EECS4030: Computer Architecture

The Processor (II)

Prof. Chung-Ta King
Department of Computer Science
National Tsing Hua University, Taiwan

(Adapted from textbook slides https://www.elsevier.com/books-and-journals/book-companion/9780128122754/lecture-slides)

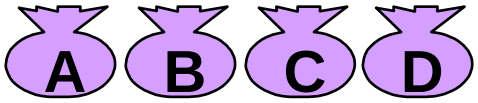
Outline

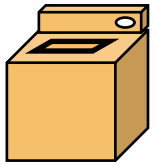
- Problem statement and logic review (Sec. 4.1, 4.2)
- Building a simple RISC-V processor with datapath and control (Sec. 4.3, 4.4)
- Building a pipelined RISC-V processor with datapath and control (Sec. 4.5, 4.6)
- Dealing hazards in pipelined processor: data and control hazards (Sec. 4.7, 4.8)
- Handling exceptions (Sec. 4.9)
- More advanced topics: parallelism via instructions, ARM Cortex-A53 and Intel Core i7 Pipelines, instruction-level parallelism (Sec. 4.10, 4.11, 4.12)

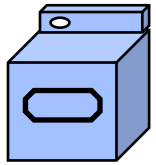
Pipelining Is Natural!

Laundry example:

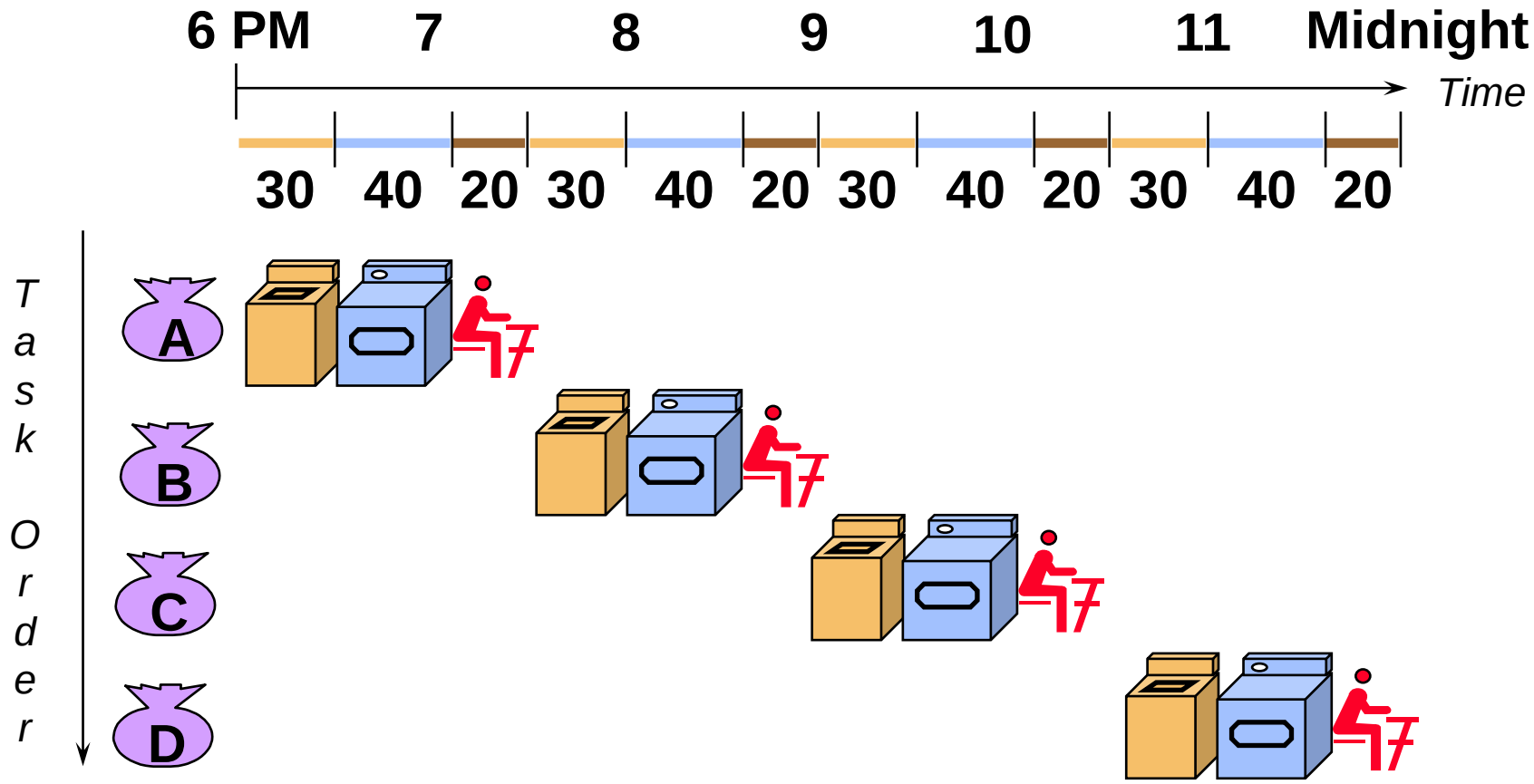
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes





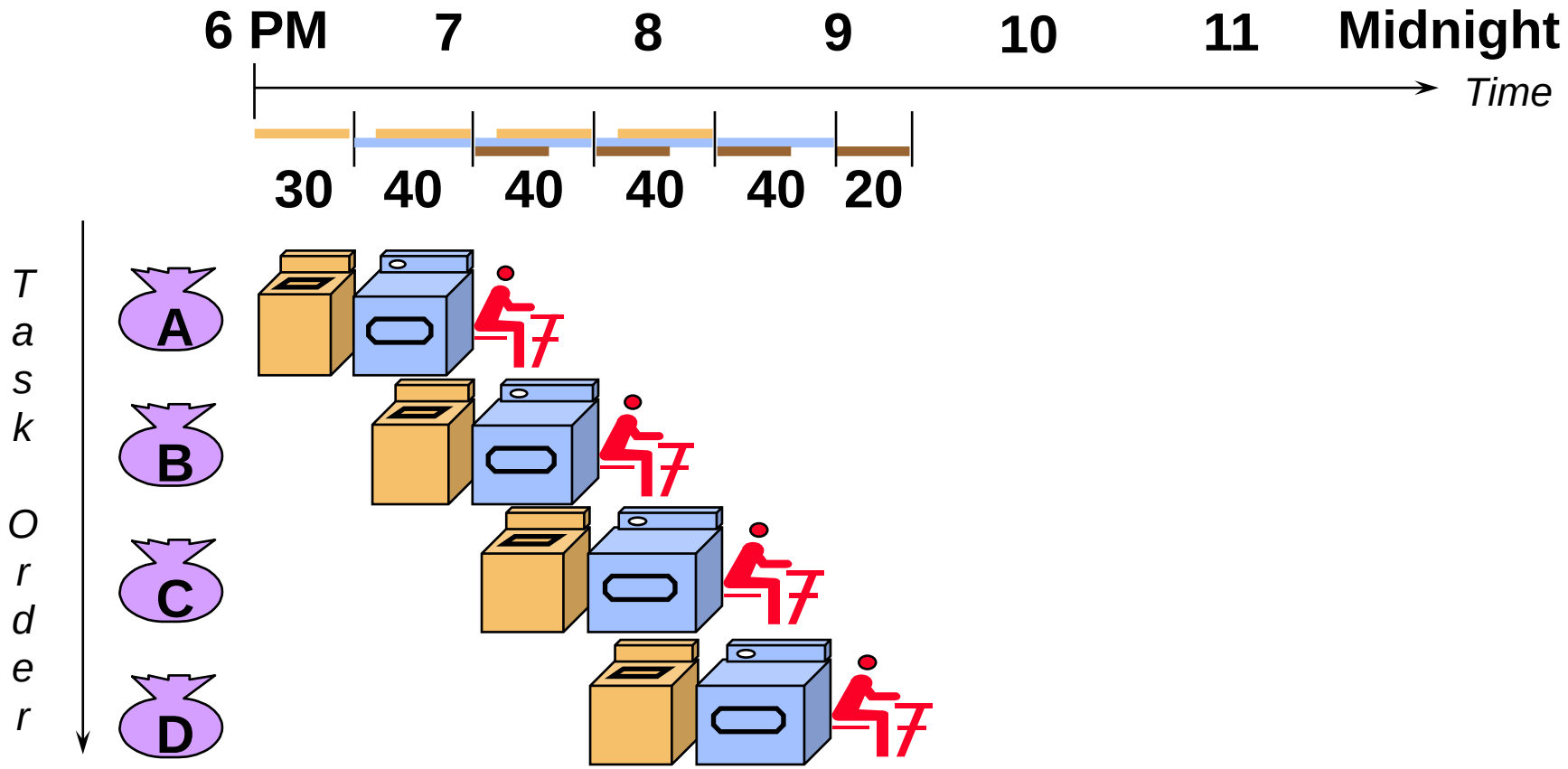


Sequential Laundry



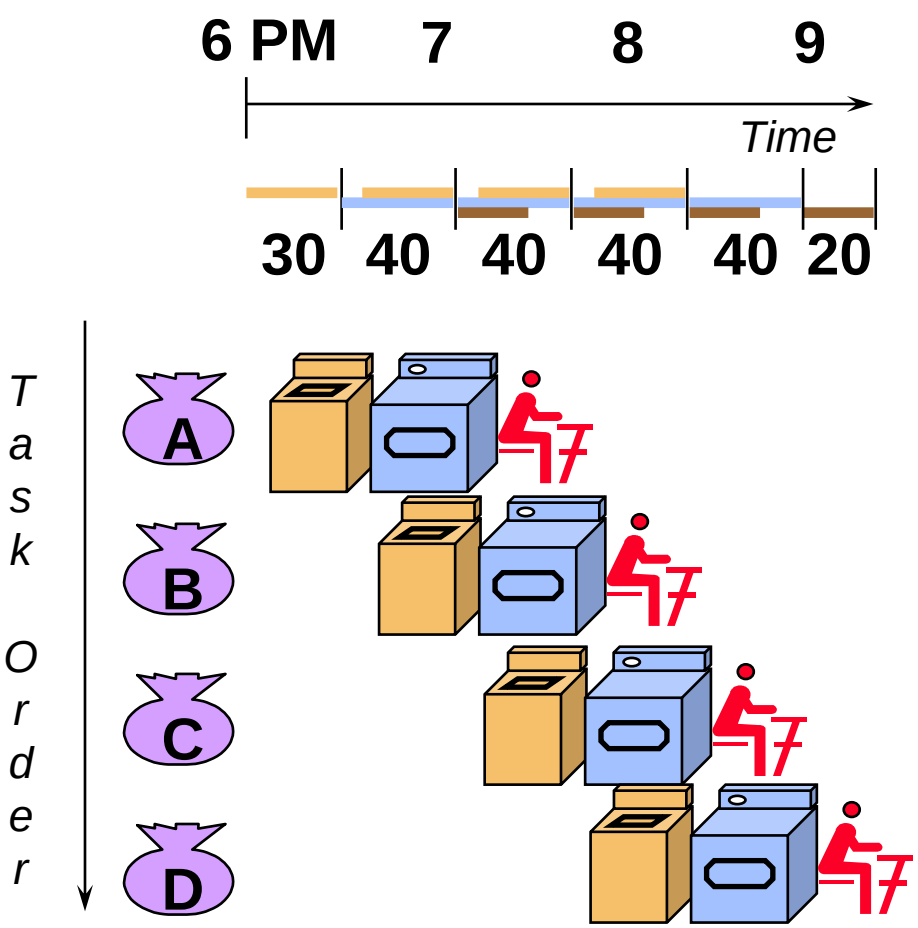
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would it take?

