CS / EE 320
Computer Organization and Assembly Language
Spring 2025
Lecture 17

Shahid Masud

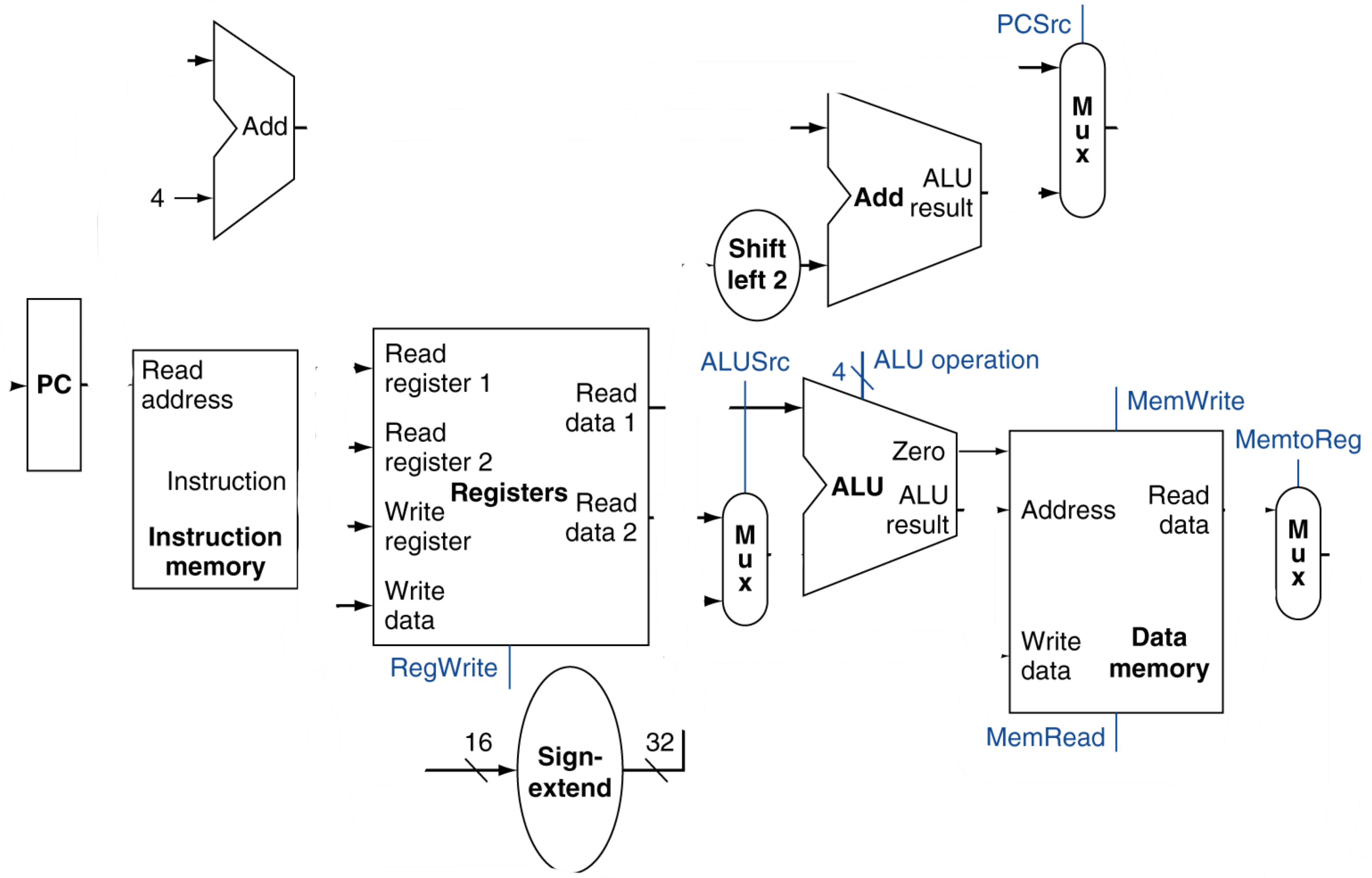
Topics: From Single Cycle MIPS to Multicyle MIPS, Pipelining

Topics

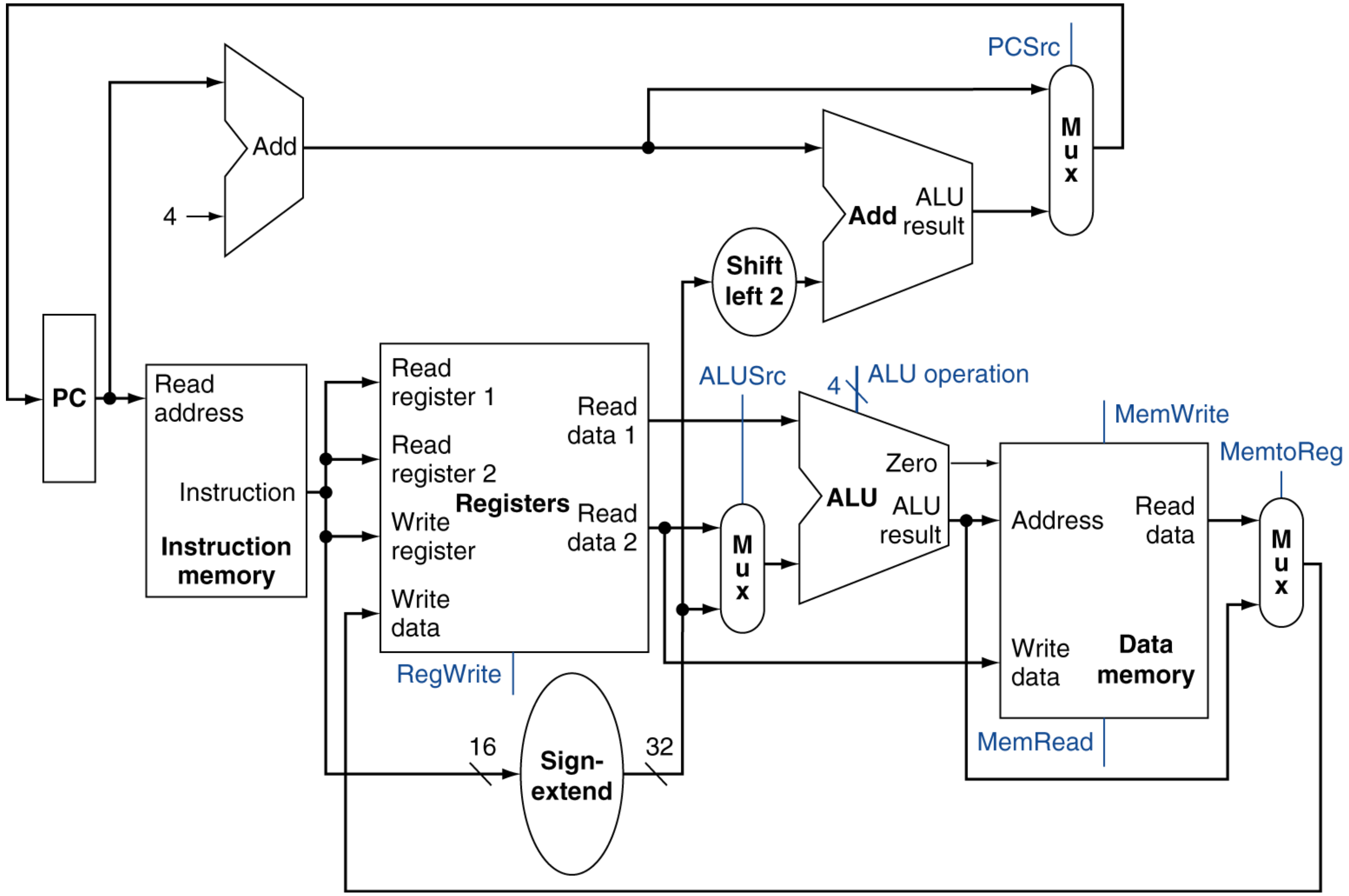
- Timing calculations in Single-Cycle Simple MIPS CPU
- How to add pipelining in simple MIPS CPU
- Detailed analysis of 5-Stage Pipelined MIPS CPU
- Calculations Throughput, Latency, Critical Path of 5-Stage MIPS CPU
- Design of individual Pipelined stages in a 5-Stage MIPS CPU
- Hazards in Pipelined MIPS CIPS
  - Structural Hazard (Hardware Related)
  - Data Hazard (Due to Assembly program)
  - Control Hazard (Architecture related)

SIMPLE MIPS WITH NO PIPELINE

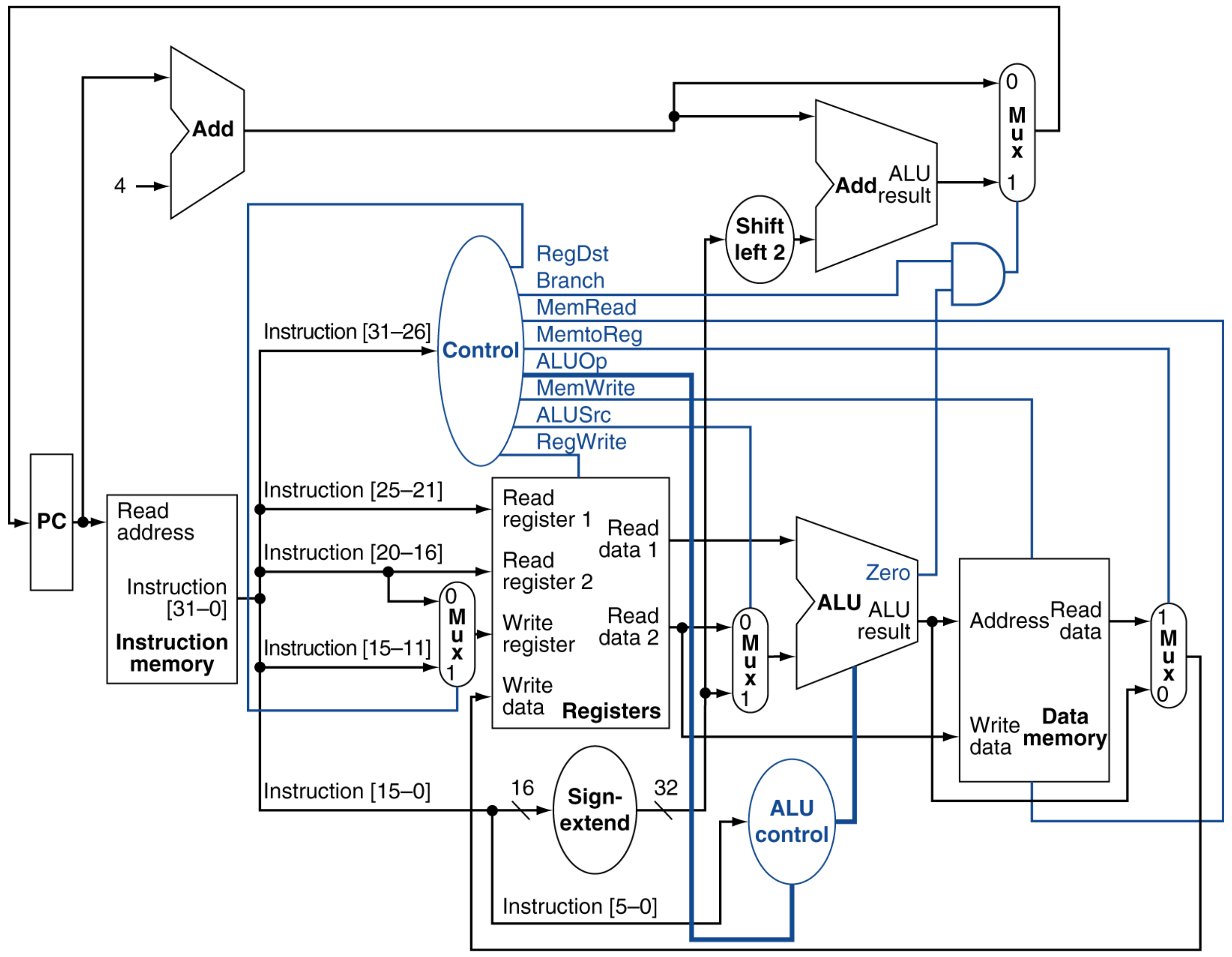
Full Datapath – for practice of connections



