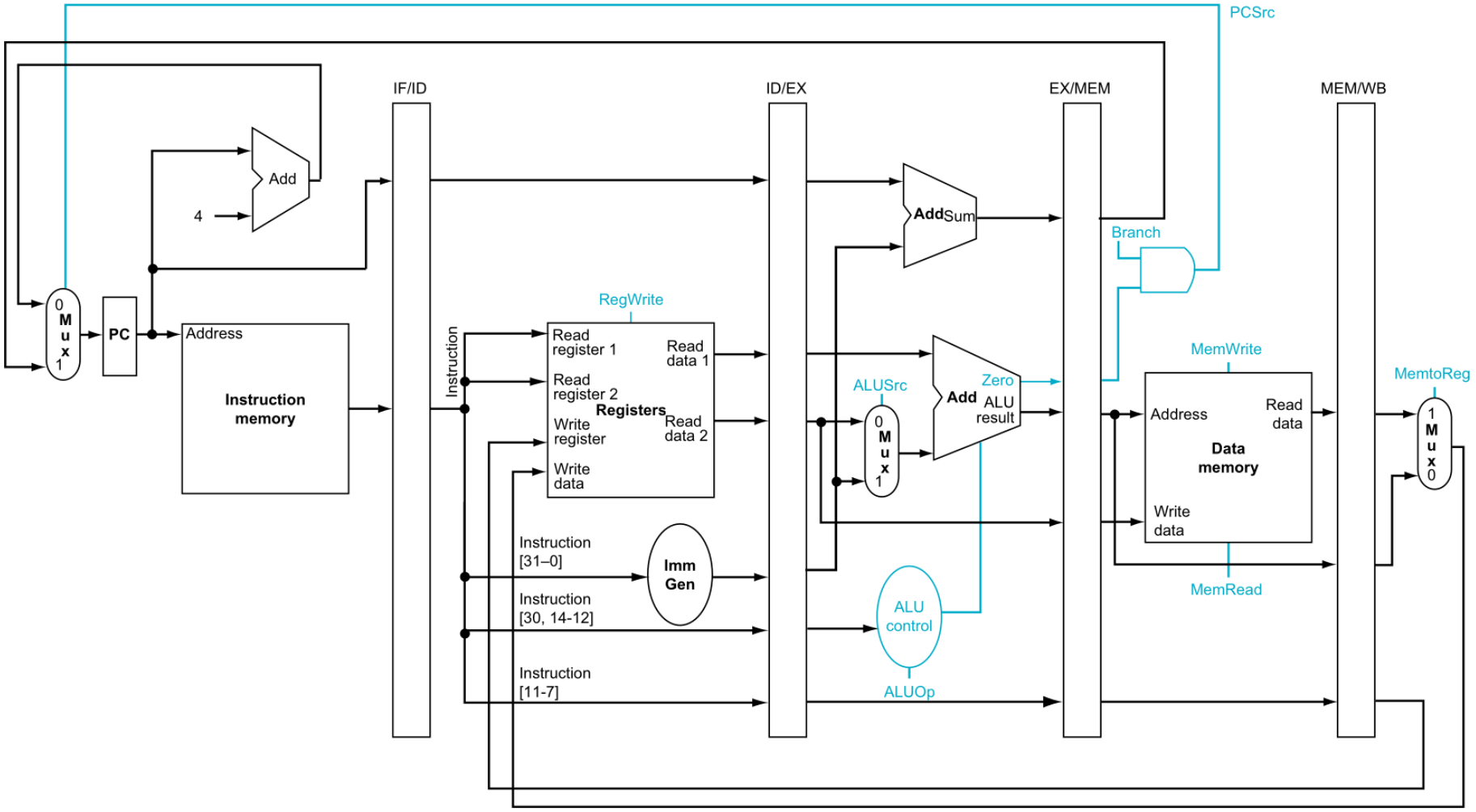


datapath for save register idx



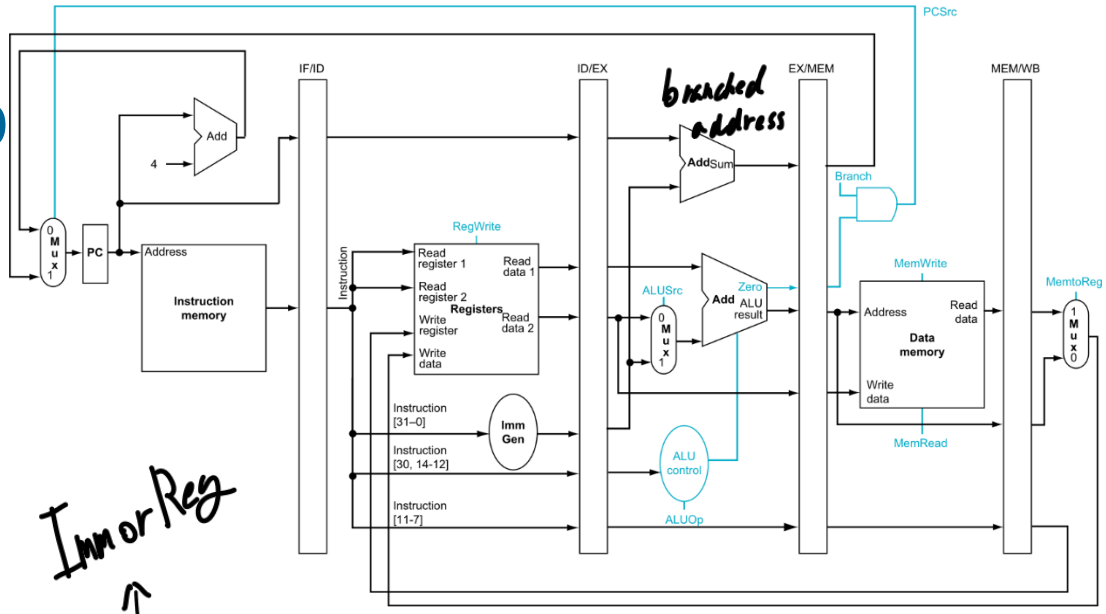
A Pipelined Control Path

Pipelined Control (Simplified)



Control Signals Recap

- **Six 1-bit control signals**
  - Same signals as in single-cycle implementation



Signal	Action when Low	Action when High
RegWrite	None	Write value at *Write data*$_{RF}$ port to register[*Write register*]
ALUSrc	Second ALU operand = *Read data 2*	Second ALU operand = sign-extended, 12 bits of instruction
PCSrc	new PC = old PC + 4	new PC = result of branch target adder
MemRead	None	*Read data* = mem[*Address*]
Mem Write	None	mem[*Address*] = *Write data*$_{DM}$
MemtoReg	*Write data*$_{RF}$ = output of ALU	*Write data*$_{RF}$ = output of data memory

ALU Control Signals Recap

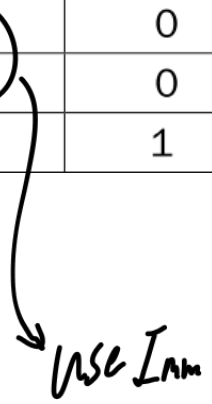
- **Combinational logic derives ALU control**
  - Input
    - 2-bit ALUOp derived from opcode field
    - funct7 and funct3 from instruction
  - Output: ALU control

opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU control
lw	00	load register	XXXXXXX	XXX	add	0010
sw	00	store register	XXXXXXX	XXX	add	0010
beq	01	branch on equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
		subtract	0100000	000	subtract	0110
		AND	0000000	111	AND	0000
		OR	0000000	110	OR	0001