Pipelined Laundry: Start ASAP



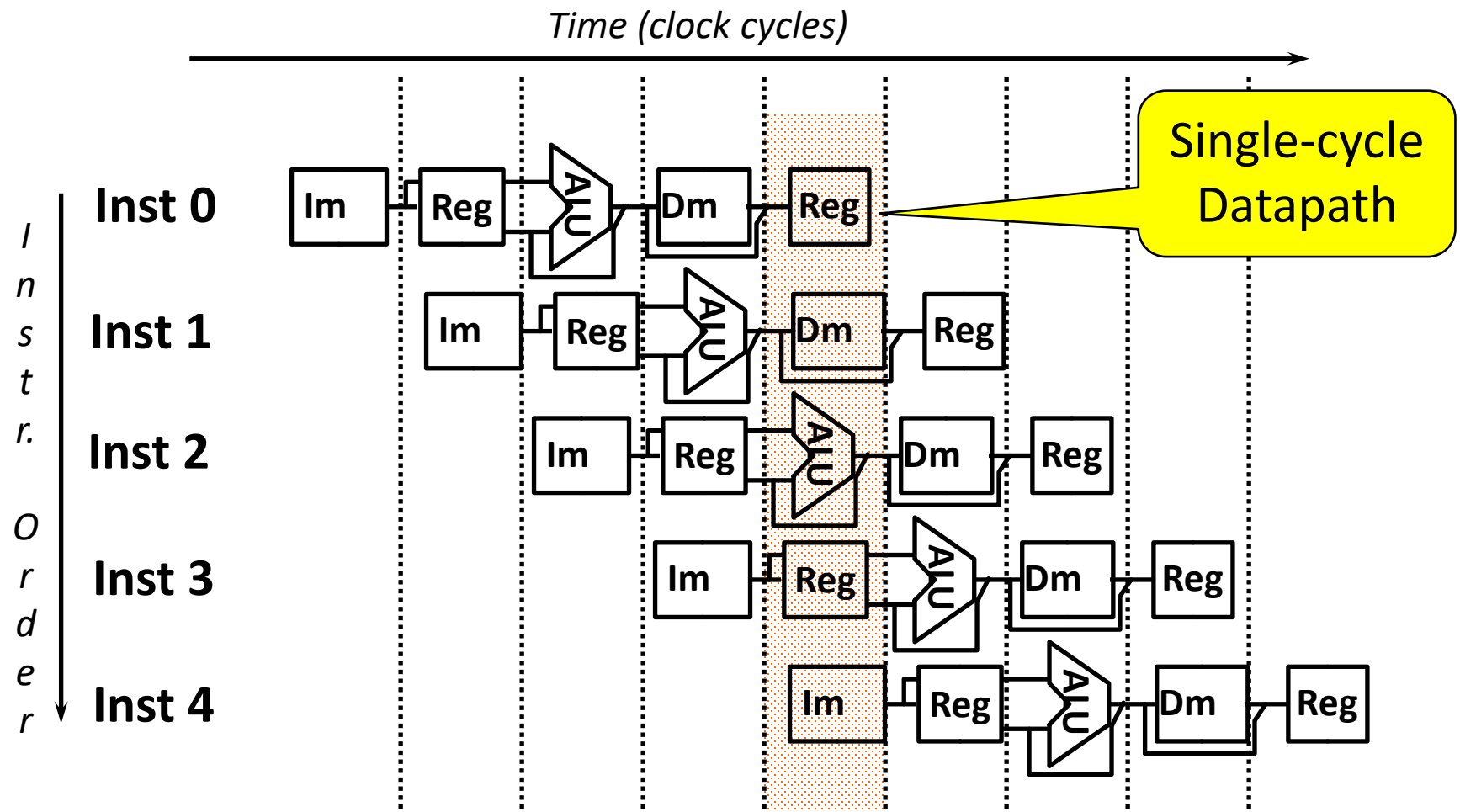
- Pipelined laundry takes 3.5 hours for 4 loads
- Speedup = 8/3.5 = 2.3

Pipelining Lessons



- Multiple tasks working at same time using different resources
- Doesn't help latency of single task, but throughput of entire
- Pipeline rate limited by slowest stage
  → balance stage length
- Ideal speedup = # of stages
  - kN / (k+(N–1)): for k stages
  - Time to "fill" & "drain" the pipeline reduce speedup

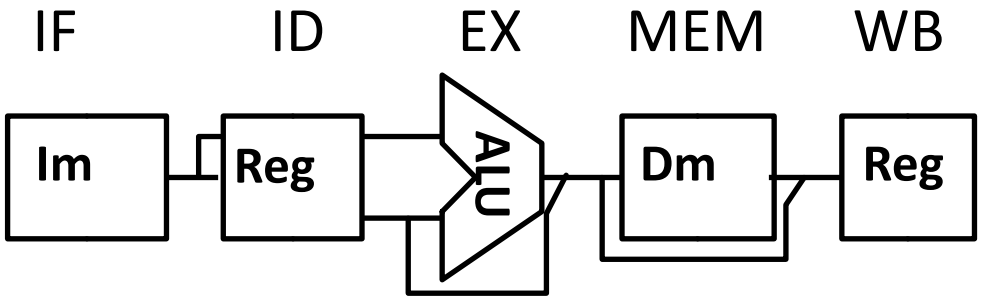
Why Pipeline? Because Resources Are There!



RISC-V Pipeline

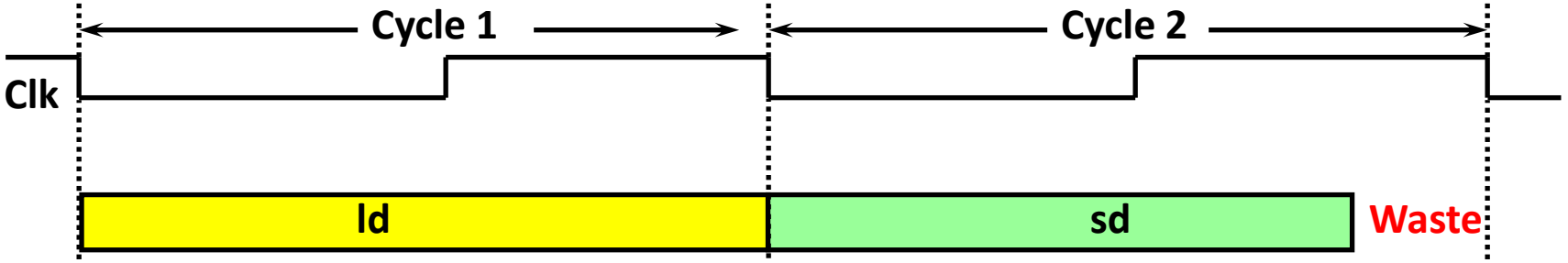
Five stages, one step per stage

- IF: Instruction fetch from memory
- ID: Instruction decode and register read
- EX: Execute operation or calculate address
- MEM: Access memory operand
- WB: Write result back to register

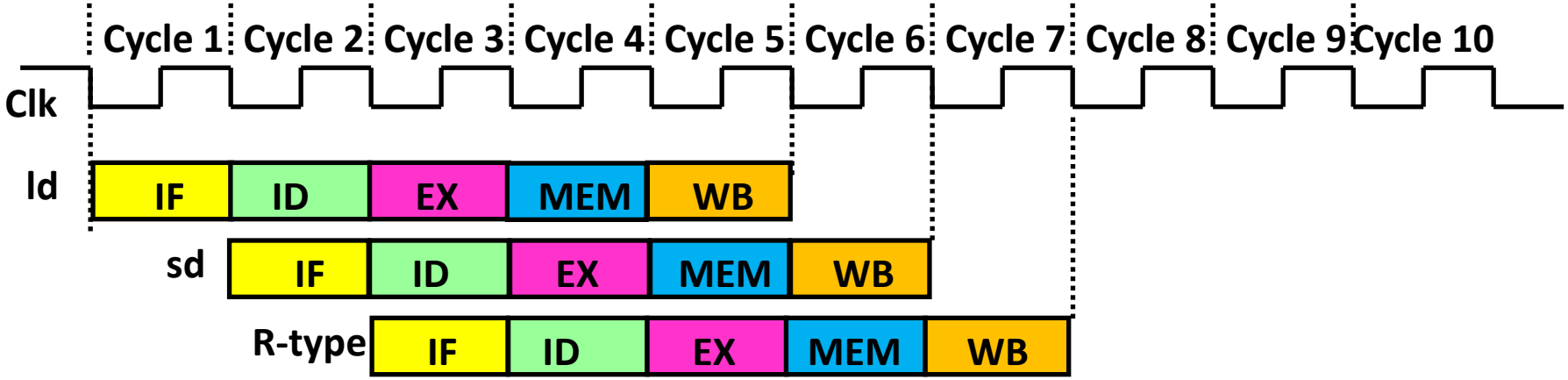


Single-Cycle vs. Pipeline

- Single-cycle implementation:



- Pipeline implementation:



Pipeline Performance

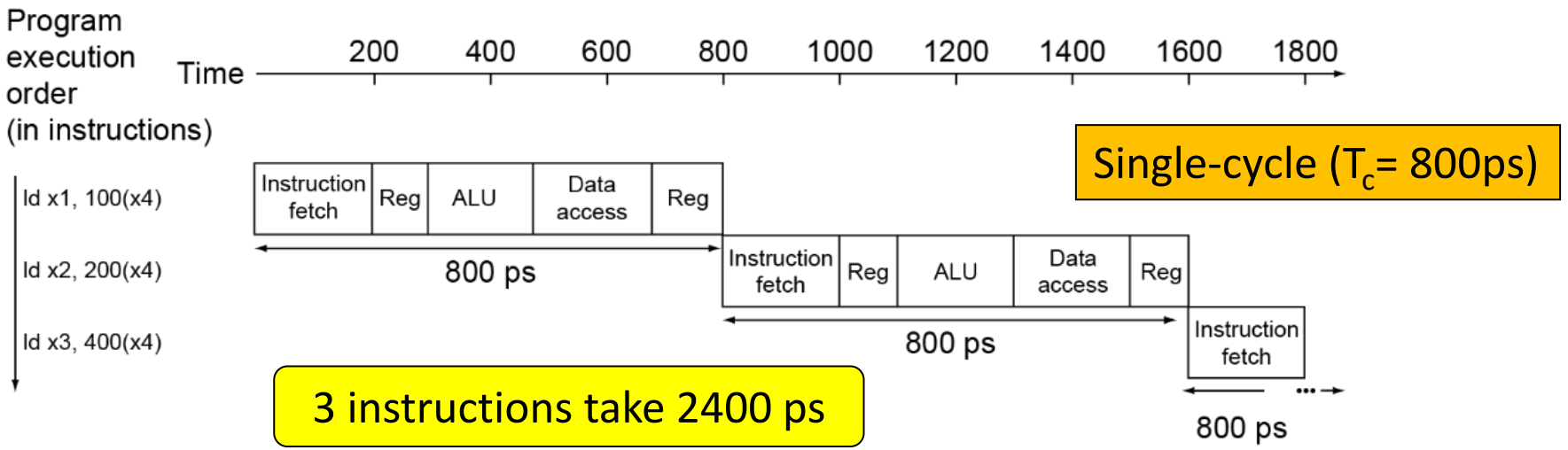
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages

Fig. 4.24

Instr	IF	ID	EX	MEM	WB	Total time
ld	200ps	100 ps	200ps	200ps	100 ps	800ps
sd	200ps	100 ps	200ps	200ps		700ps
R-type	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- Compare pipelined with single-cycle datapath
  - Slowest stage vs. slowest instruction

Pipeline Performance



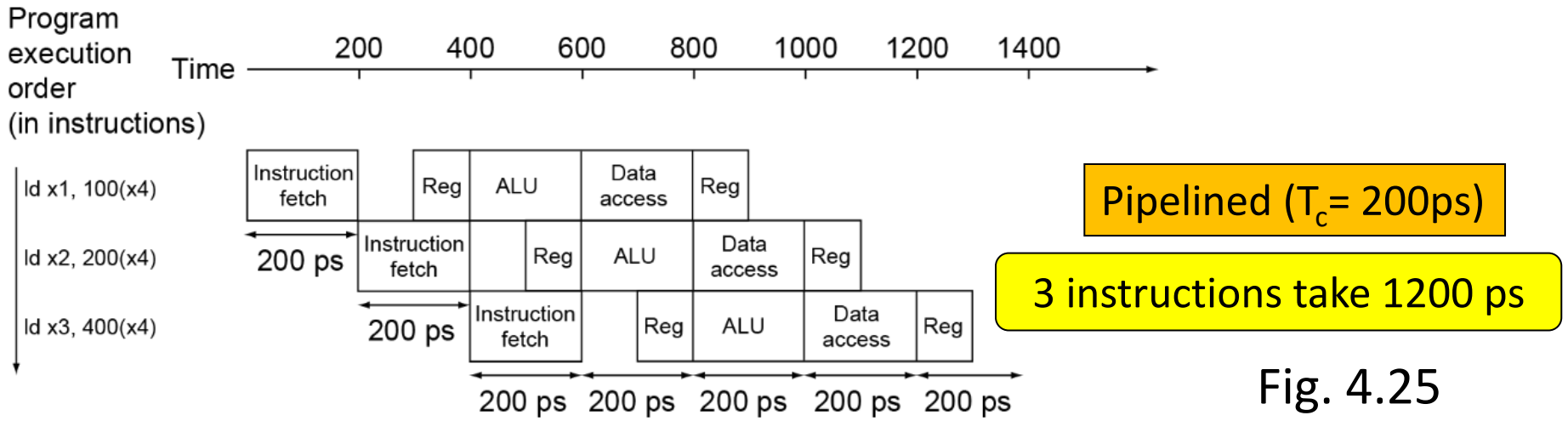


Fig. 4.25

Pipeline Speedup

- If all stages are balanced,
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$
    $= \frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}} = \frac{800 \text{ ps}}{5} = 160 \text{ ps}$
  - Speedup for N ld instructions
    $= \frac{N * 800 \text{ ps}}{800 \text{ ps} + (N-1)*160 \text{ ps}} \cong 5$ (when N is large)
- If not balanced (as in previous slide), speedup is less
  - $(N * 800) / (800 + (N-1)*200) \cong 4$
- Speedup of pipeline is due to increased throughput
  - Latency (time for each instruction) does not decrease