Full Datapath



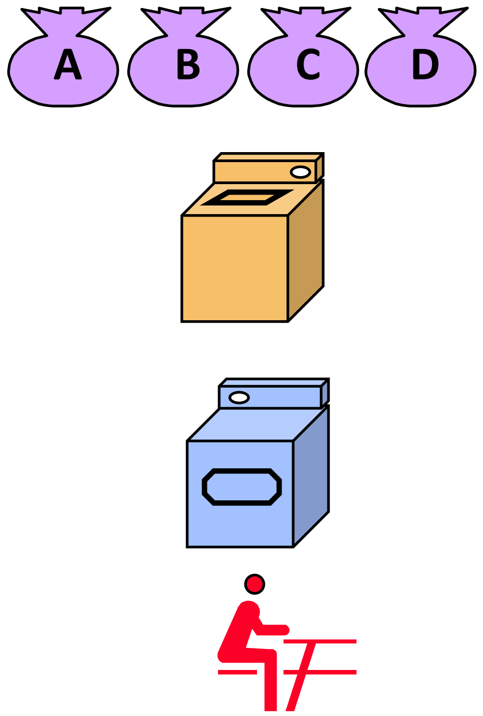
MIPS Datapath With Control



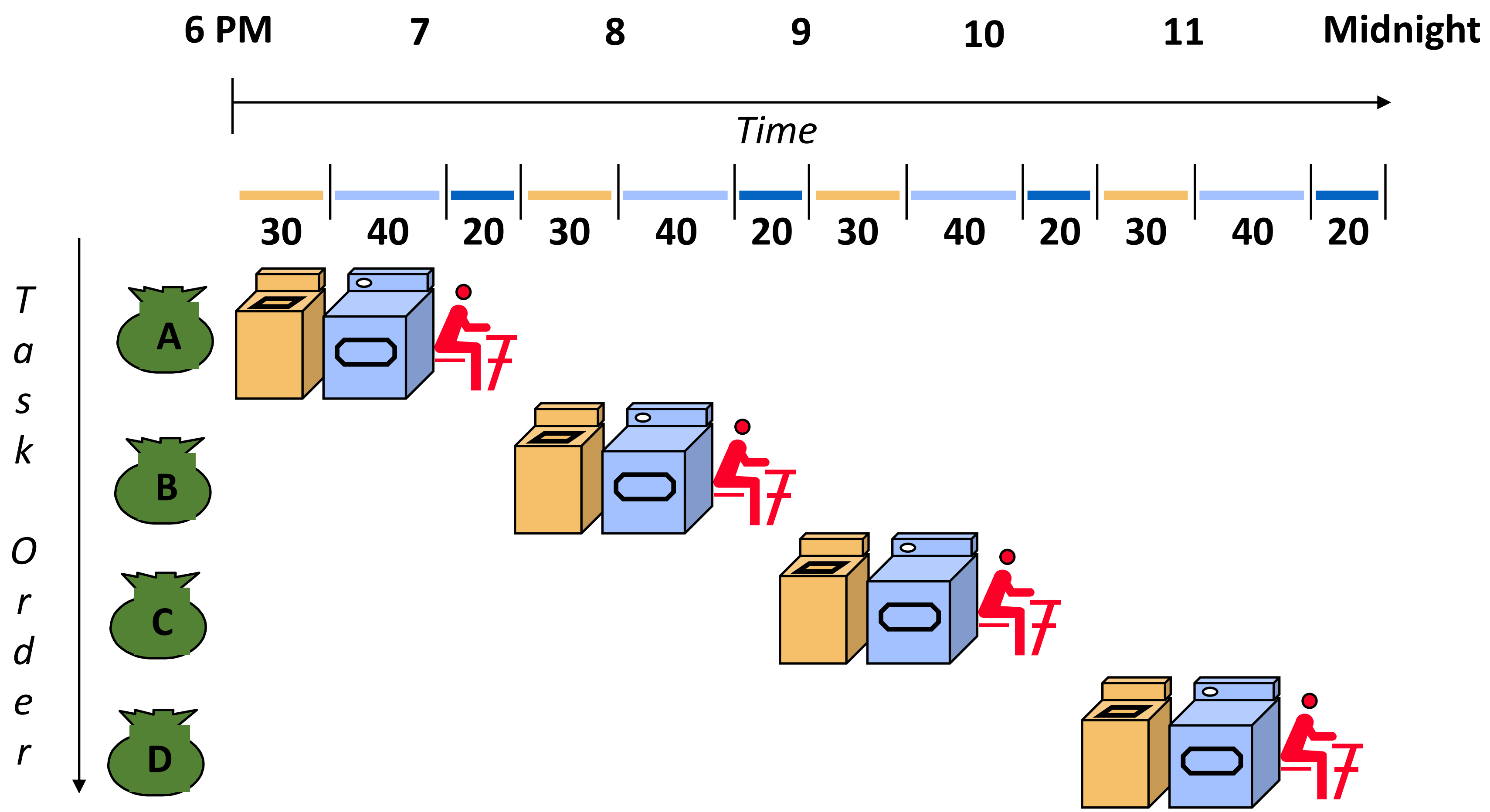
BASICS OF PIPELINING

What Is Pipelining?

- 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
- "Folding" takes 20 minutes
- One load takes 90 minutes



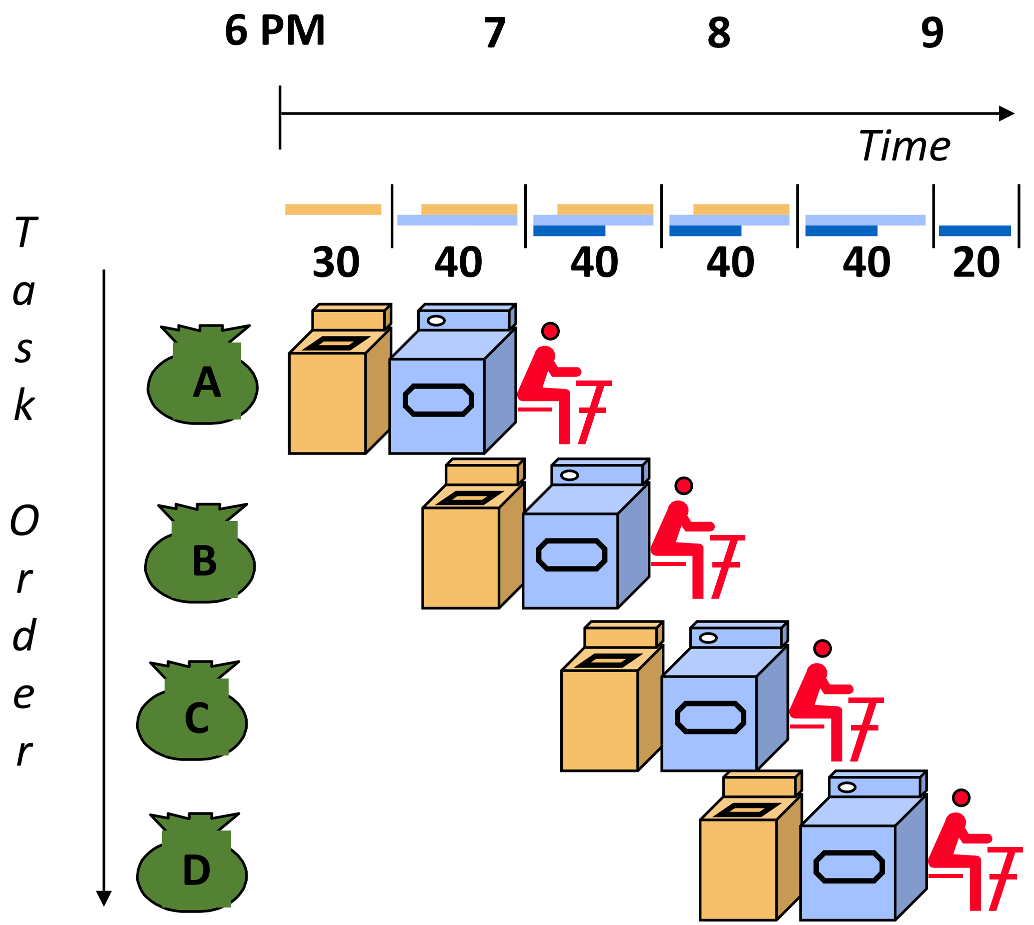
What Is Pipelining? ...contd



Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

Overlapping Different Operations



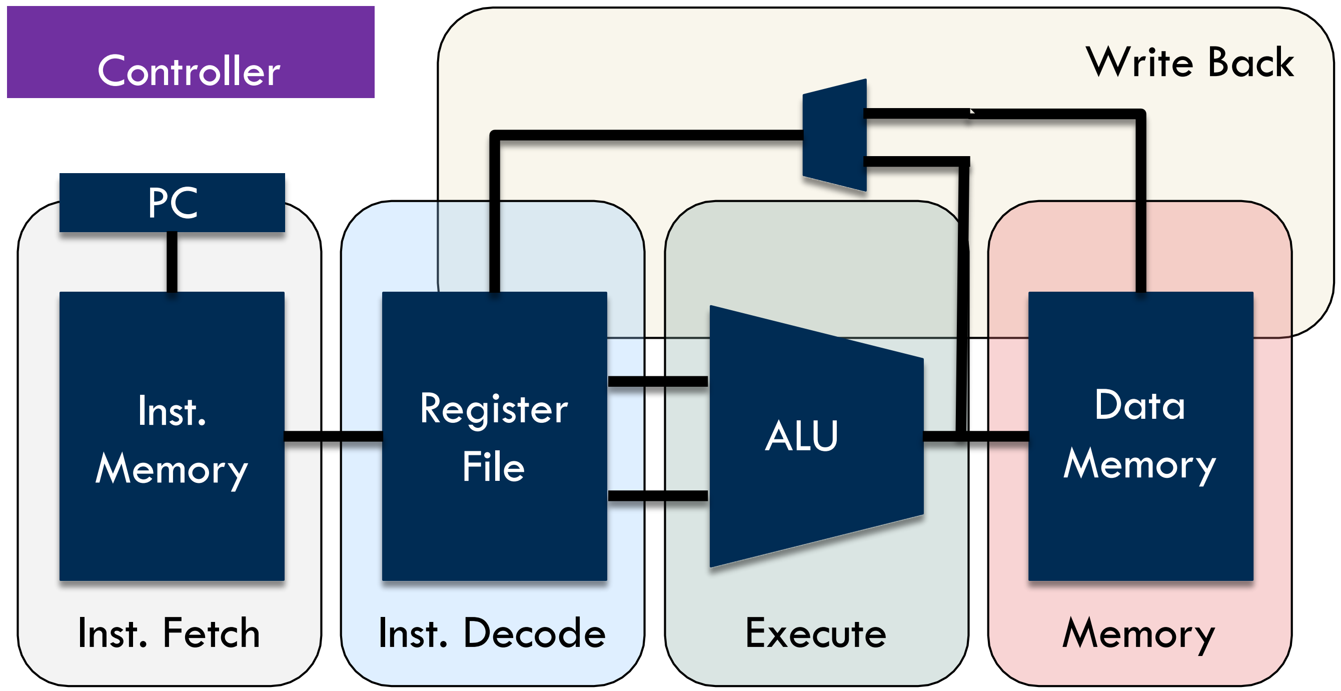
- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipeline stages
- Unbalanced lengths of pipeline stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup

Introducing Pipelining for MIPS

Single-cycle MIPS RISC Architecture

Example: simple MIPS architecture

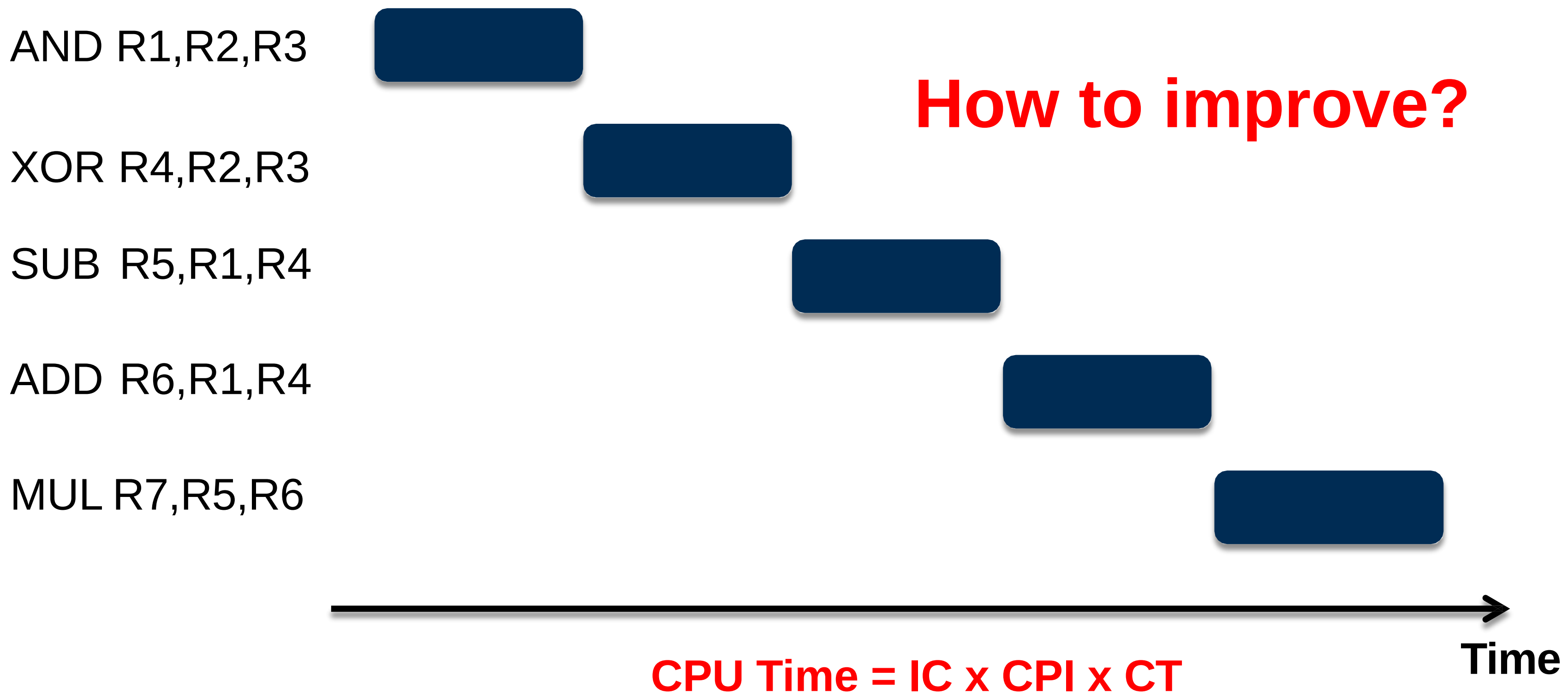
Critical path includes all of the processing steps



Execution on Single-cycle MIPS Architecture

Example program

CT=6ns; CPU Time = 5 x 1 x 6ns = 30ns


```
AND R1,R2,R3
XOR R4,R2,R3
SUB  R5,R1,R4
ADD  R6,R1,R4
MUL R7,R5,R6
```

How to improve?