Generating Control Signals

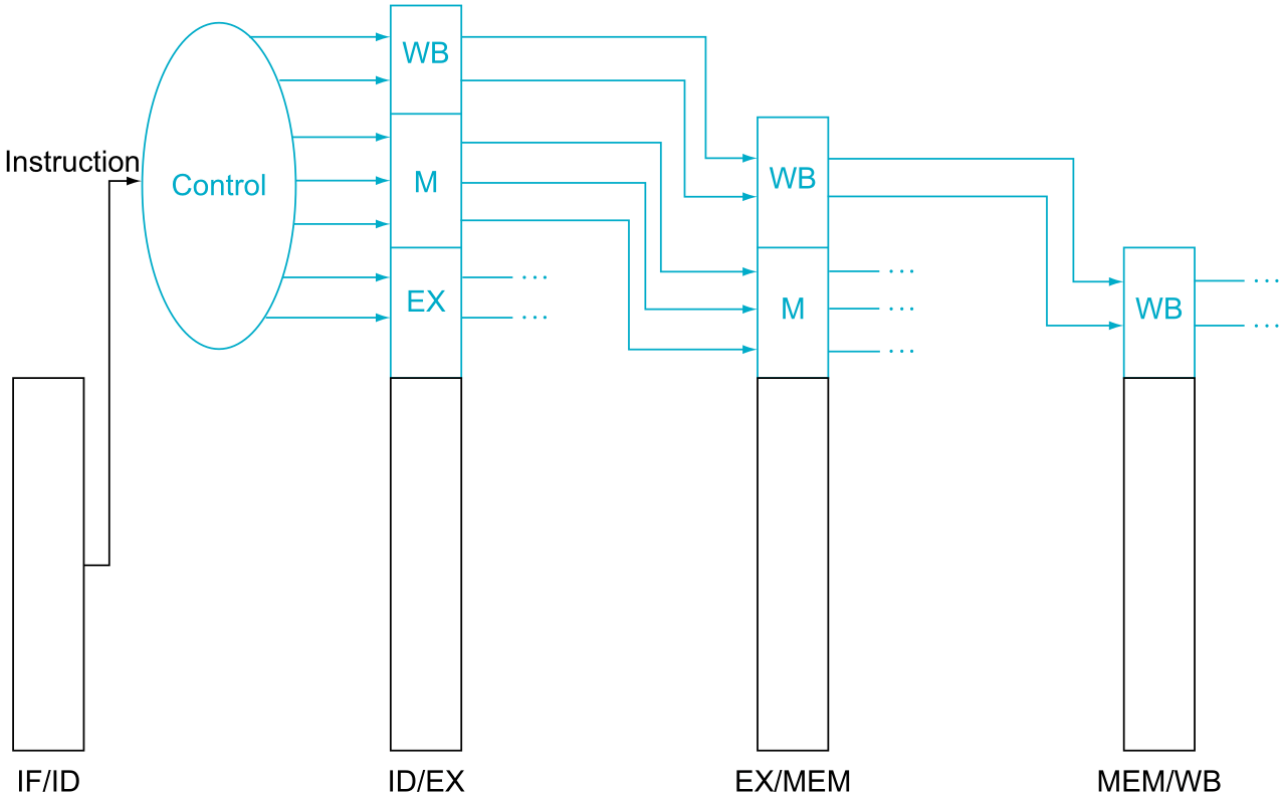
IF: None
ID: None
EX: ALUop/ALU src
MEM: PCSrc/MemRead/MemWrite
WB: MemtoReg/RegWrite

Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
lw	00	1	0	1	0	1	1
sw	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