Design a Pipelined RISC-V Processor

RISC-V ISA designed for pipelining

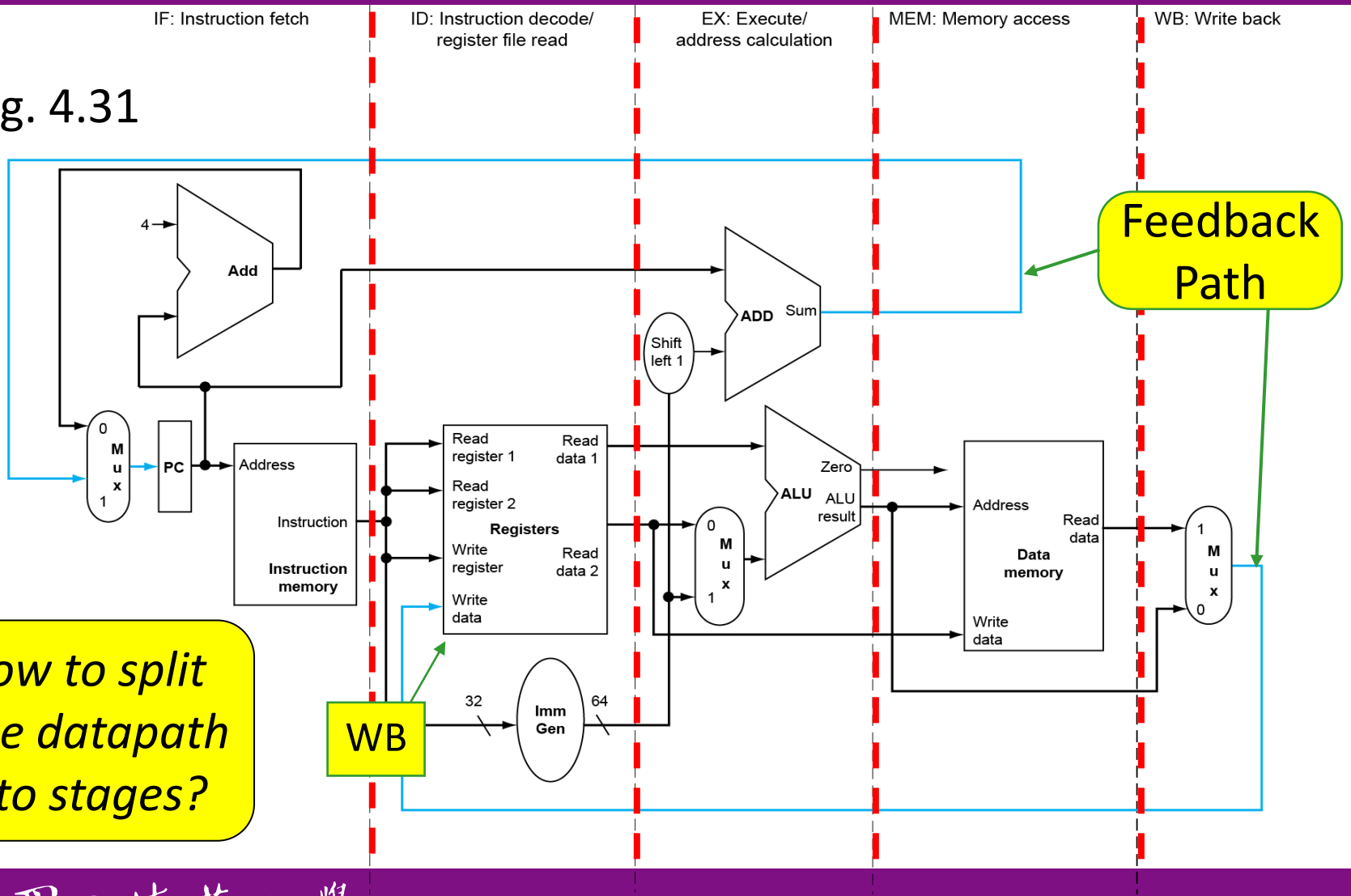
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - c.f. x86: 1- to 17-byte instructions
- Few and regular instruction formats
  - Can decode and read registers at the same step
- Load/store addressing
  - Can calculate address in the 3rd stage, access memory in the 4th stage

Design a Pipelined RISC-V Processor

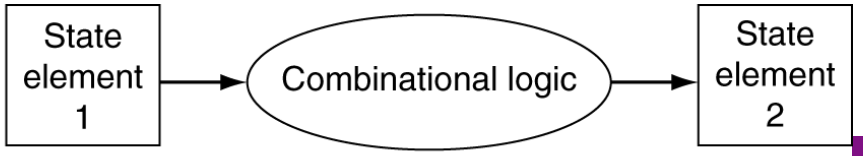
- Examine the datapath and control diagram
  - Starting with single-cycle datapath
- Partition datapath into stages:
  - 5 stages: IF (instruction fetch), ID (instruction decode and register file read), EX (execution or address calculation), MEM (data memory access), WB (write back)
- Associate resources with stages
- Ensure that flows do not conflict, or figure out how to resolve
- Assert control in appropriate stage

Split Single-cycle Datapath

Fig. 4.31



Recall Signal Propag



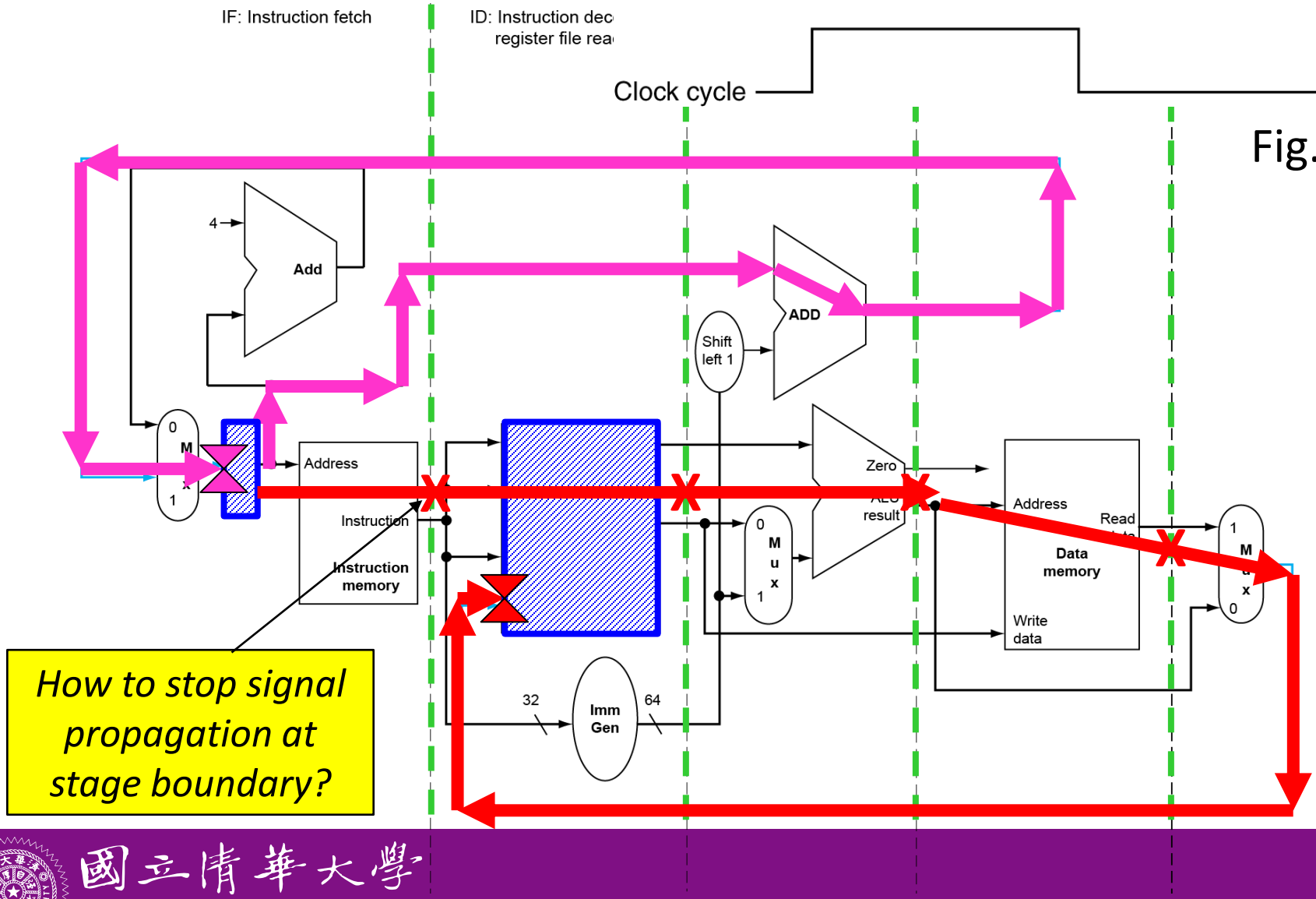
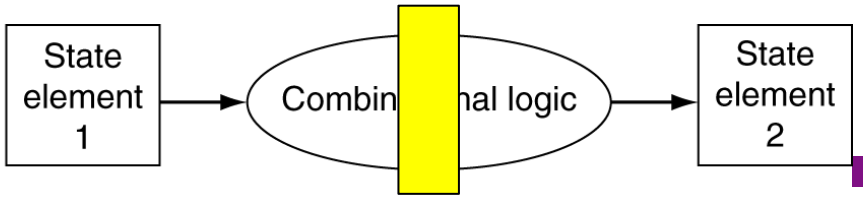