CPU Time = IC x CPI x CT

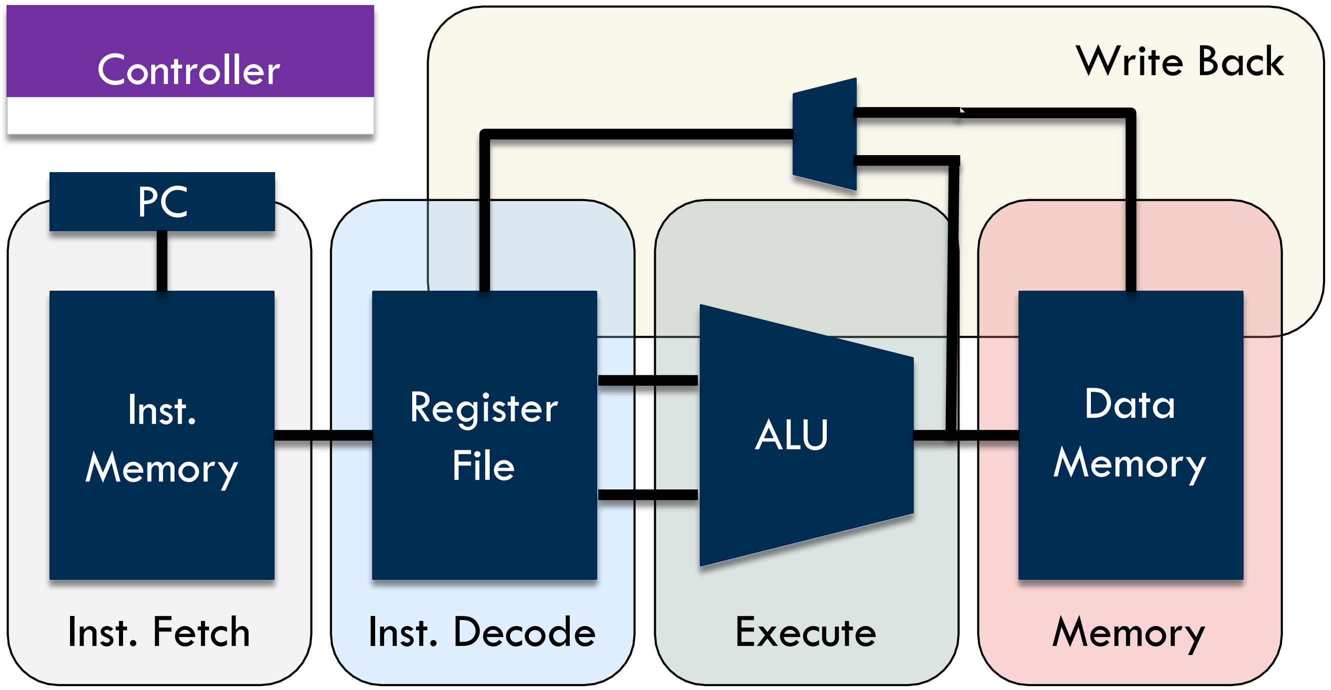
RISC Instruction Set

- Every instruction can be implemented in at most 5 clock cycles/ stages
  - Instruction fetch cycle (IF): send PC to memory, fetch the current instruction from memory, and update PC to the next sequential PC by adding 4 to the PC.
  - Instruction decode/register fetch cycle (ID): decode the instruction, read the registers corresponding to register source specifiers from the register file.
  - Execution/effective address cycle (EX): perform Memory address calculation for Load/Store, Register-Register ALU instruction and Register-Immediate ALU instruction.
  - Memory access (MEM): Perform memory access for load/store instructions.
  - Write-back cycle (WB): Write back results to the dest operands for Register-Register ALU instruction or Load instruction.

Re-Using Idle Resources

Each processing step finishes in a fraction of a cycle

- Idle resources can be reused for processing next instruction

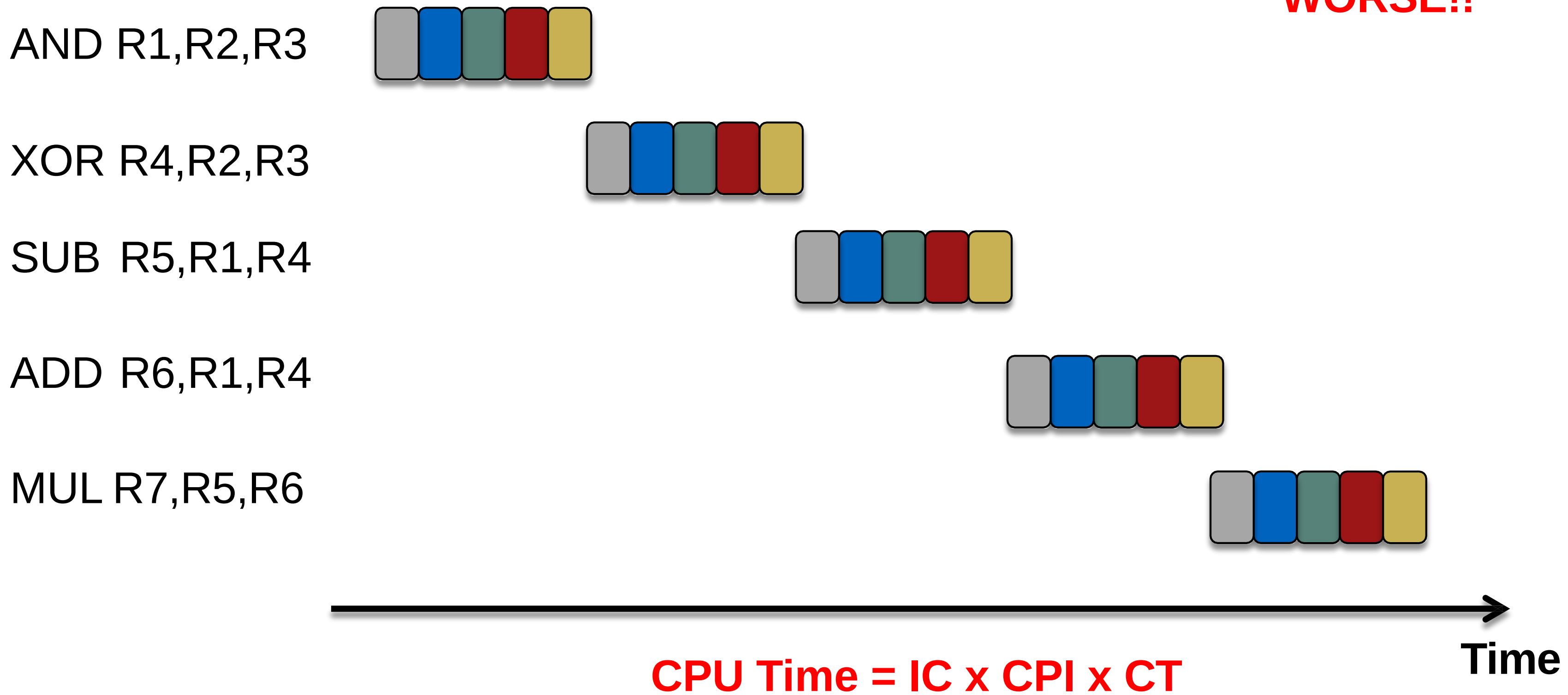


Non-Pipeline Architecture

Example program

CT=1.5ns; CPU Time = 5 x 5 x 1.5ns = 37.5ns > 30ns

WORSE!!



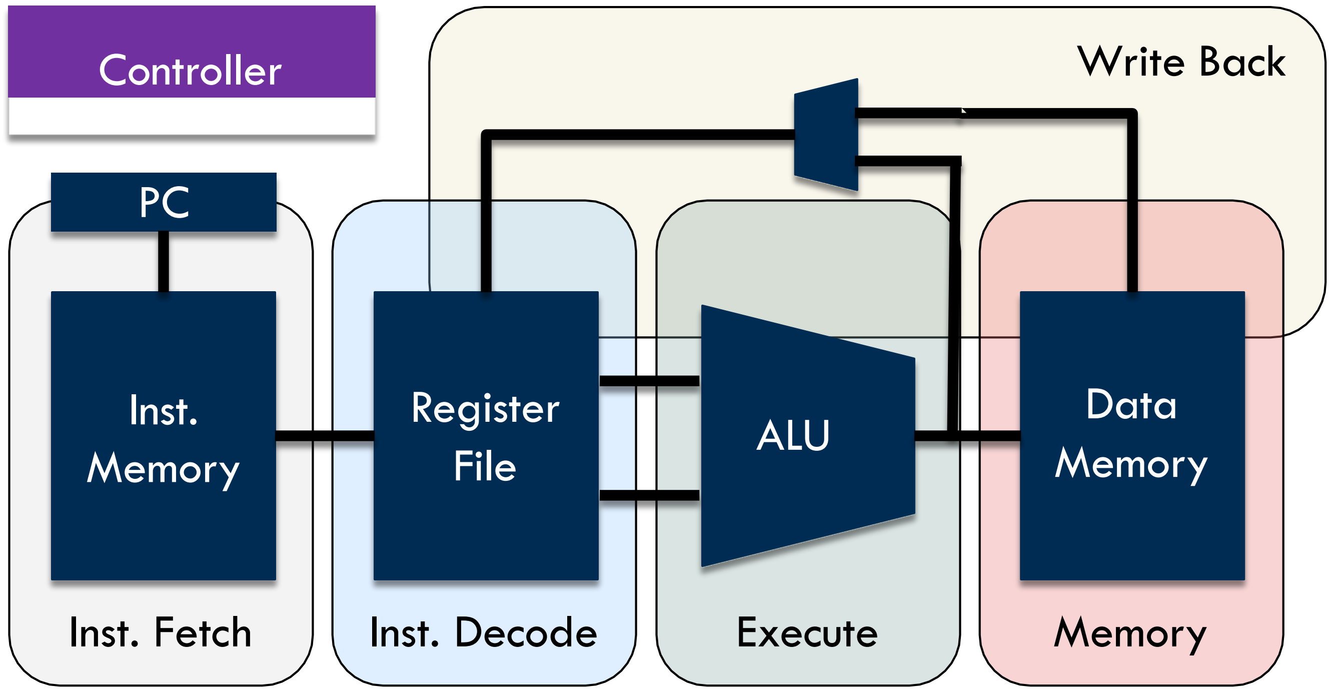
CPU Time = IC x CPI x CT

Developing a Multi-cycle Data Path

5

Develop Simple Five-Stage MIPS Pipeline

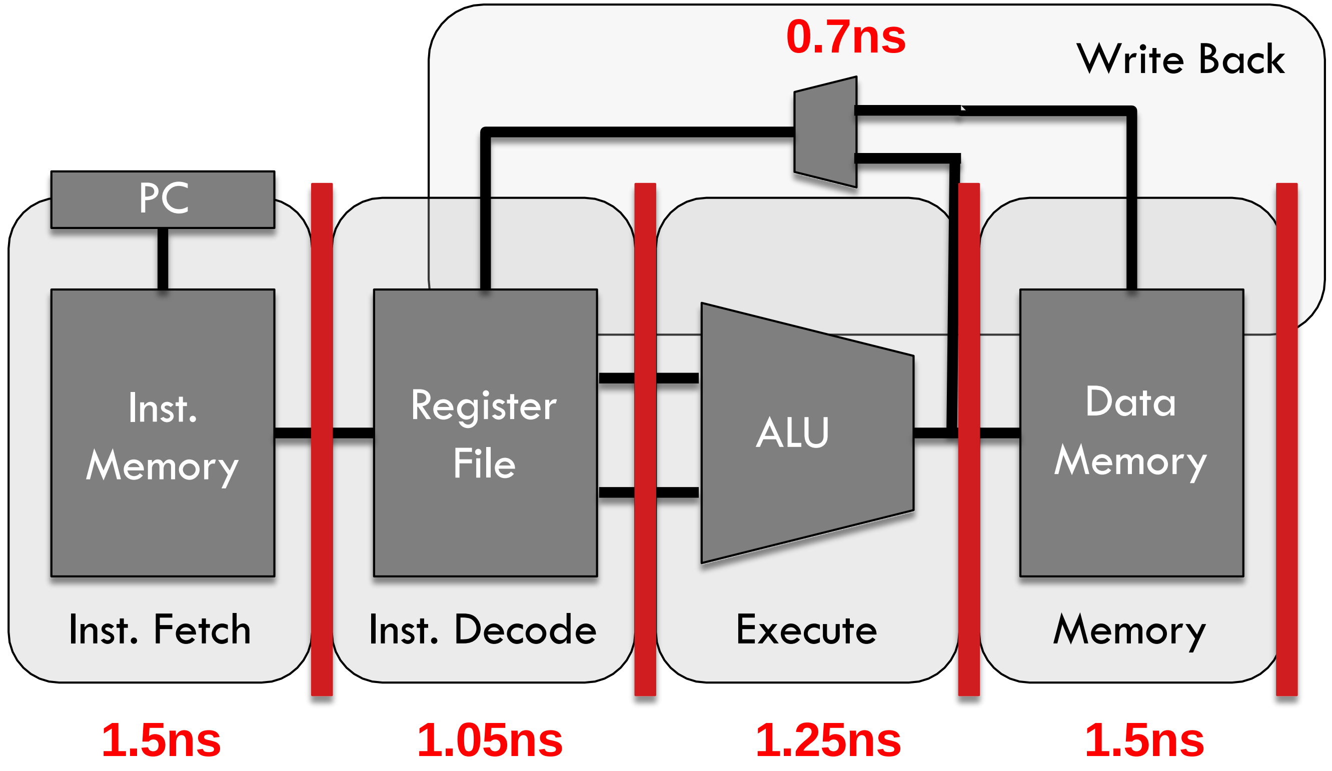
A pipelined load-store architecture that processes up to one instruction per cycle