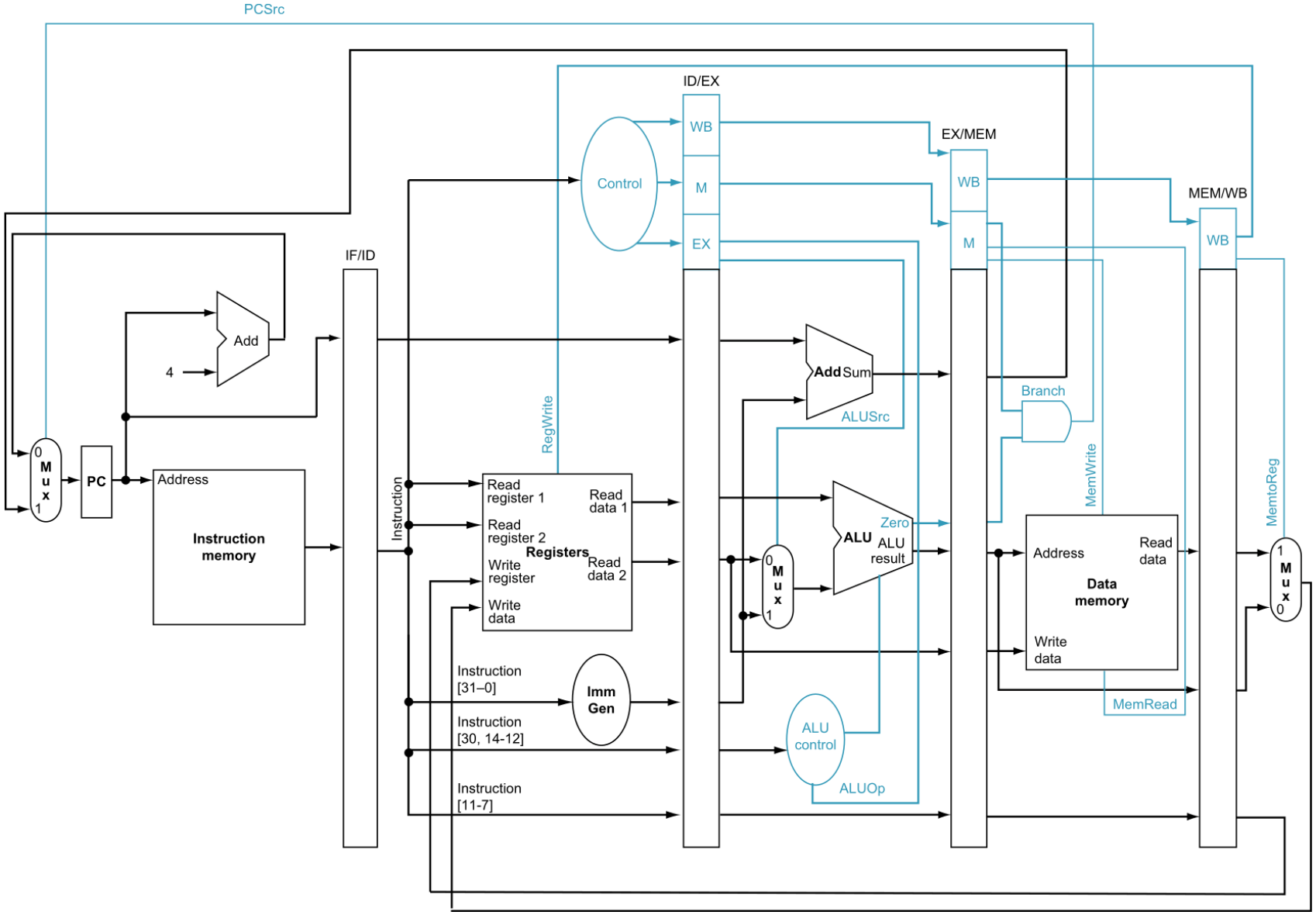


Pipelined Control

- Control signals derived from instruction (ID stage)
  - as in single-cycle implementation
  - conveyed to appropriate stage via pipeline registers

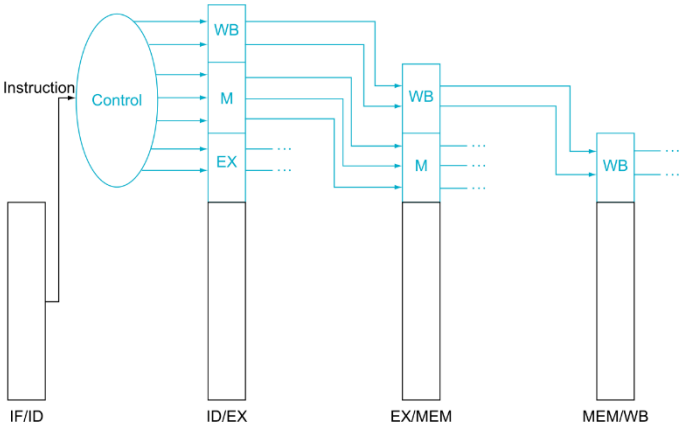


Pipelined Control



RegWrite: write reg or not

MemtoReg: MUXing executed data or memory data



Module Summary

Module Summary

- **Pipelinened datapath**
  - Insert pipeline registers between stages
  - Forward all data signals one stage per cycle
- **Pipelined control path**
  - Pipelined control identical to single-cycle control
  - Control signals forwarded through pipeline registers to respective stage
- Current implementation does not yet handle hazards