Fig. 4.3

Idea: Add Registers



- *Pipeline registers* to ho
  "state" of an instruction

Fig. 4.3

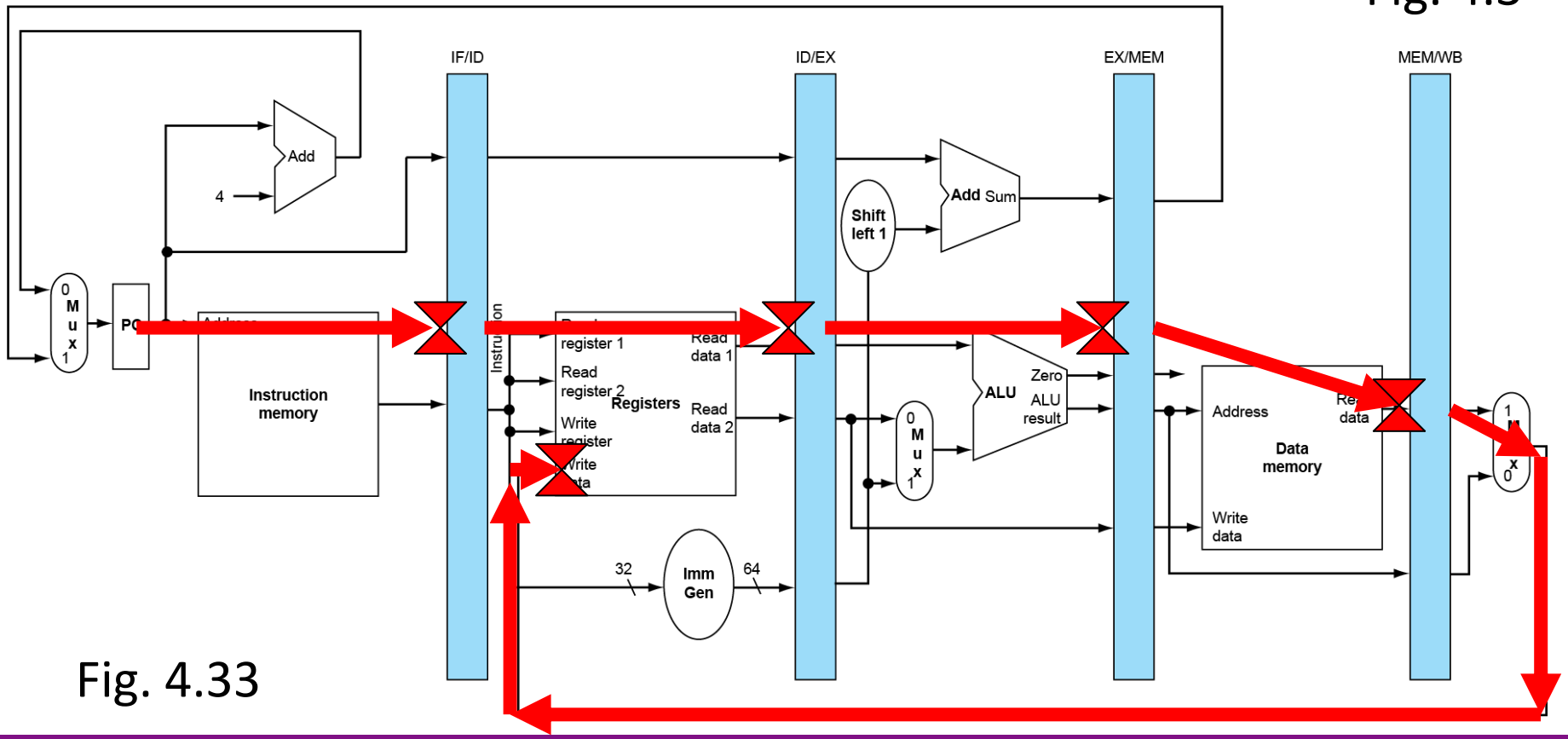


Fig. 4.33

IF Stage for ld

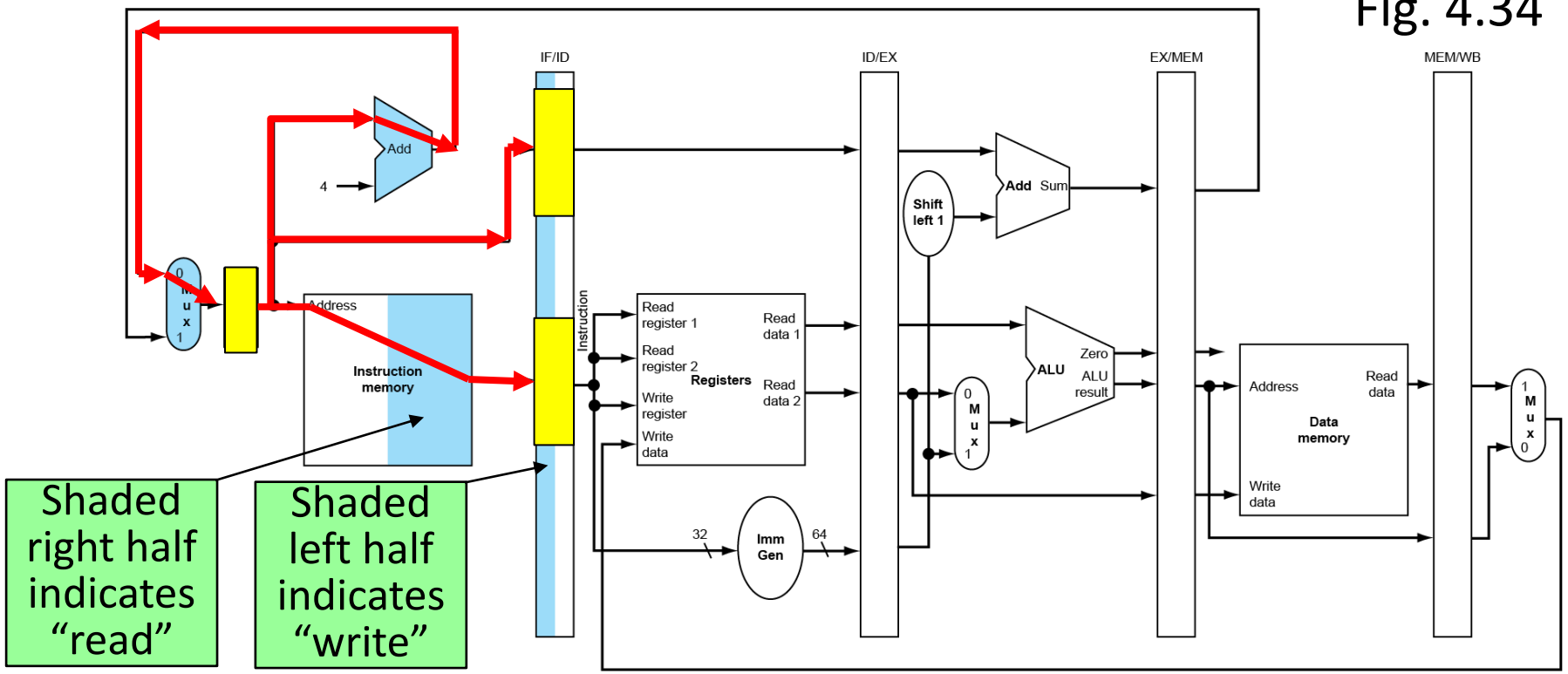
- IF/ID ← mem[PC]; IF/ID ← PC; PC ← PC + 4

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

ld

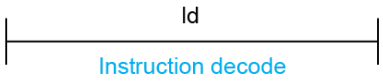
Instruction fetch

Fig. 4.34



ID Stage for ld

- ID/EX ← IF/ID[PC]; ID/EX[A] ← Reg[IF/ID[19-15]]; ID/EX[B] ← Reg[IF/ID[24-20]]; ID/EX ← sign-ext(IF/ID[31-20]); decode



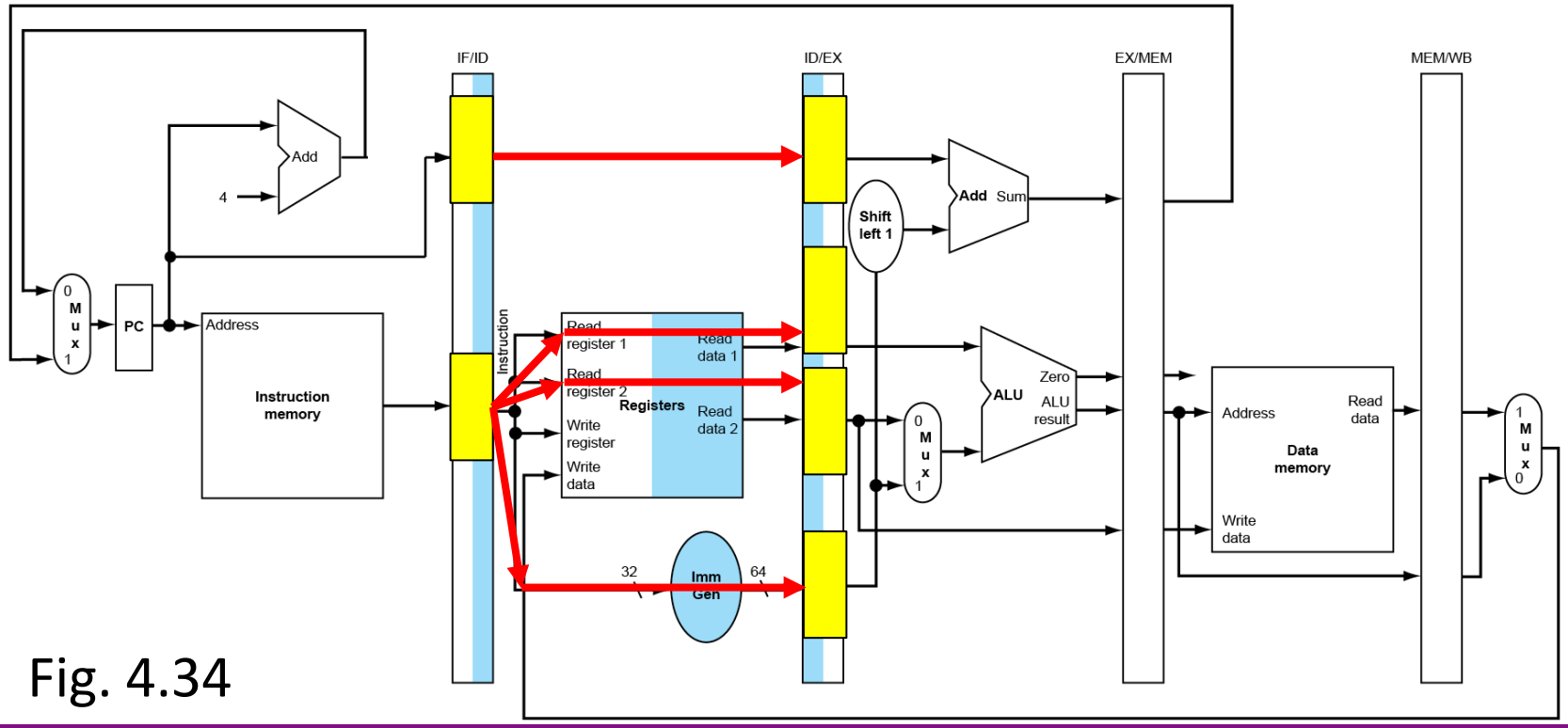


Fig. 4.34

EX Stage for ld

- EX/MEM[ALUout] ← ID/EX[A] + ID/EX[sign-ext]

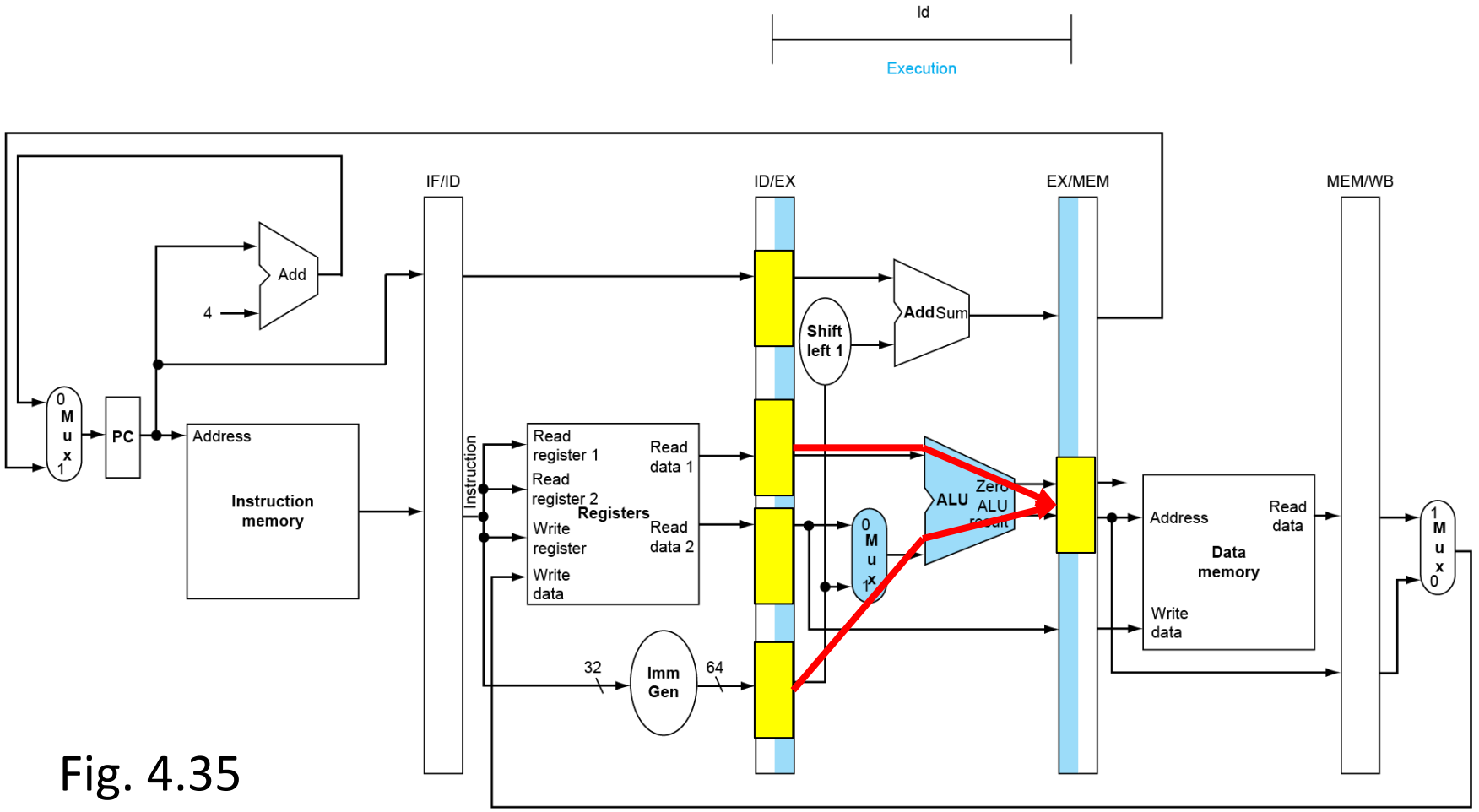


Fig. 4.35

MEM Stage for ld

- MEM/WB ← mem[EX/MEM[ALUout]]