Identify Distinct Processing Stages in Architecture

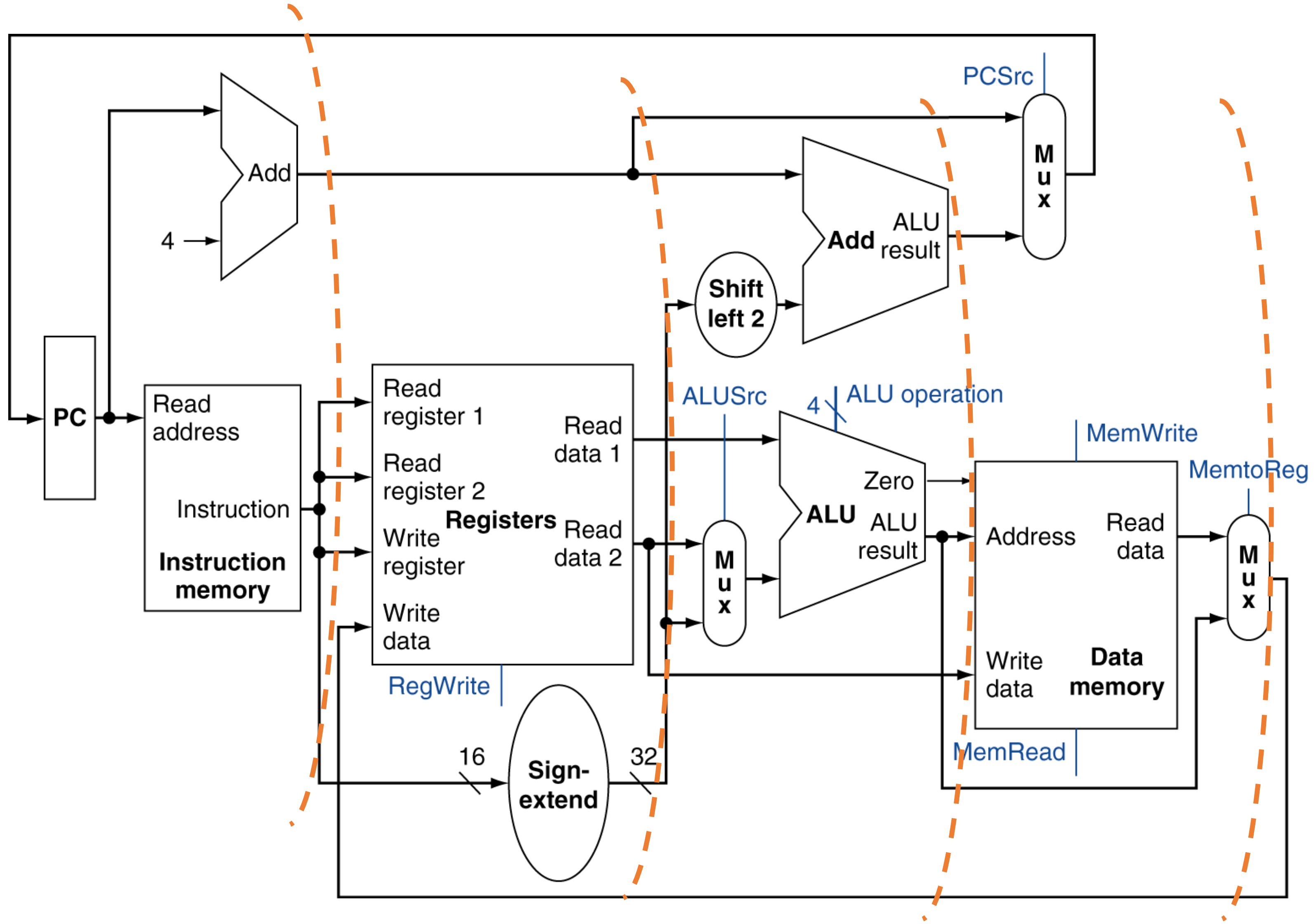
Five stage pipeline

Critical path determines the cycle time



Identify CPU Operations where Pipeline Registers could be Inserted

Recap - Full Datapath

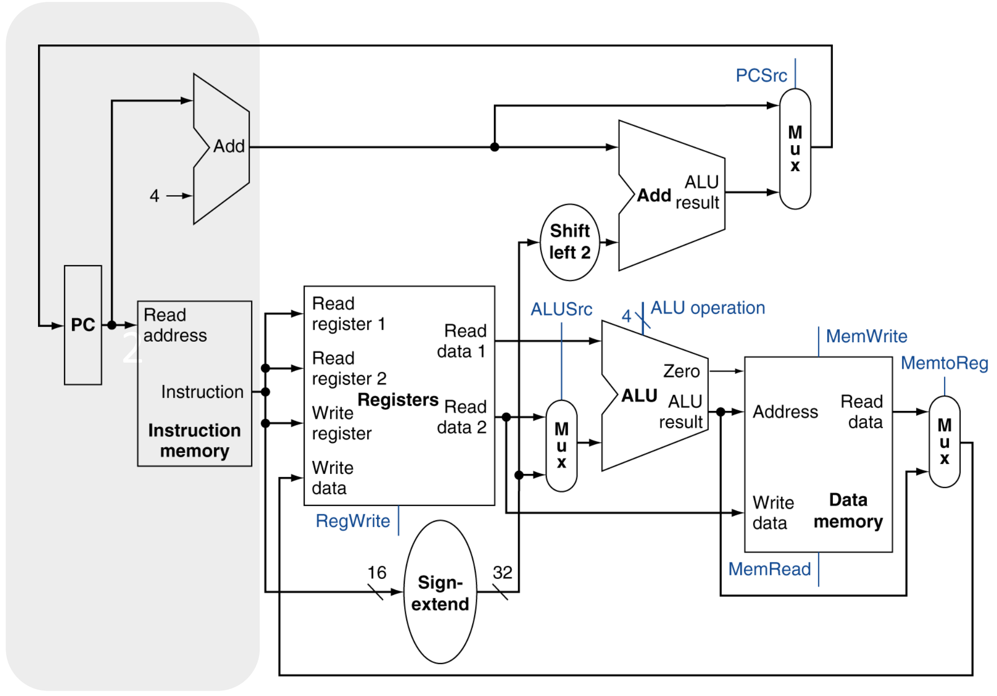


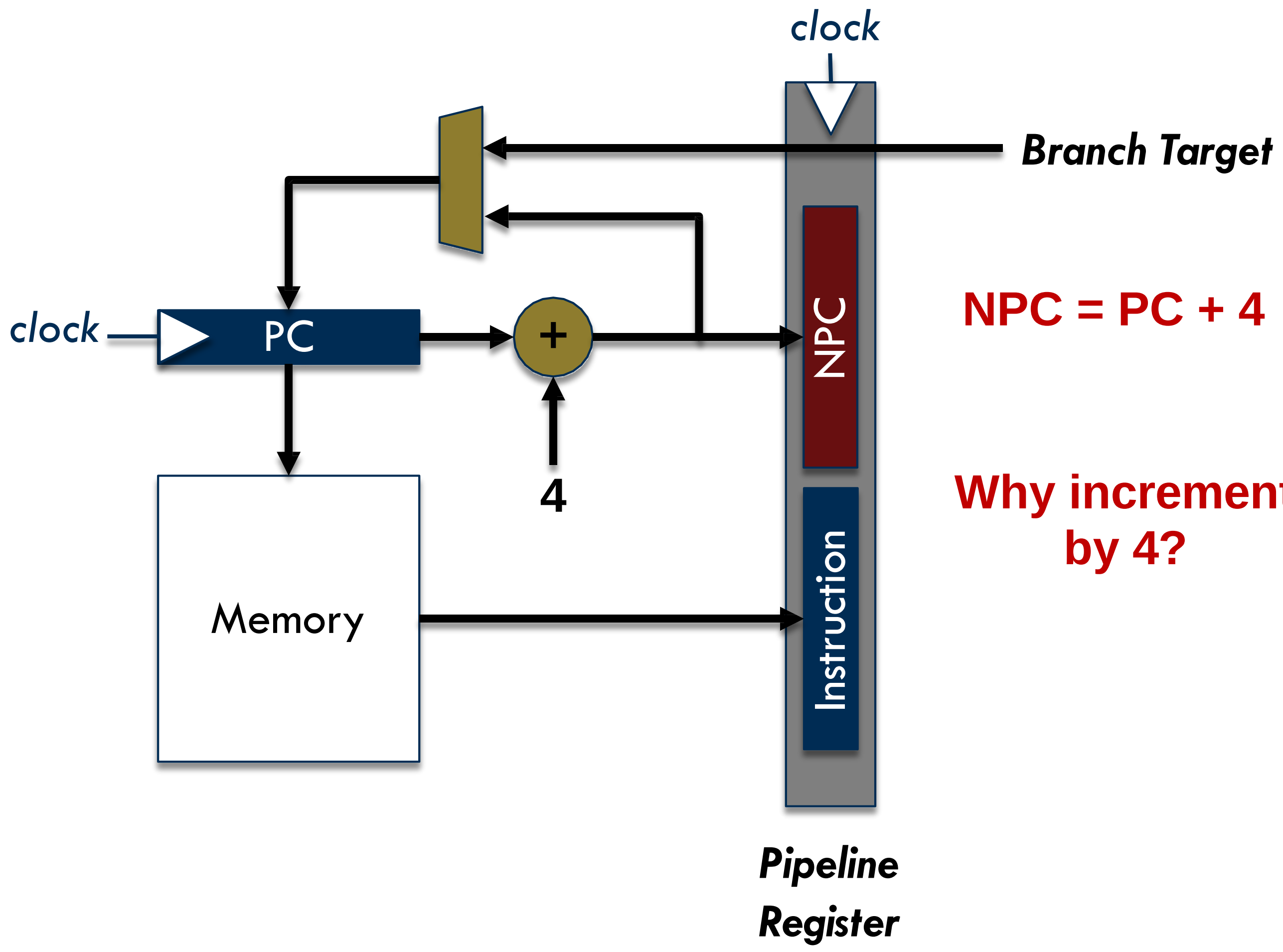
Instruction Fetch

- Read an instruction from memory (I-Memory)
  - Use the program counter (PC) to index into the I- Memory
  - Compute New PC (NPC) by incrementing current PC
    - What about branches?
      - (later?)

- Update pipeline registers
  - Write the instruction into the pipeline registers

Instruction Fetch Stage





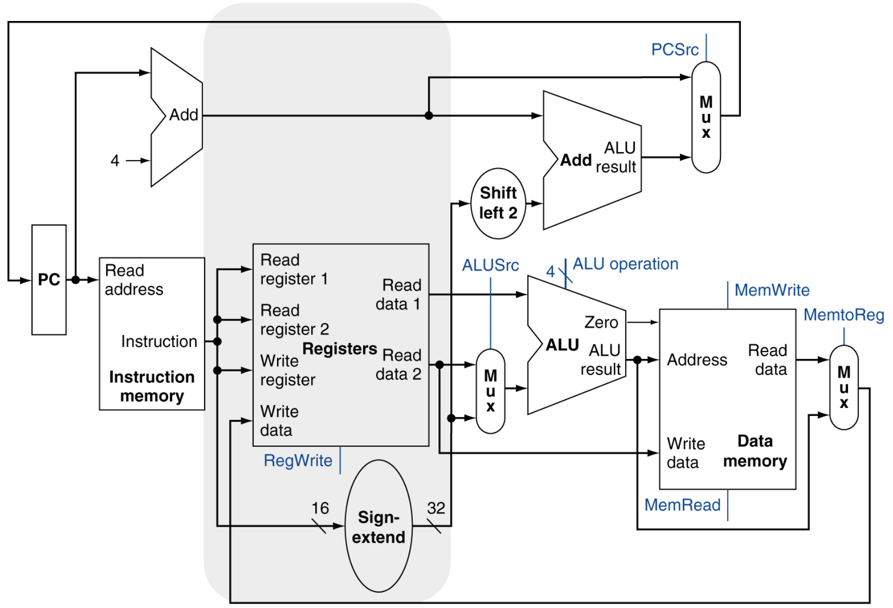
NPC = PC + 4

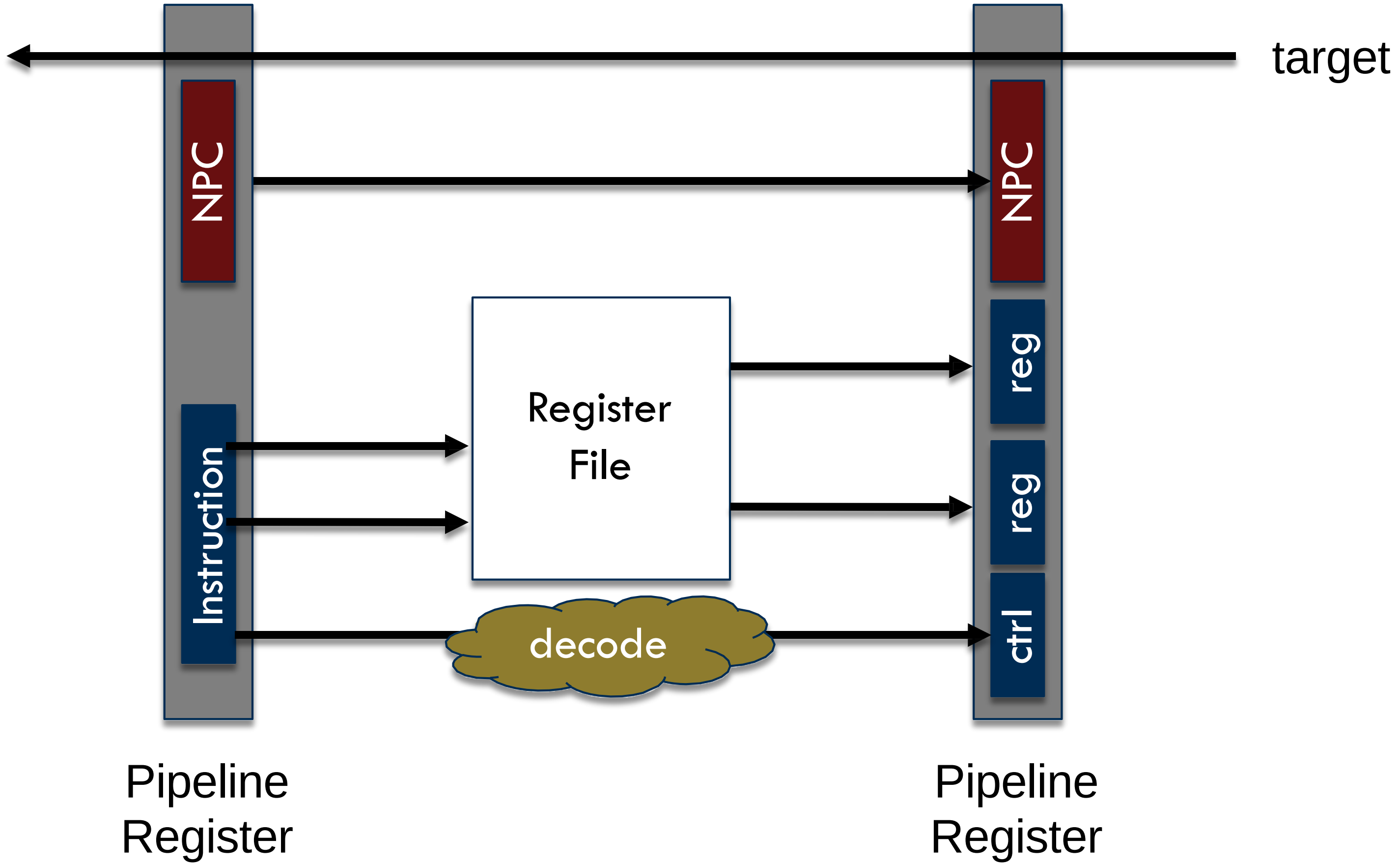
Why increment by 4?

Instruction Decode

- Generate control signals for the opcode bits
- Read source operands from the register file (RF)
  - Use the specifiers for indexing RF
    - How many read ports are required?
- Update pipeline registers
  - Send the operand and immediate values to next stage
  - Pass control signals and NPC to next stage

Instruction Decode Stage

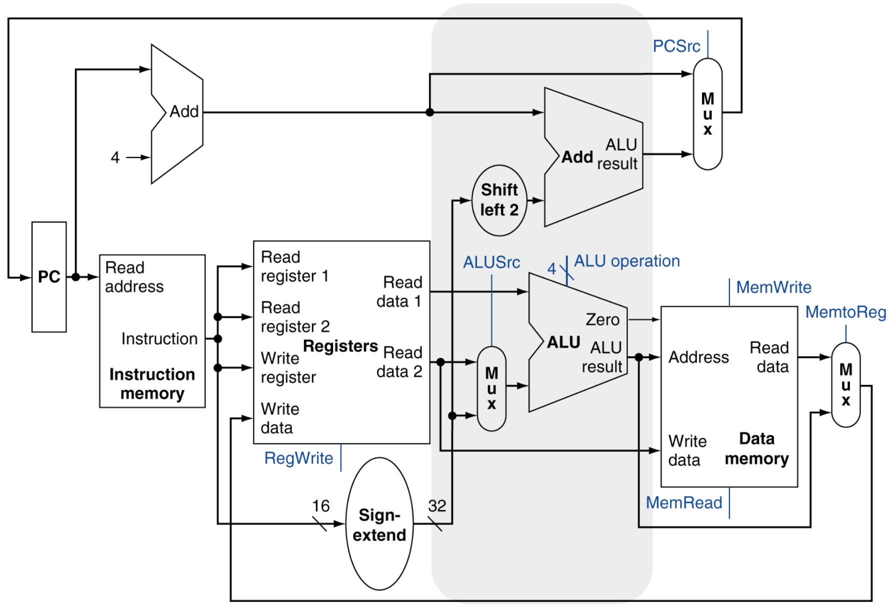


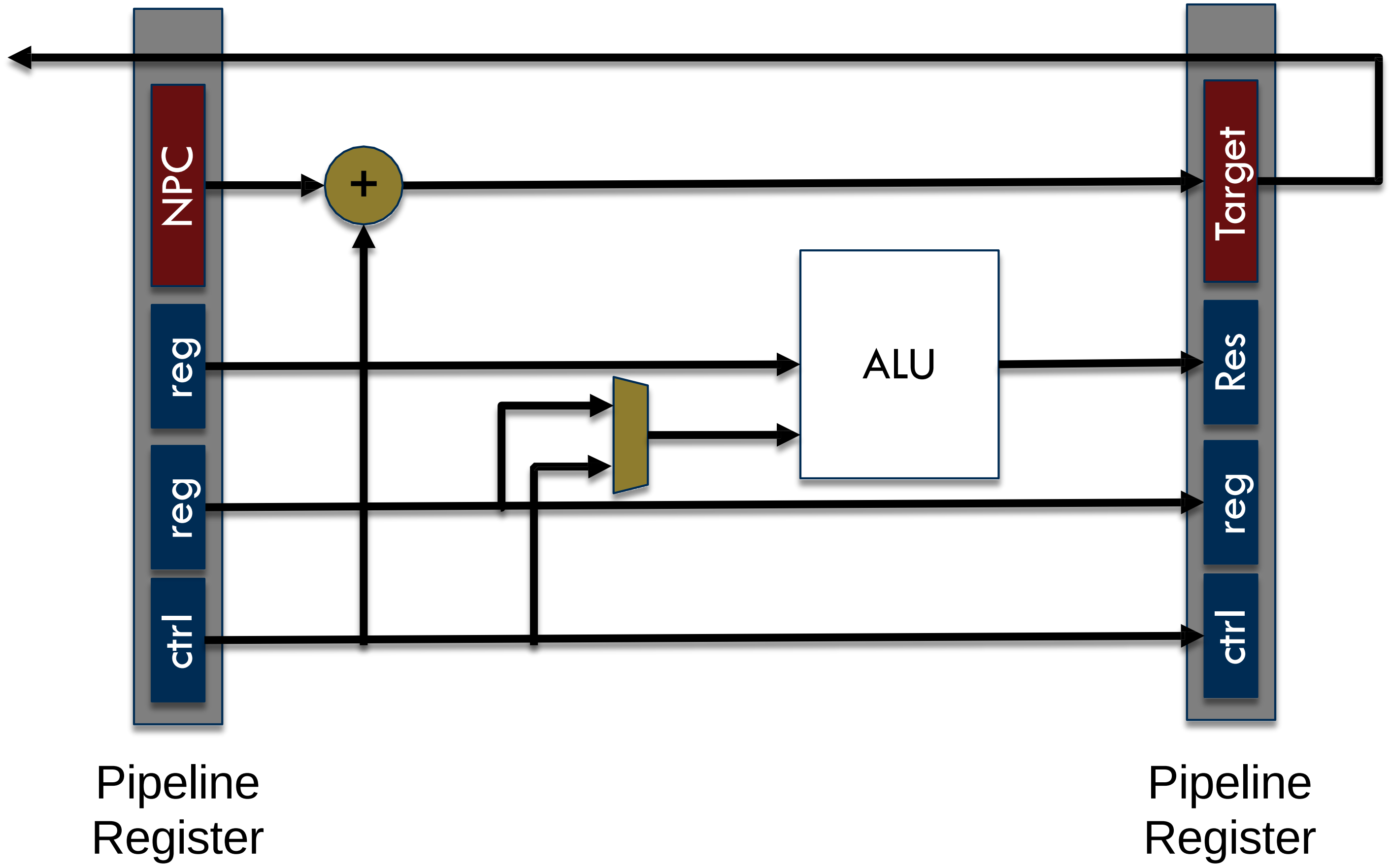


Instruction Execute

- Perform ALU operation
  - Compute the result of ALU
    - Operation type: control signals
    - First operand: contents of a register
    - Second operand: either a register or the immediate value
  - Compute branch target
    - Target = NPC + immediate
- Update pipeline registers
  - Control signals, branch target, ALU results, and destination

Instruction Execute Stage

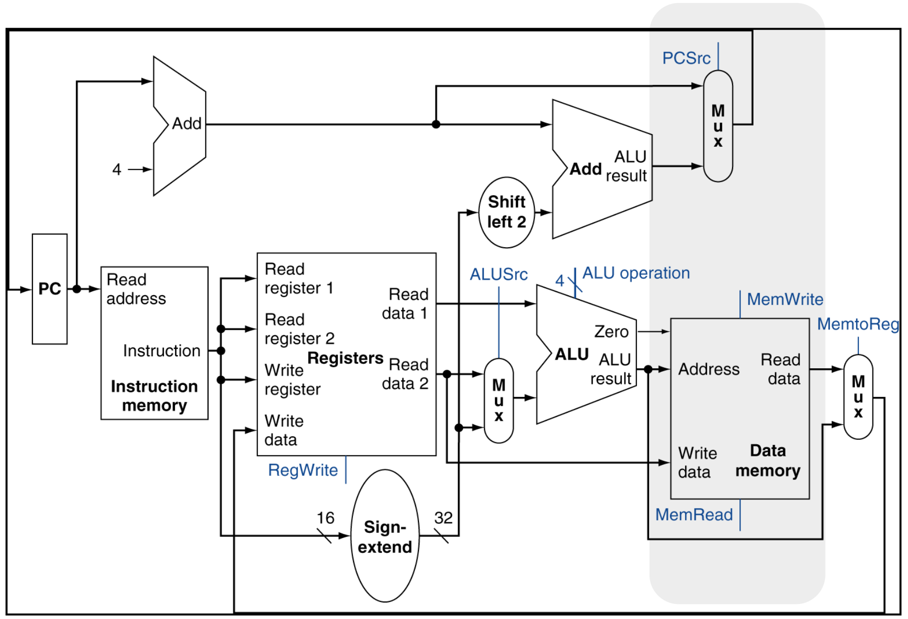


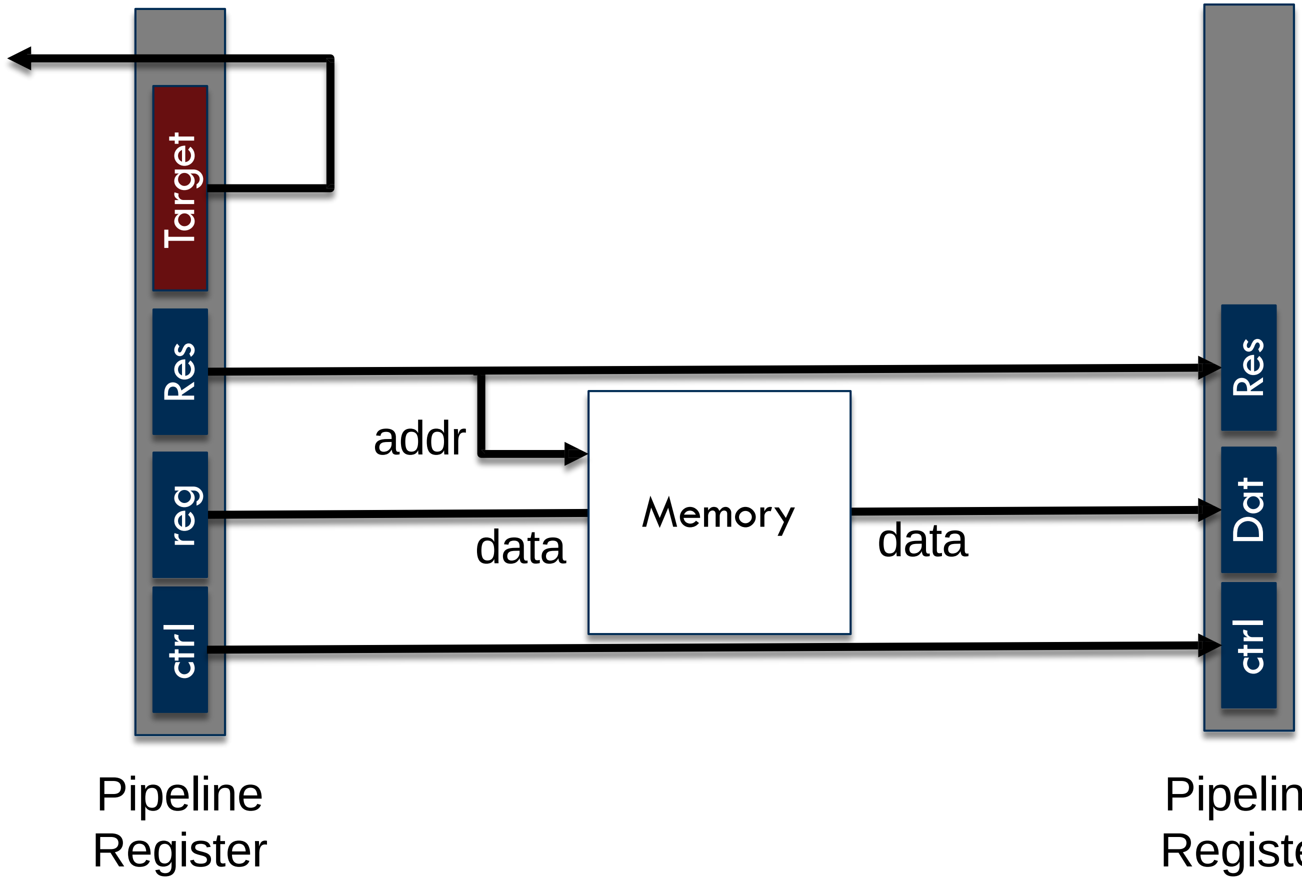


Memory Access

- Access data memory
  - Load/store address: ALU outcome
  - Control signals determine read or write access
- Update pipeline registers
  - ALU results from execute
  - Loaded data from D-Memory
  - Destination register