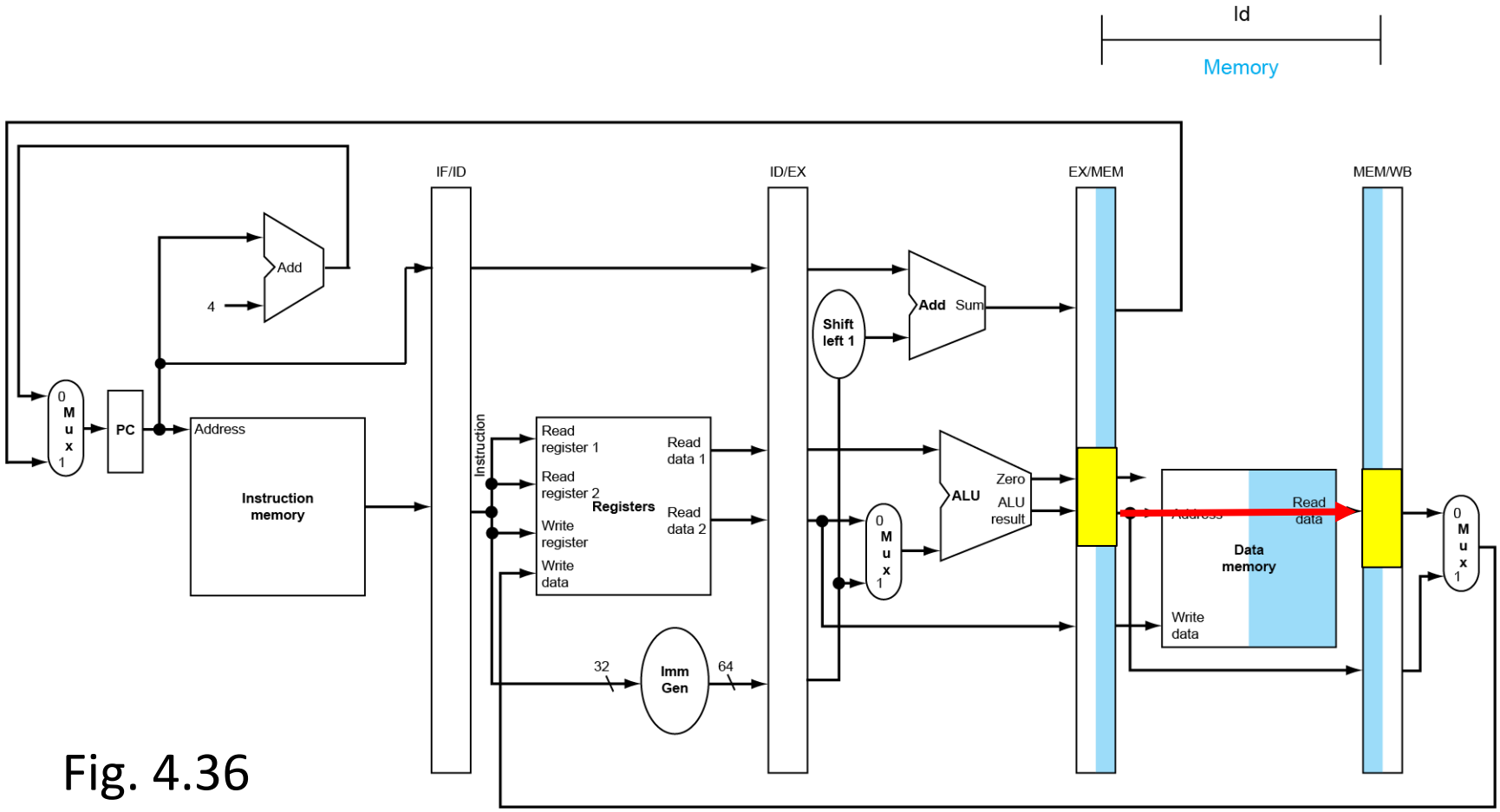
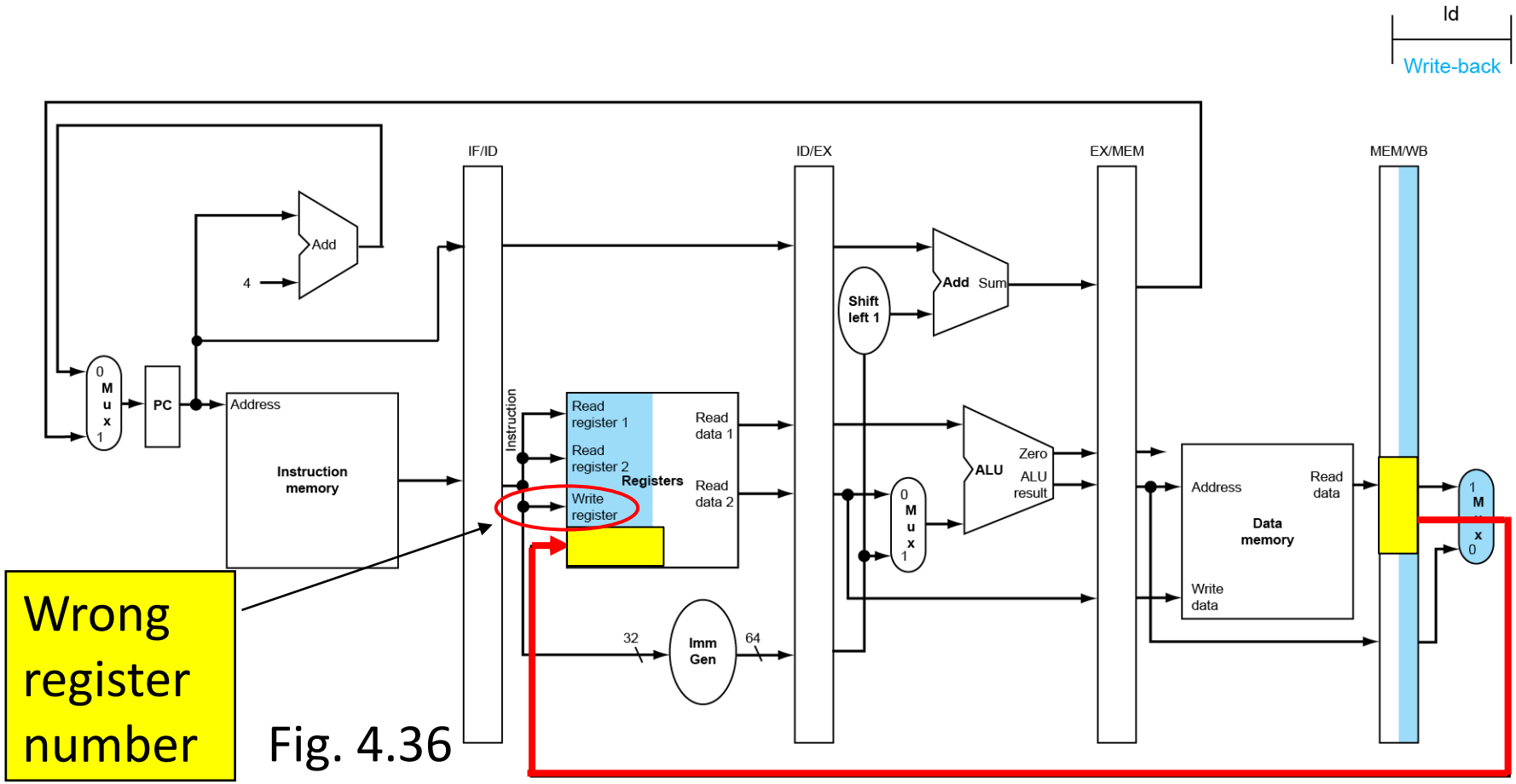


Fig. 4.36

WB Stage for ld

- Reg[IF/ID[11-7]] ← MEM/WB



Wrong register number

Fig. 4.36

Corrected Datapath for ld

- Carry rd along pipeline stages with the instruction

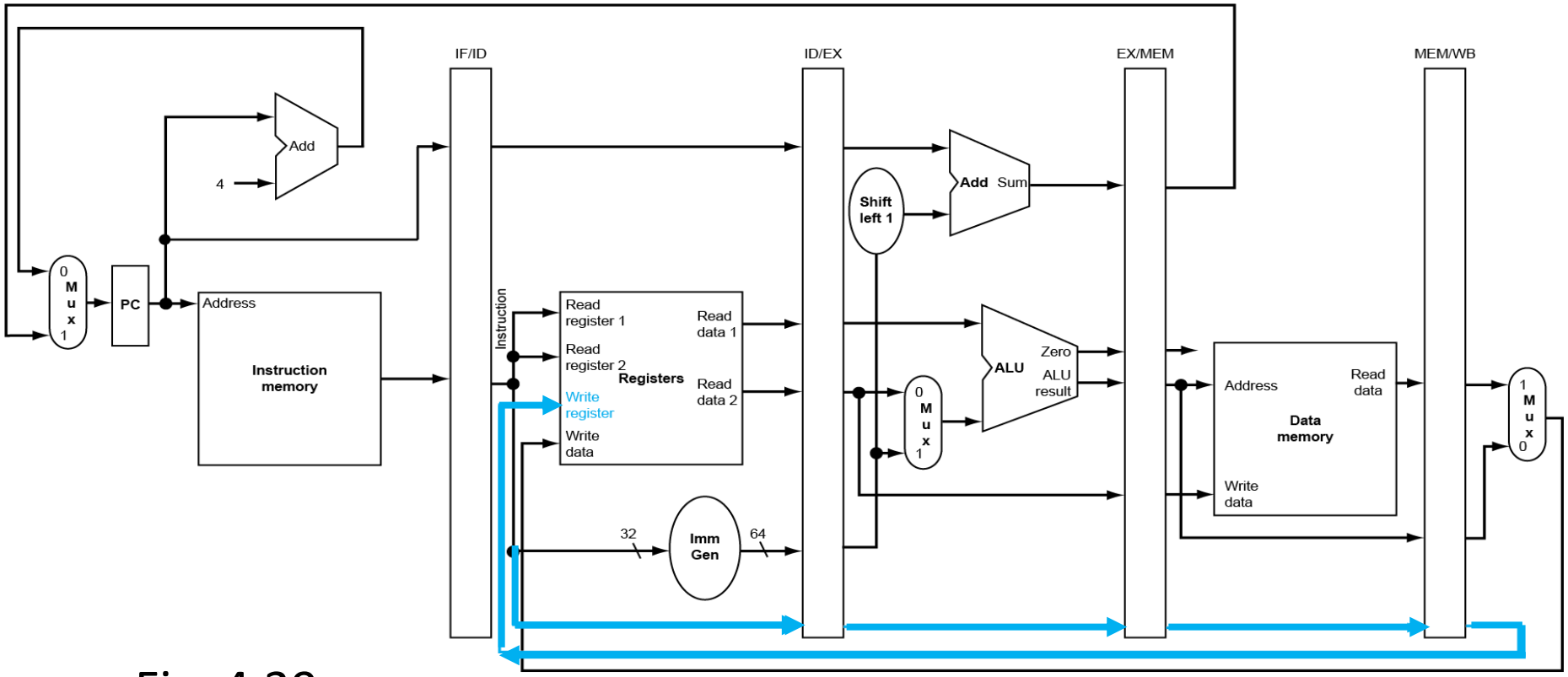


Fig. 4.39

EX Stage for sd

- EX/MEM[ALUout] ← ID/EX[A] + ID/EX[sign-ext]
  EX/MEM ← ID/EX[B]

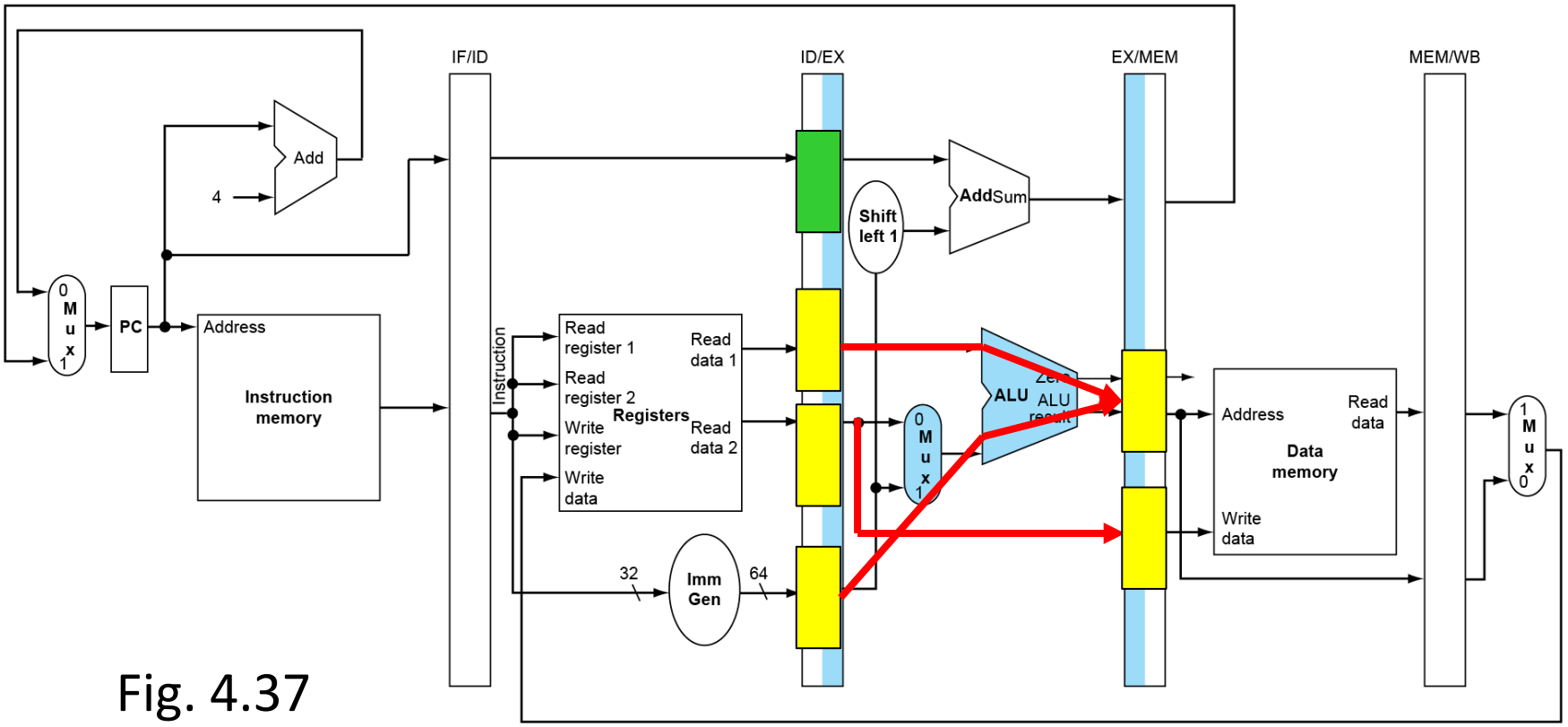
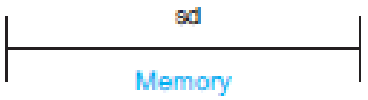


Fig. 4.37

MEM Stage for sd

- mem[EX/MEM[ALUout]] ← EX/MEM[B]



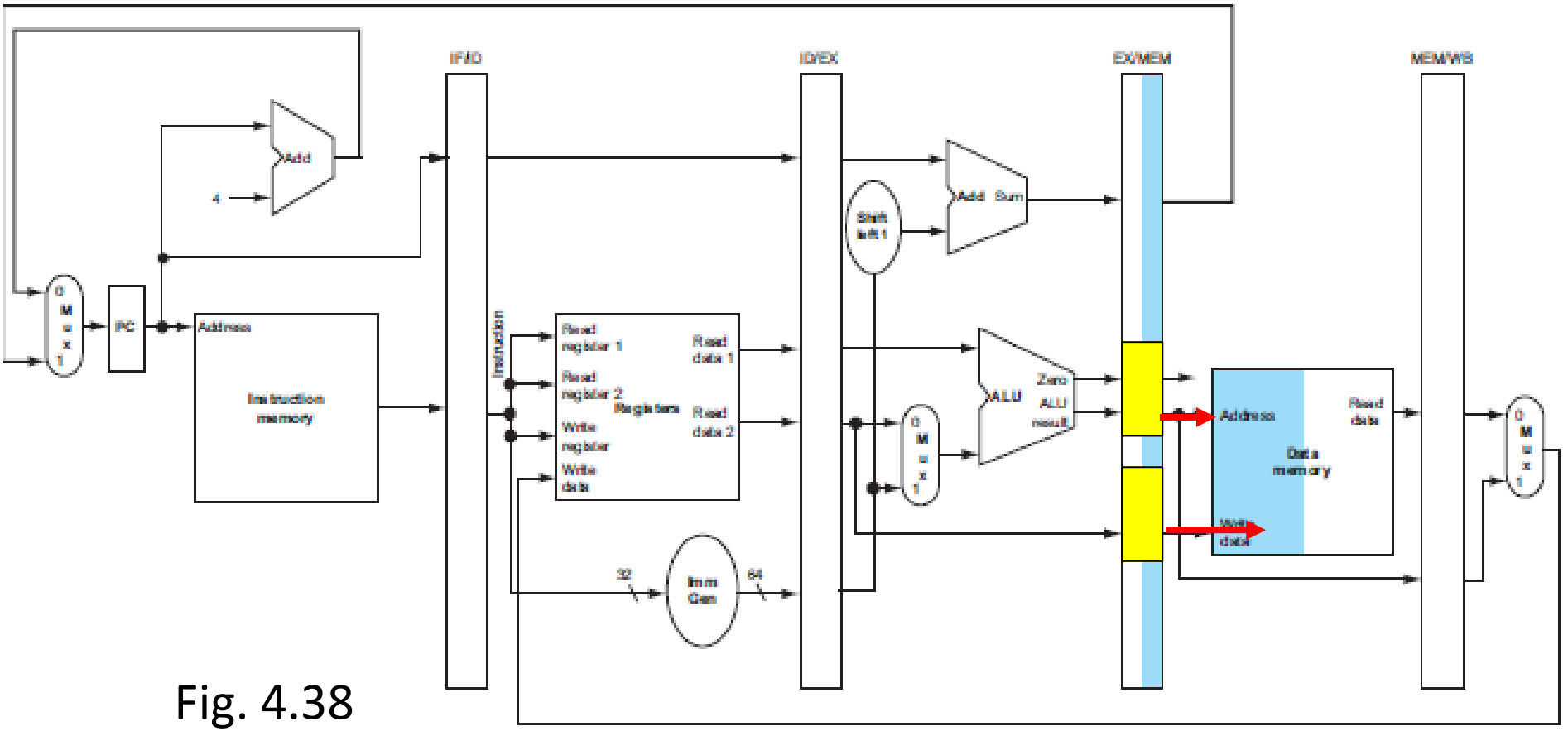


Fig. 4.38

WB Stage for sd

- No operation

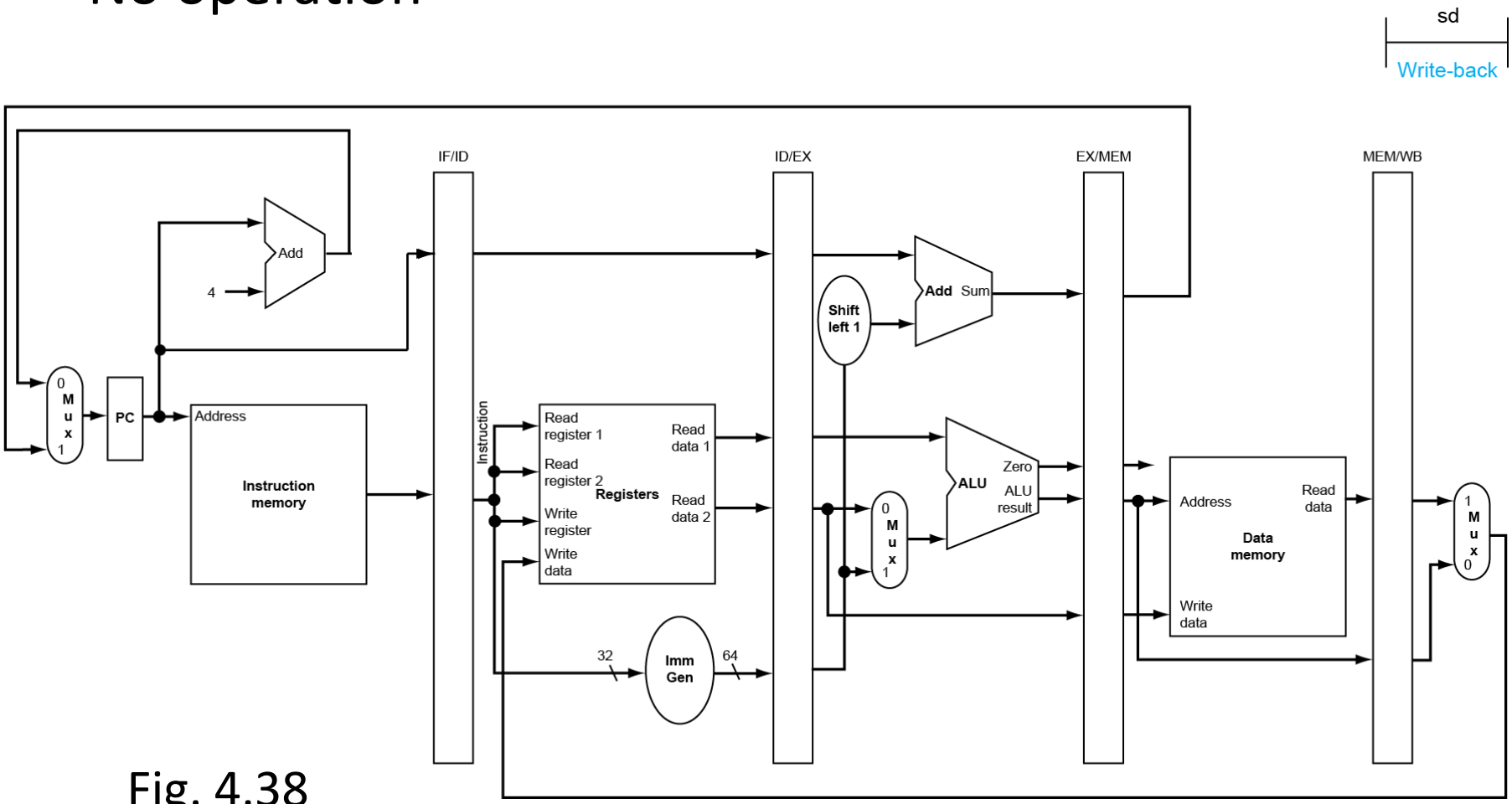
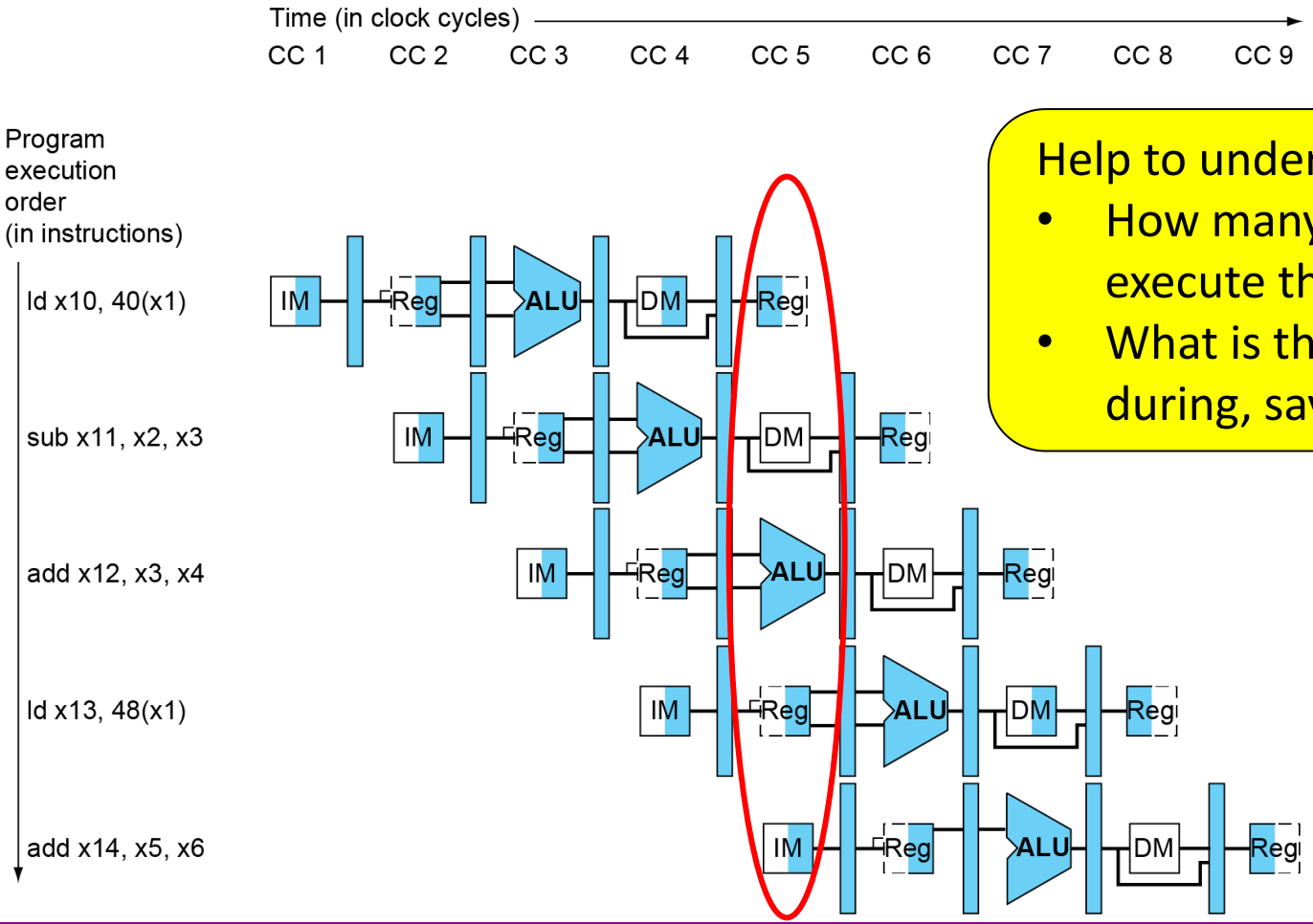