Memory Access Stage



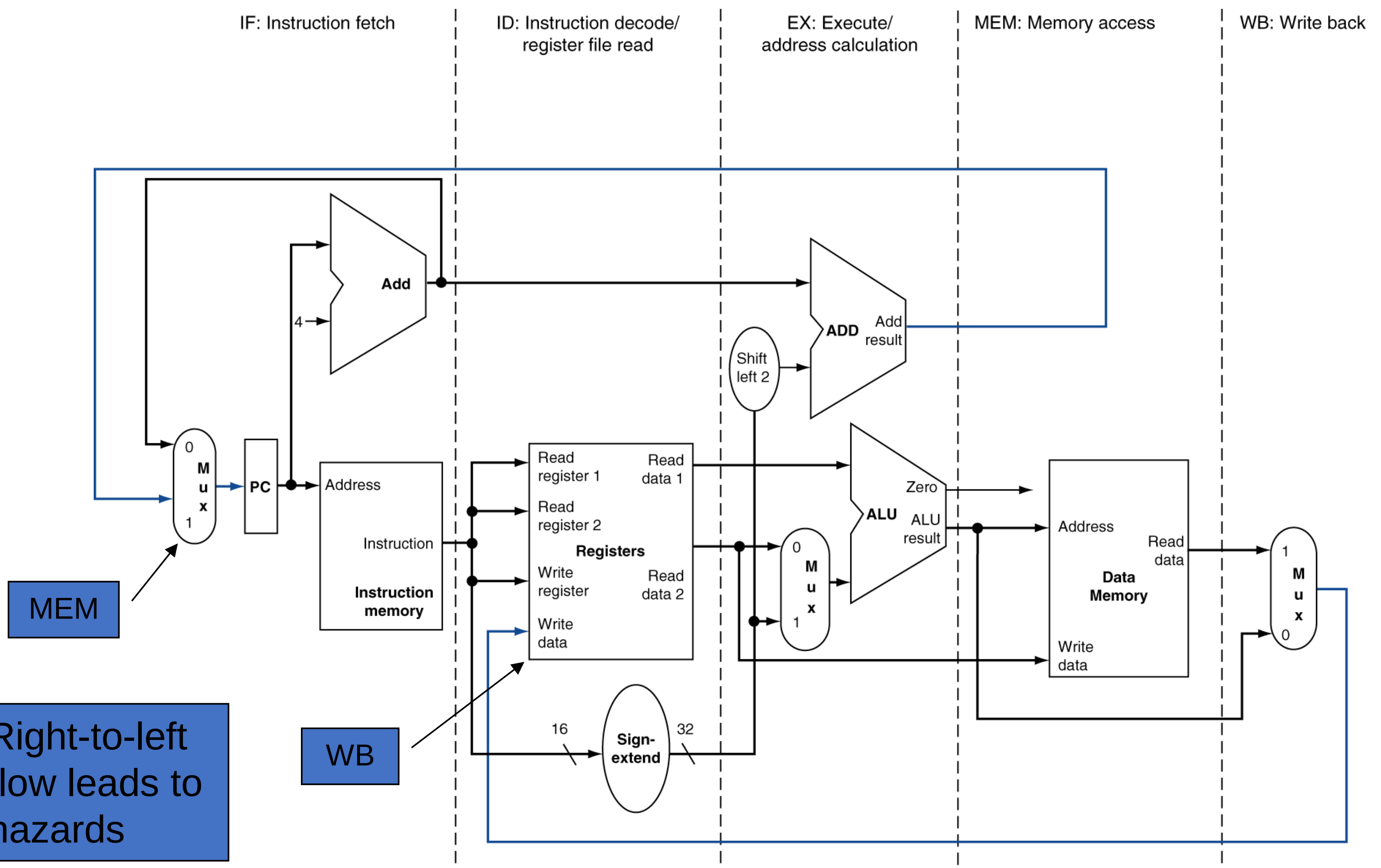


Register Write Back

Update register file

- Control signals determine if a register write is needed
- Only one write port is required
  - Write the ALU result to the destination register, or
  - Write the loaded data into the register file

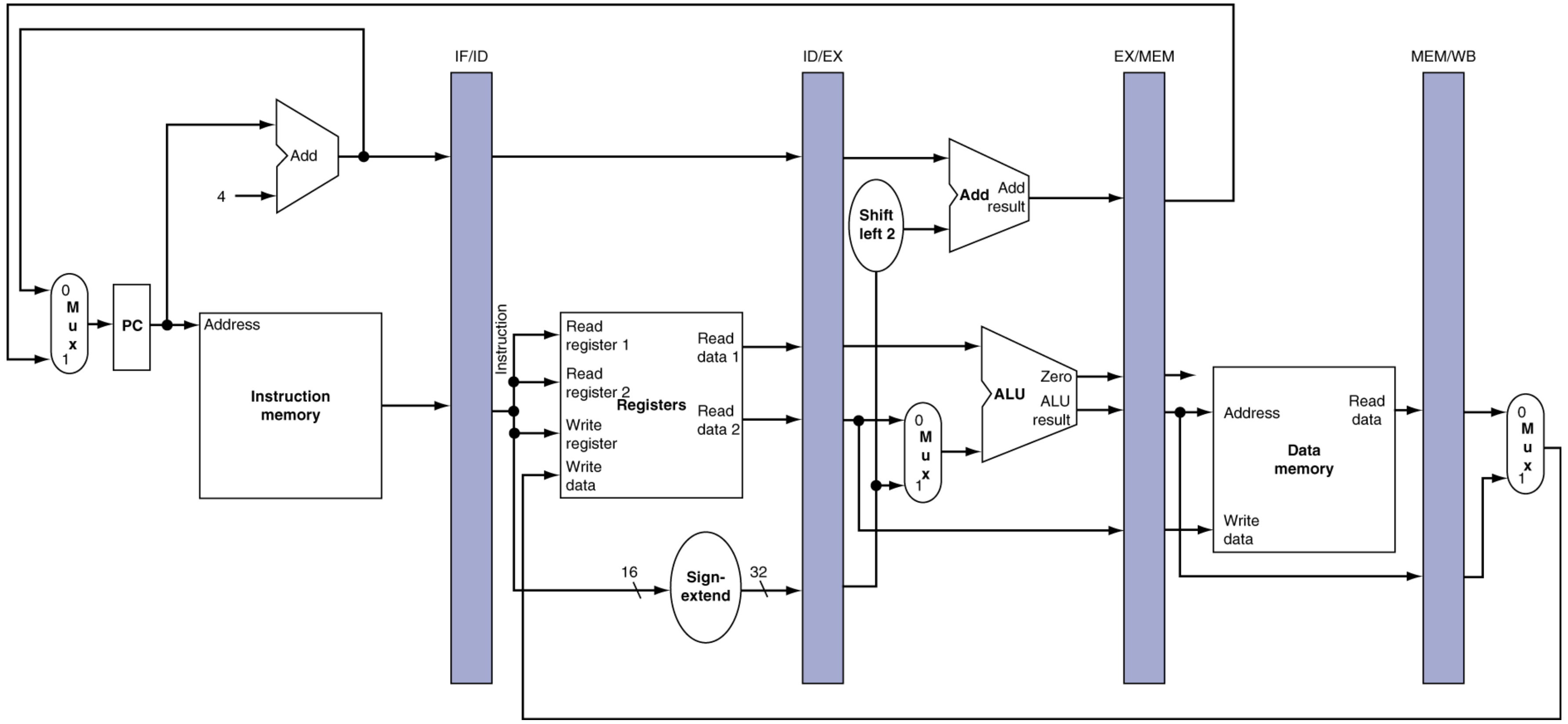
MIPS Pipelined Datapath



Right-to-left flow leads to hazards

Pipeline registers

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



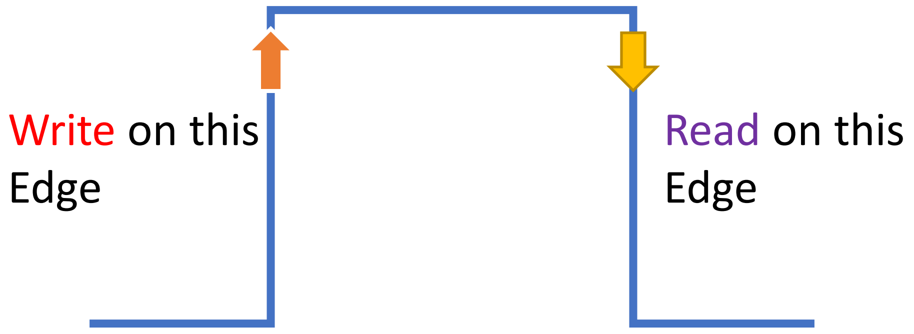
Pipeline Operation

- Cycle-by-cycle flow of instructions through the pipelined datapath
  - “Single-clock-cycle” pipeline diagram
    - Shows pipeline usage in a single cycle
    - Highlight resources used
  - c.f. “multi-clock-cycle” diagram
    - Graph of operation over time
- We’ll look at “single-clock-cycle” diagrams for load & store

RF is two operations in One Clock Cycle

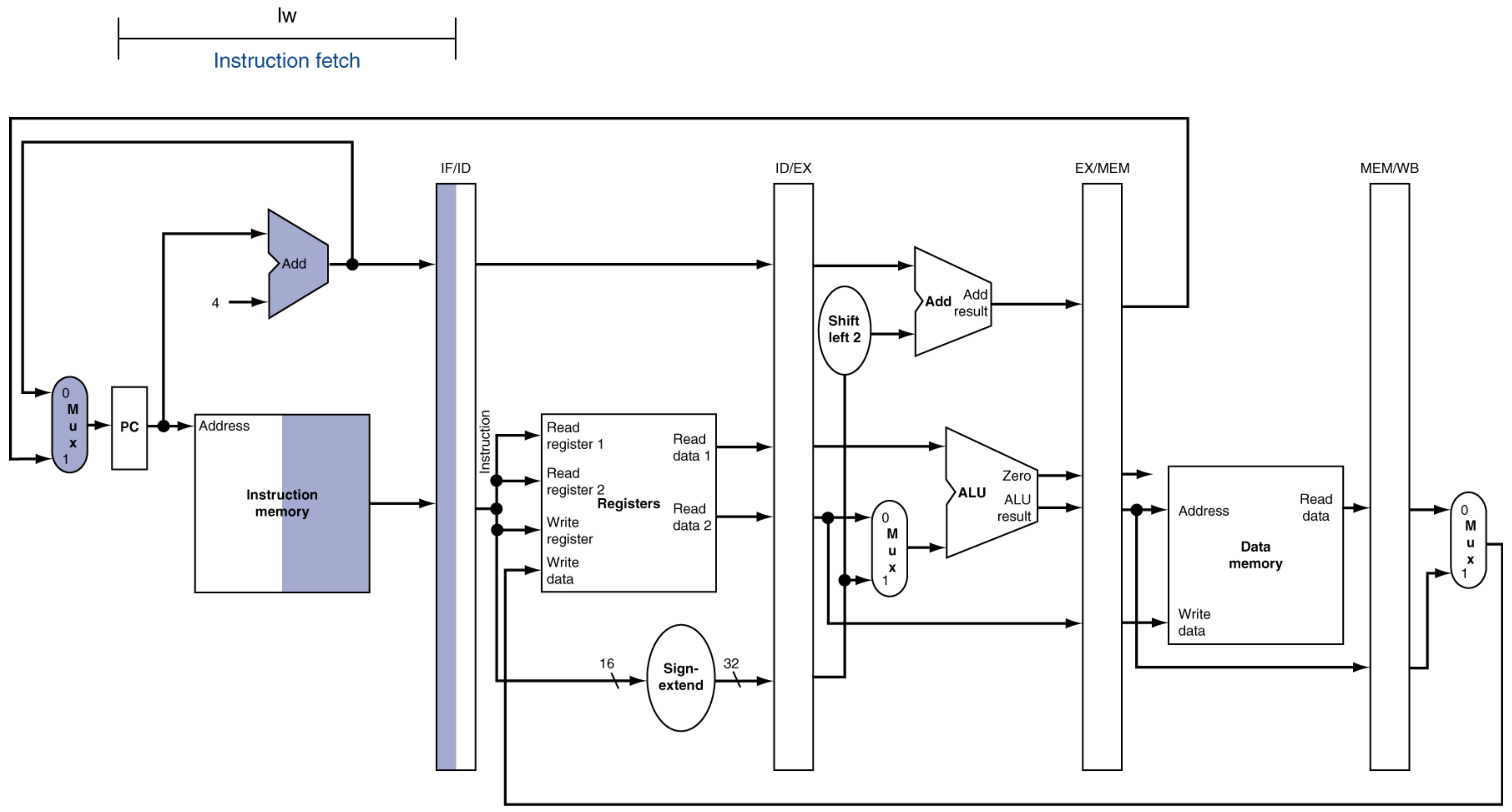
- Assume Write is when clock is going up
- Assume Read is when Clock is going down