Fig. 4.38

Multi-Cycle Pipeline Diagram

- Form showing resource usage



Help to understand:

- How many cycles to execute this code?
- What is the ALU doing during, say, cycle 5?

Fig. 4.41

Multi-Cycle Pipeline Diagram

- Traditional form

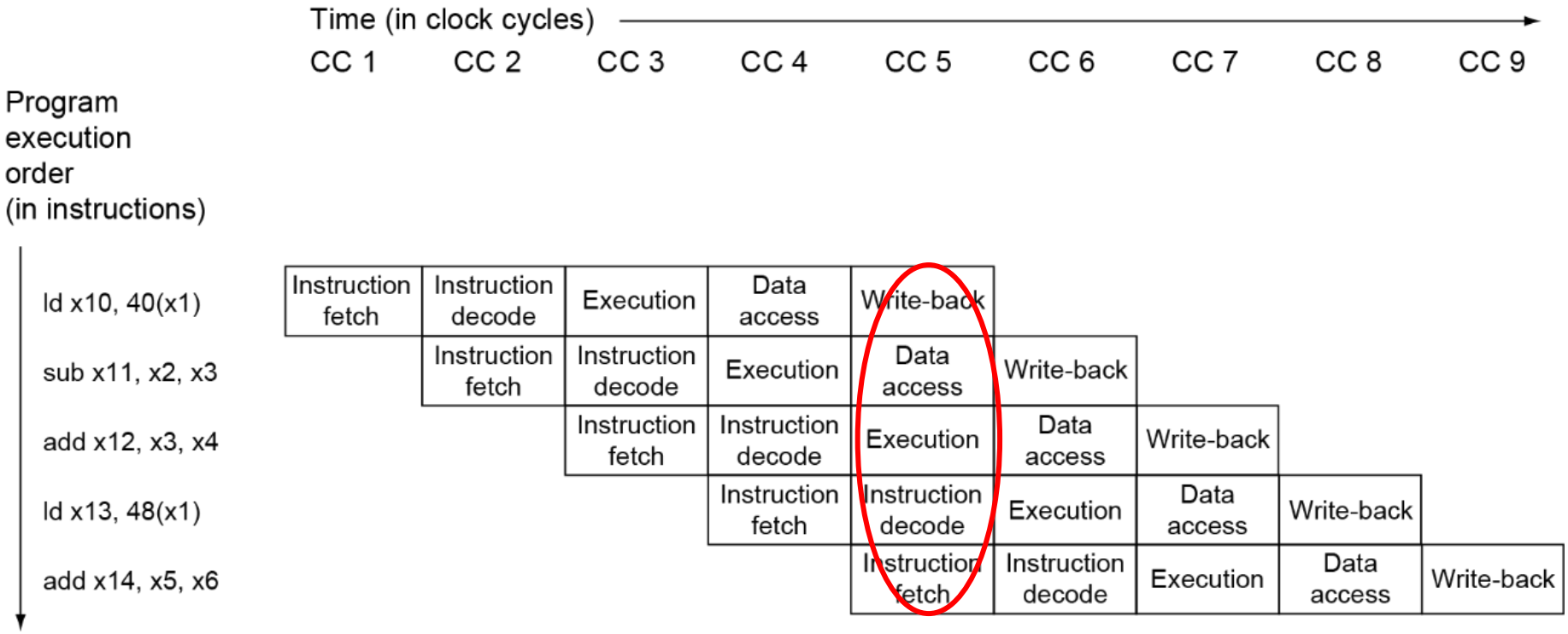


Fig. 4.42

A Snapshot of the Pipeline

- State of pipeline in cycle 5 of previous slide

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

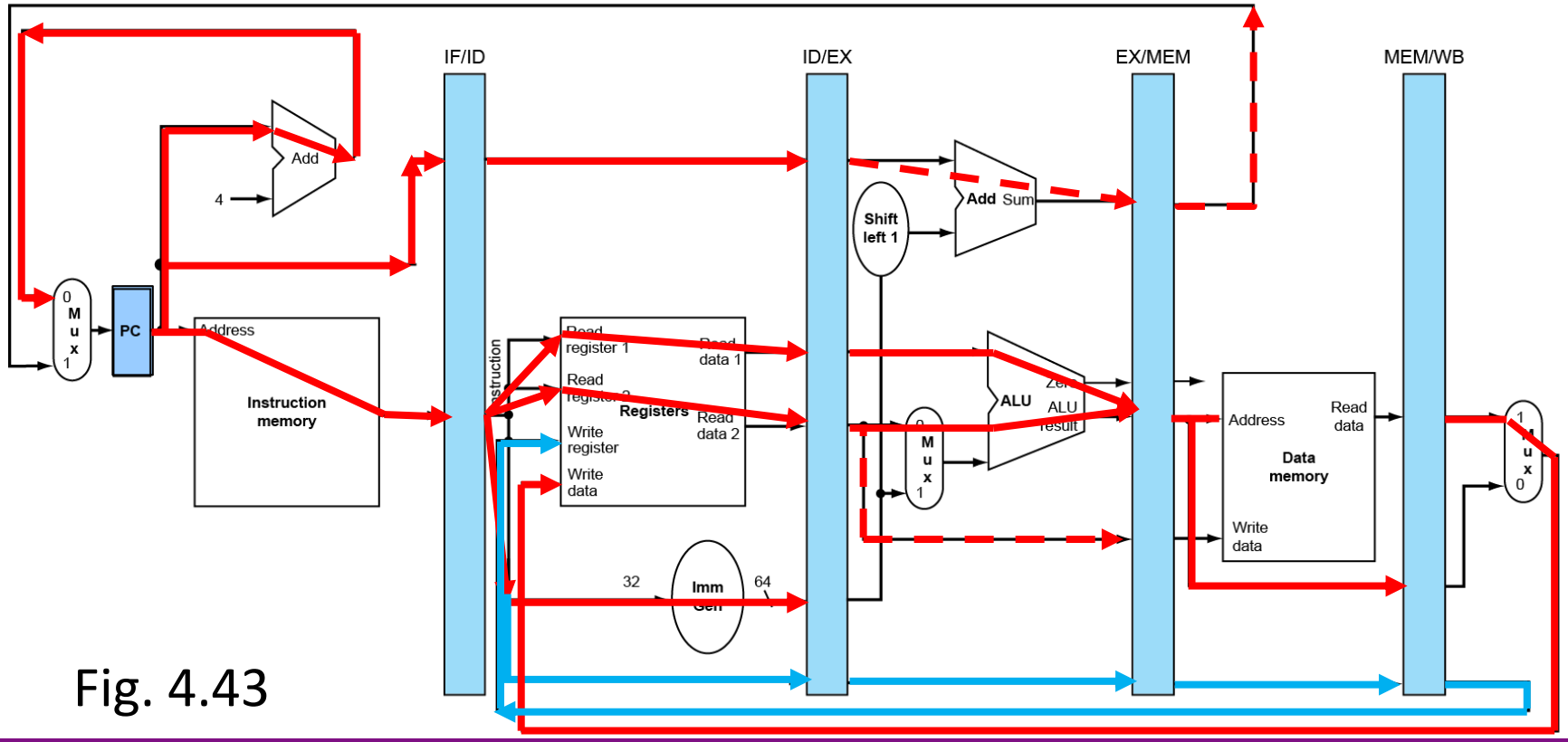


Fig. 4.43

Pipelined Control: Control Signals

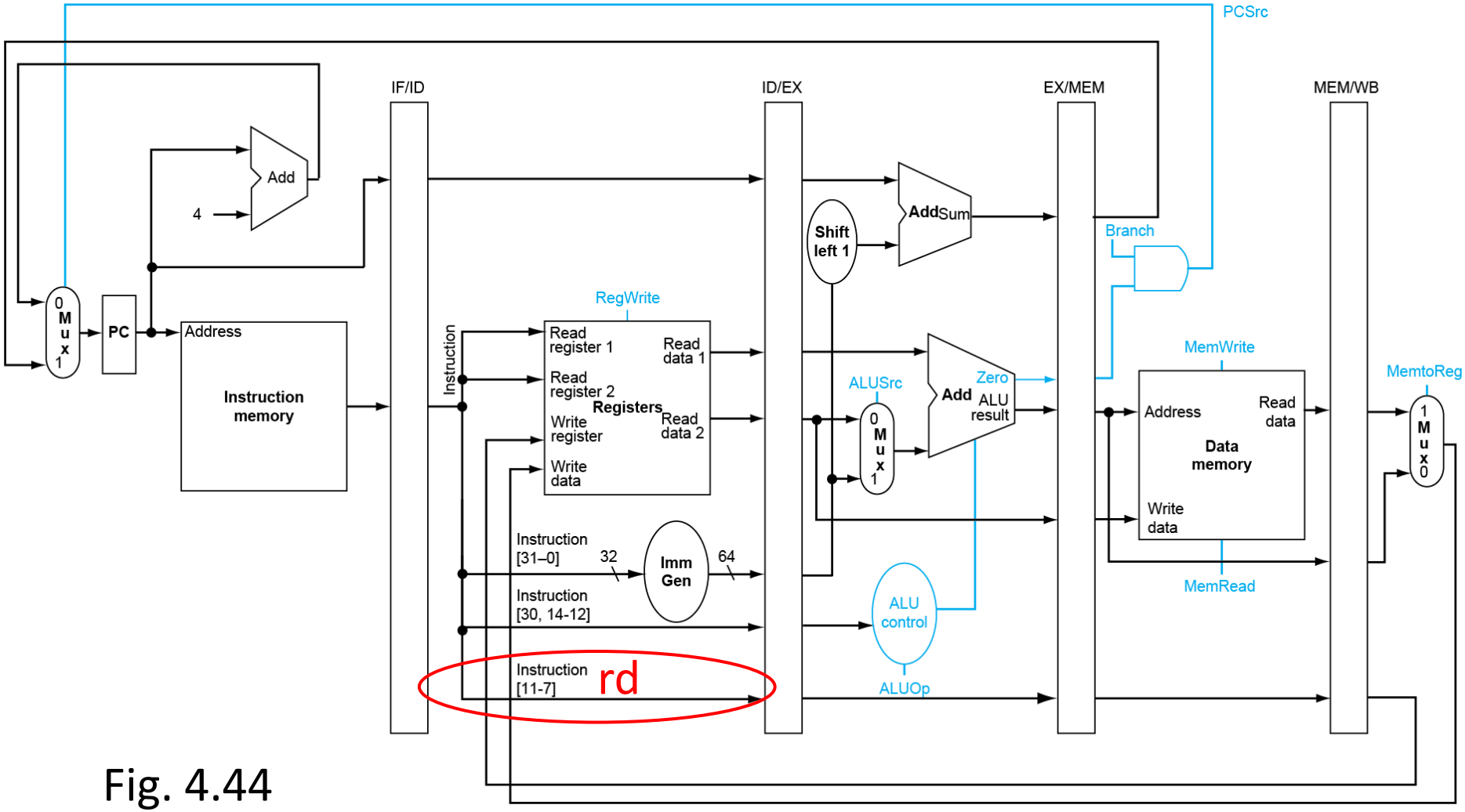


Fig. 4.44

Group Signals According to Stages

- Use control signals of single-cycle CPU

	EX Stage			MEM Stage			WB Stage	
	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
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

Fig. 4.47

  - Note: There is no need to set PCsrc, because it is controlled by the “Branch” signal

Pipelined Control

- Pass control signals along pipeline registers like data
  - Main control generates control signals during ID

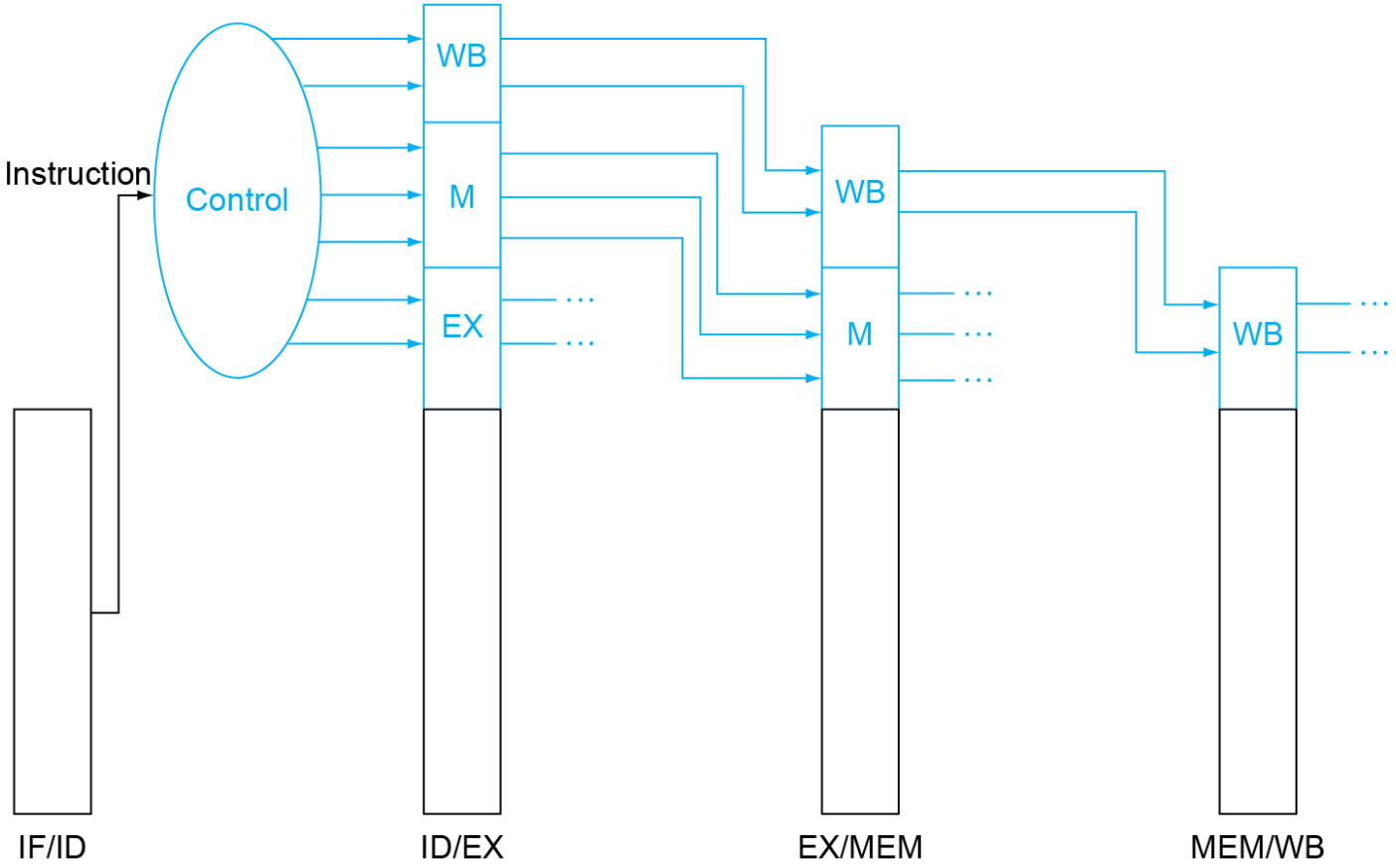
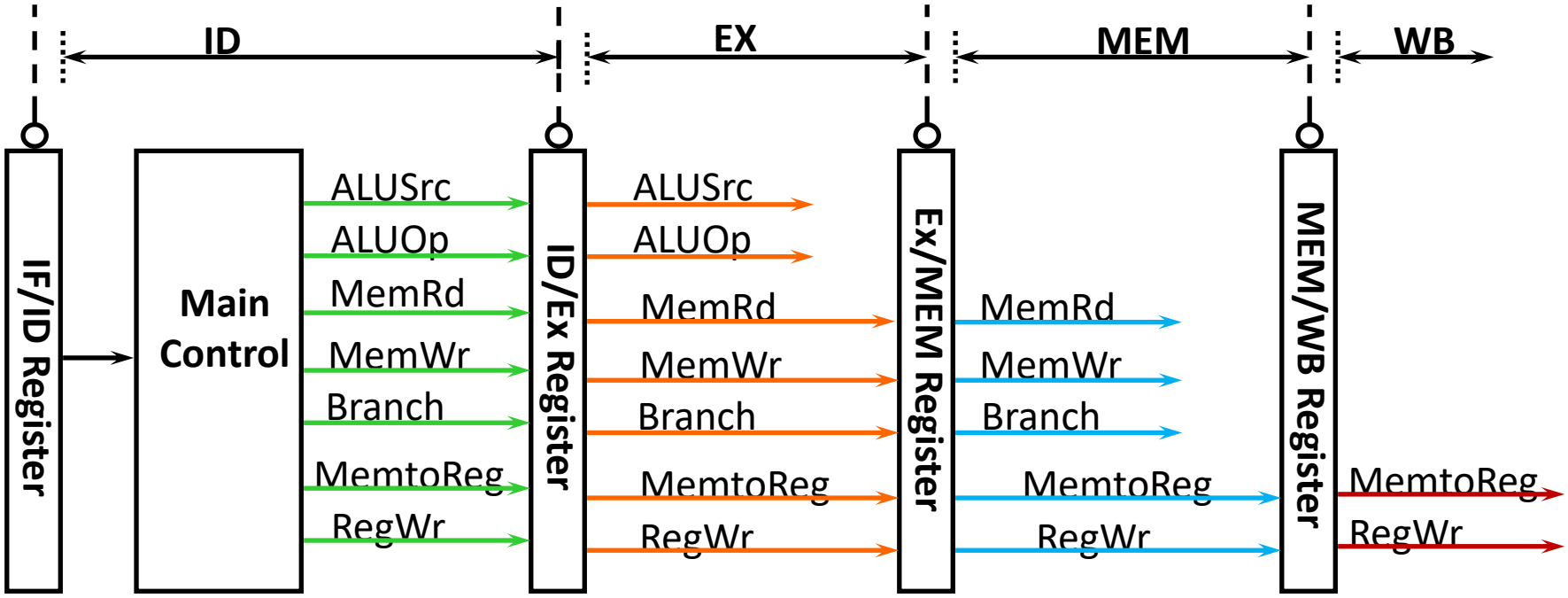


Fig. 4.48

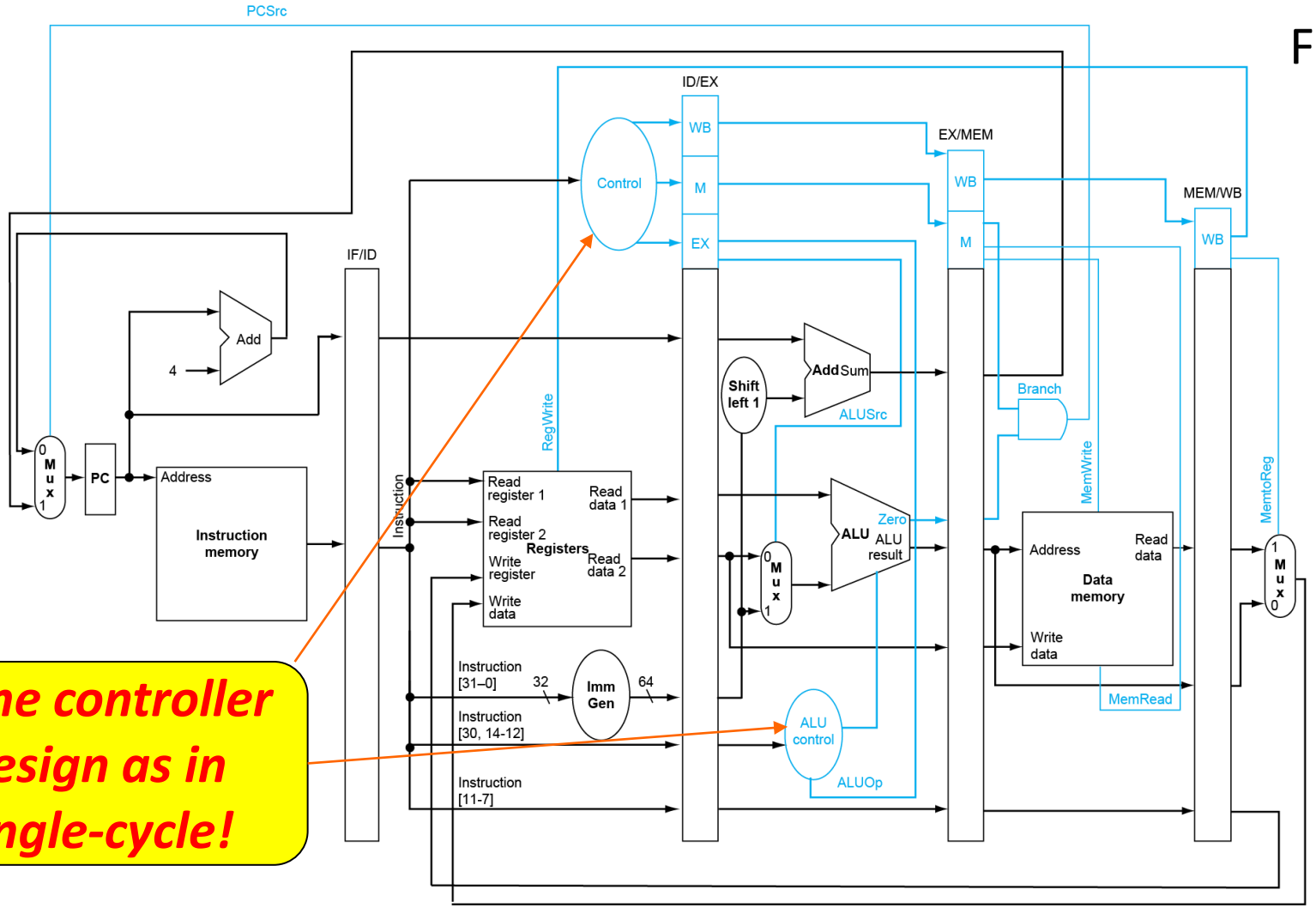
Pipelined Control

- Signals for EX (ALUSrc, ALUOp) are used 1 cycle later
- Signals for MEM (MemRd, MemWr, Branch) are used 2 cycles later
- Signals for WB (MemtoReg, RegWr) are used 3 cycles later



Datapath with Control

Fig. 4.49



Same controller design as in single-cycle!

Summary of Pipeline Basics

- Pipelining is a fundamental concept
  - Multiple steps using distinct resources
  - Utilize capabilities of datapath by pipelined instruction processing
    - Start next instruction while working on the current one
    - Limited by length of longest stage (plus fill/flush)
- CPI = ?
- What makes it easy in RISC-V?
  - All instructions are of the same length
  - Just a few instruction formats
  - Memory operands only in loads and stores
- What makes pipelining hard? Hazards!