(you can assume otherwise too, but two operations in one clock cycle)

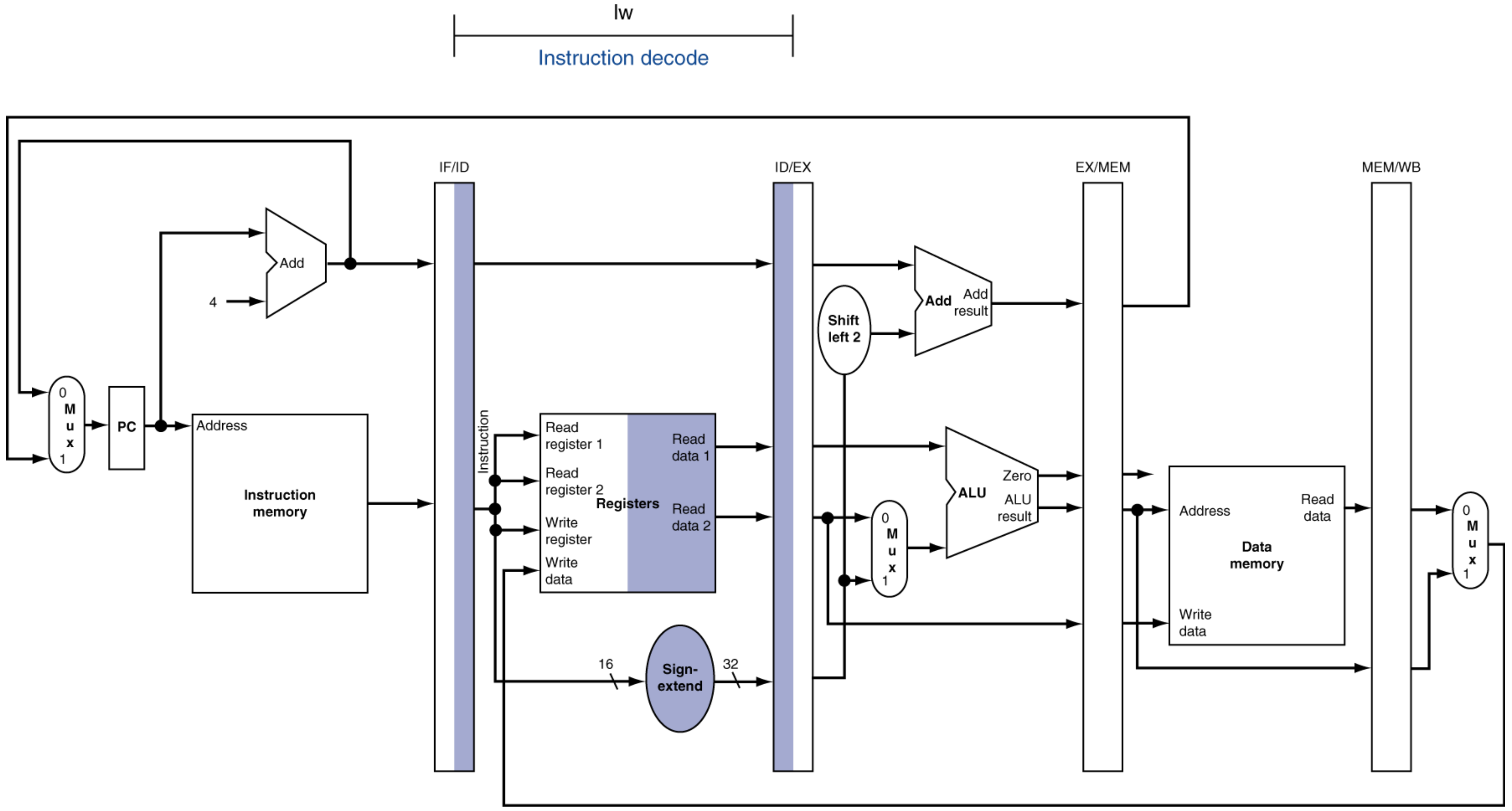


Clock Edges

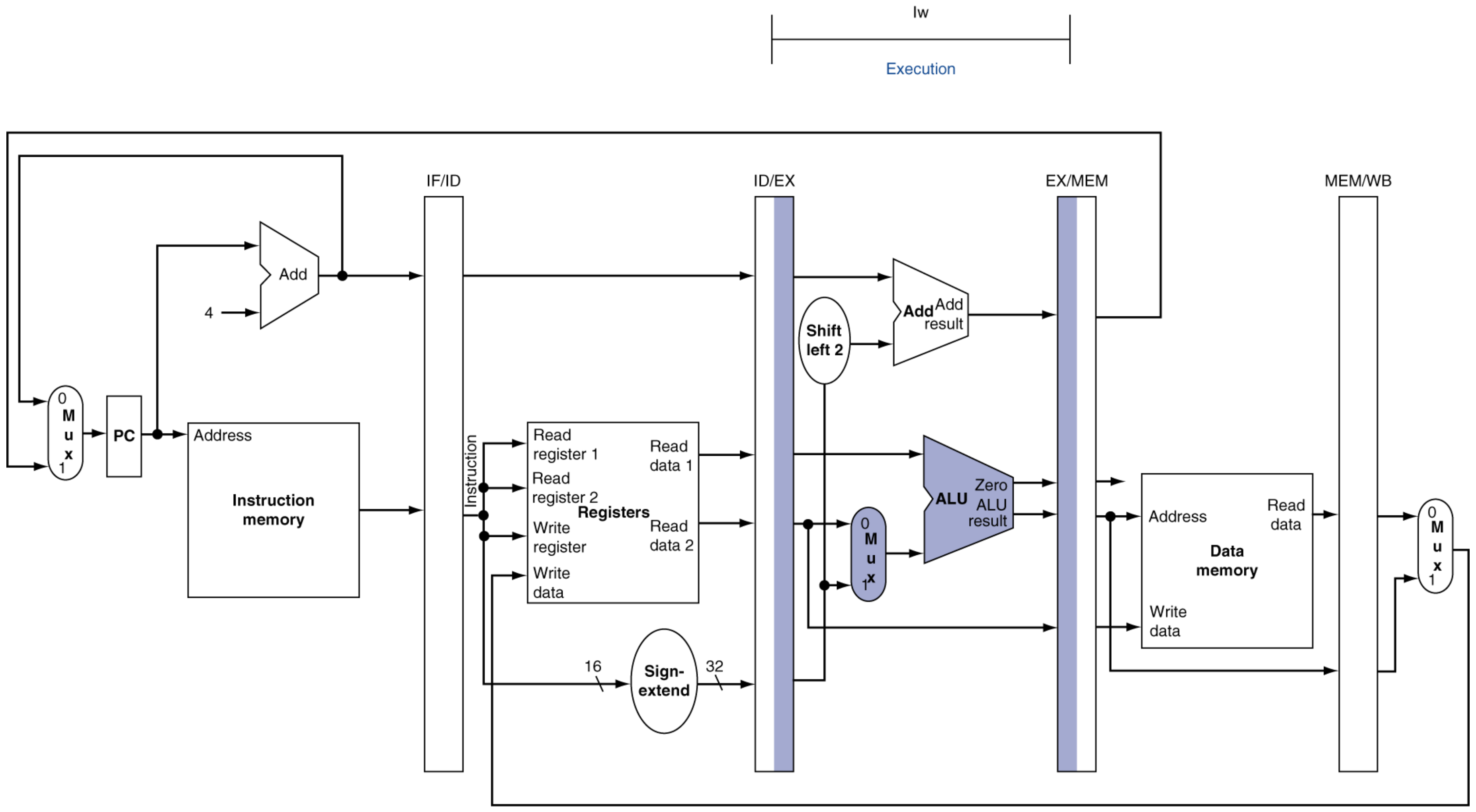
IF for Load, Store, …



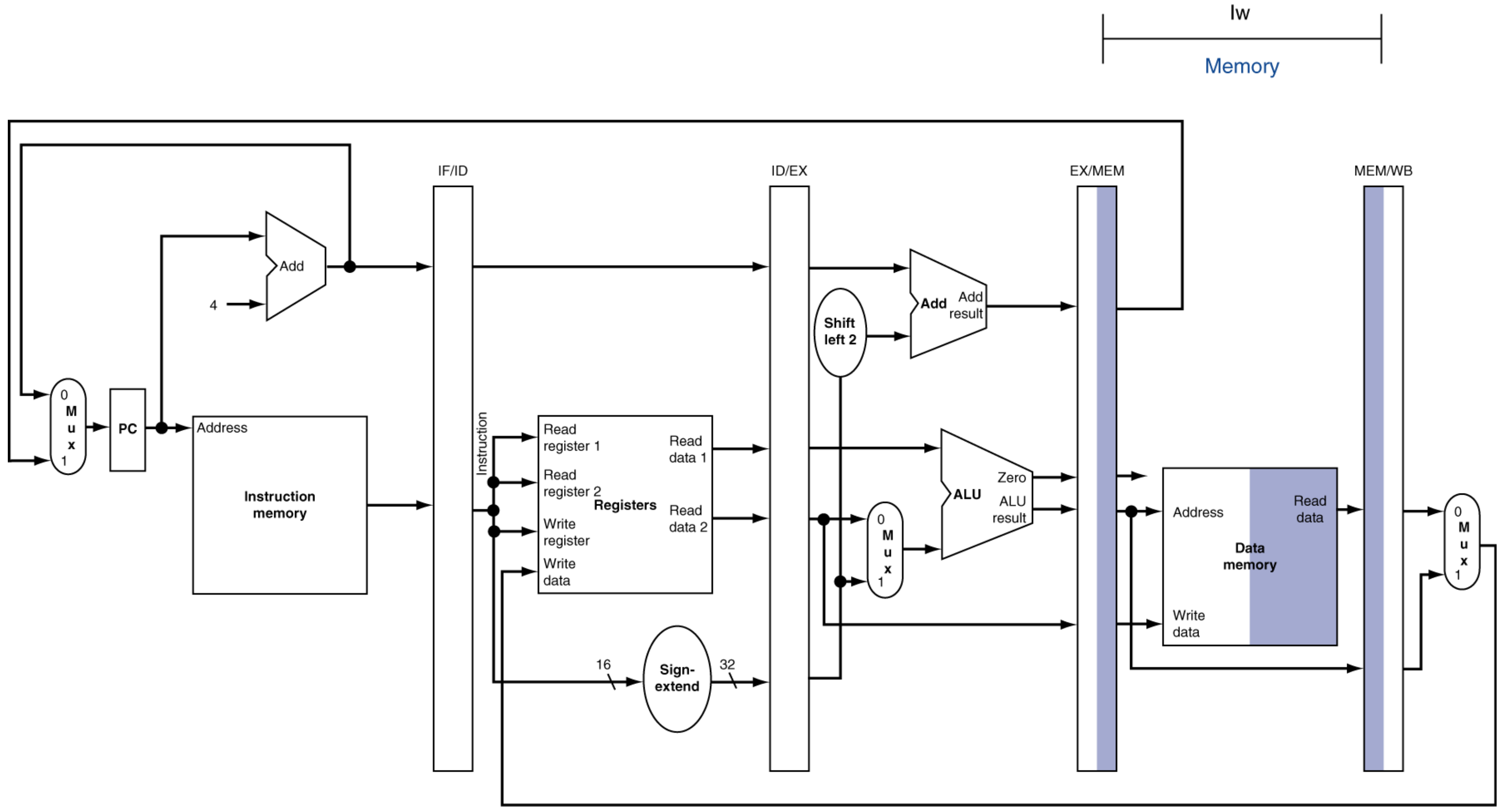
ID for Load, Store, …



EX for Load

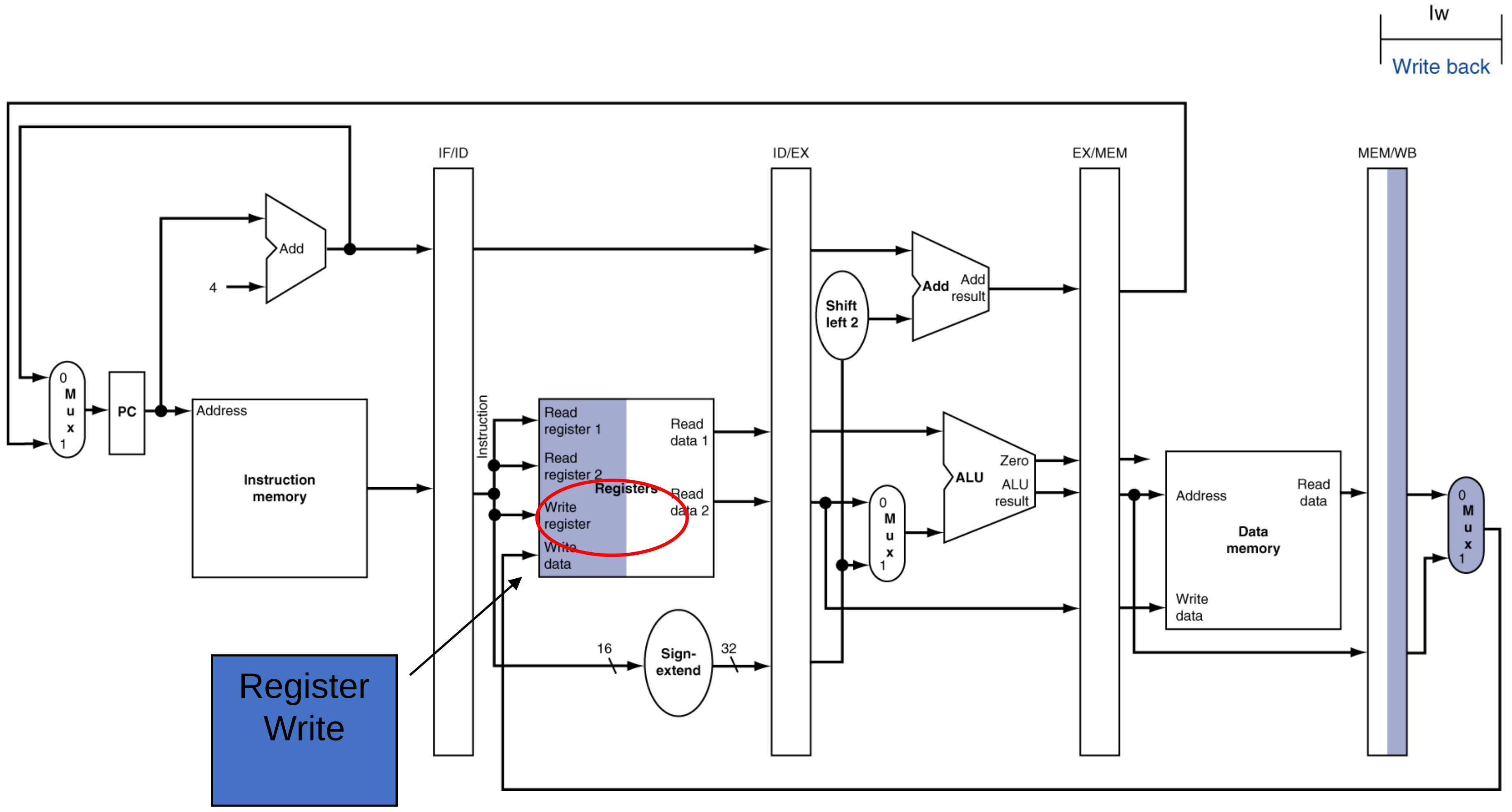

lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
Add
Add result
Shift left 2
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

MEM for Load



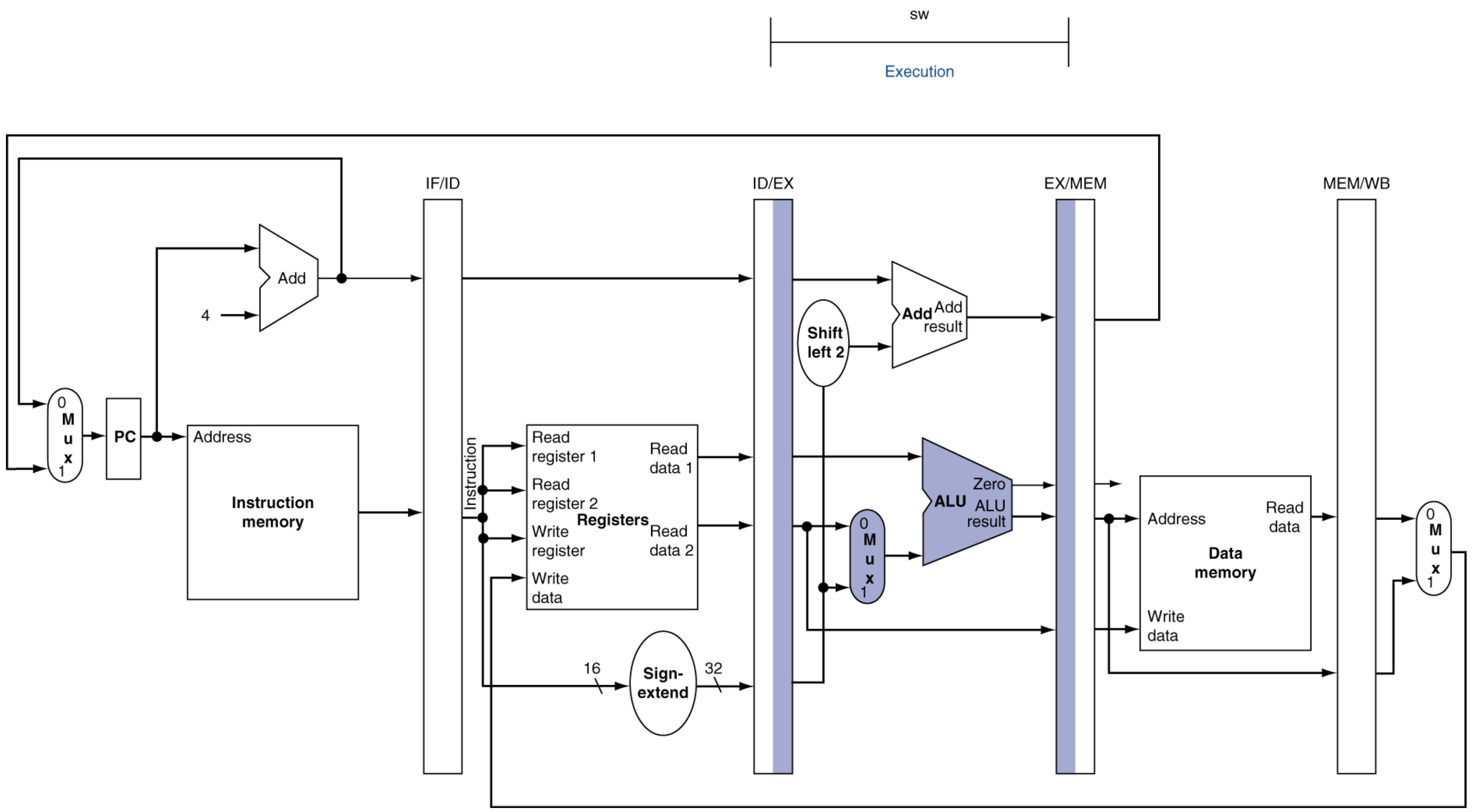
lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

WB for Load

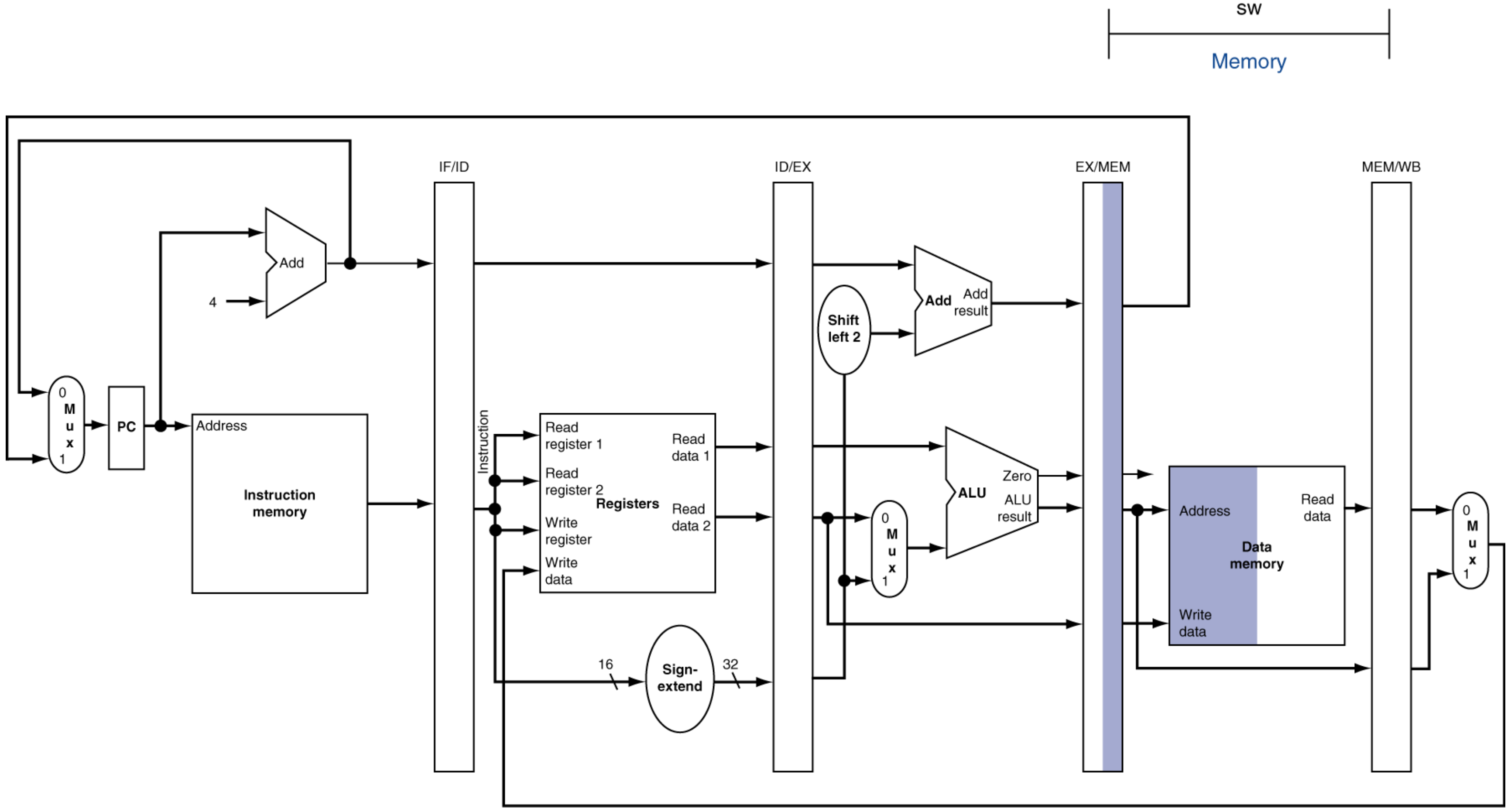


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
Register Write

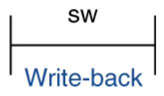
EX for Store

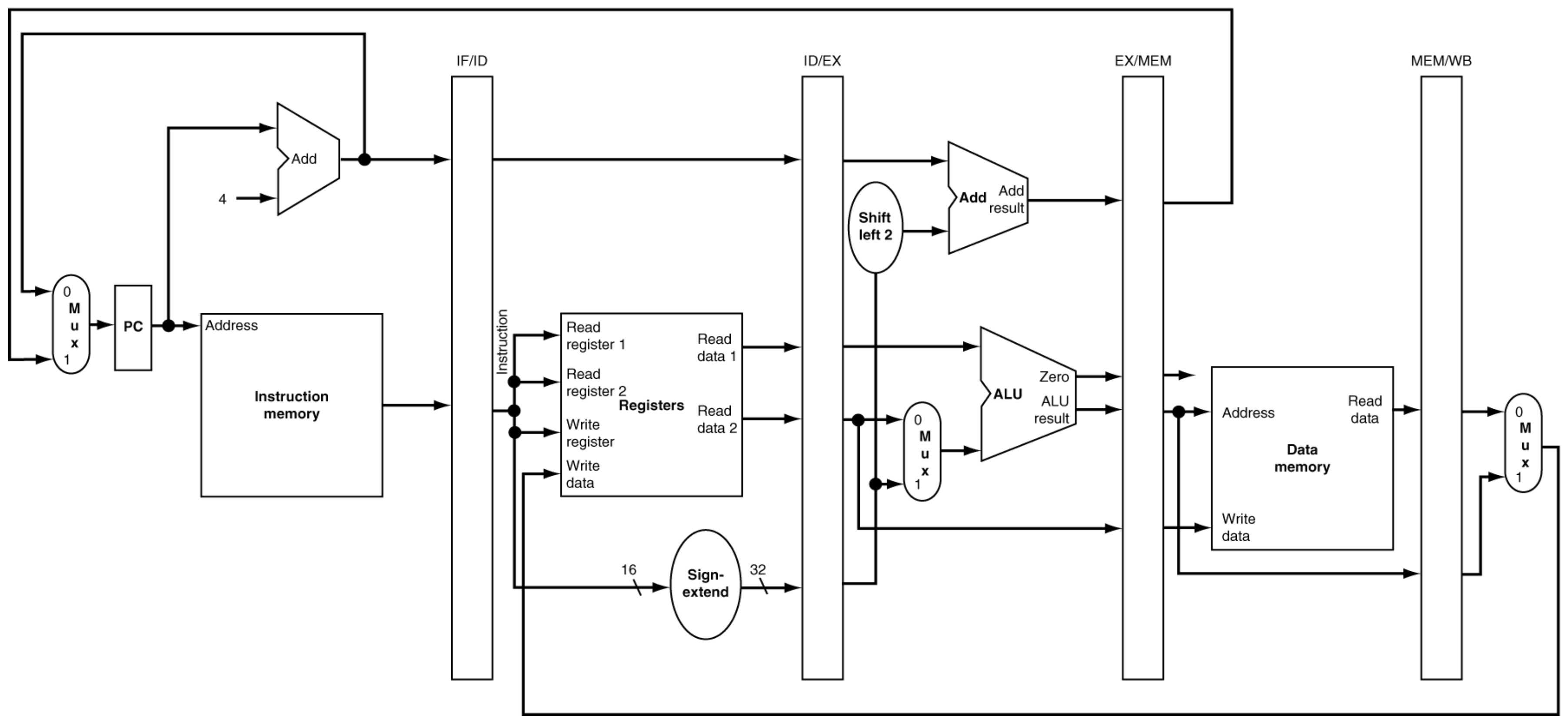

sw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

MEM for Store

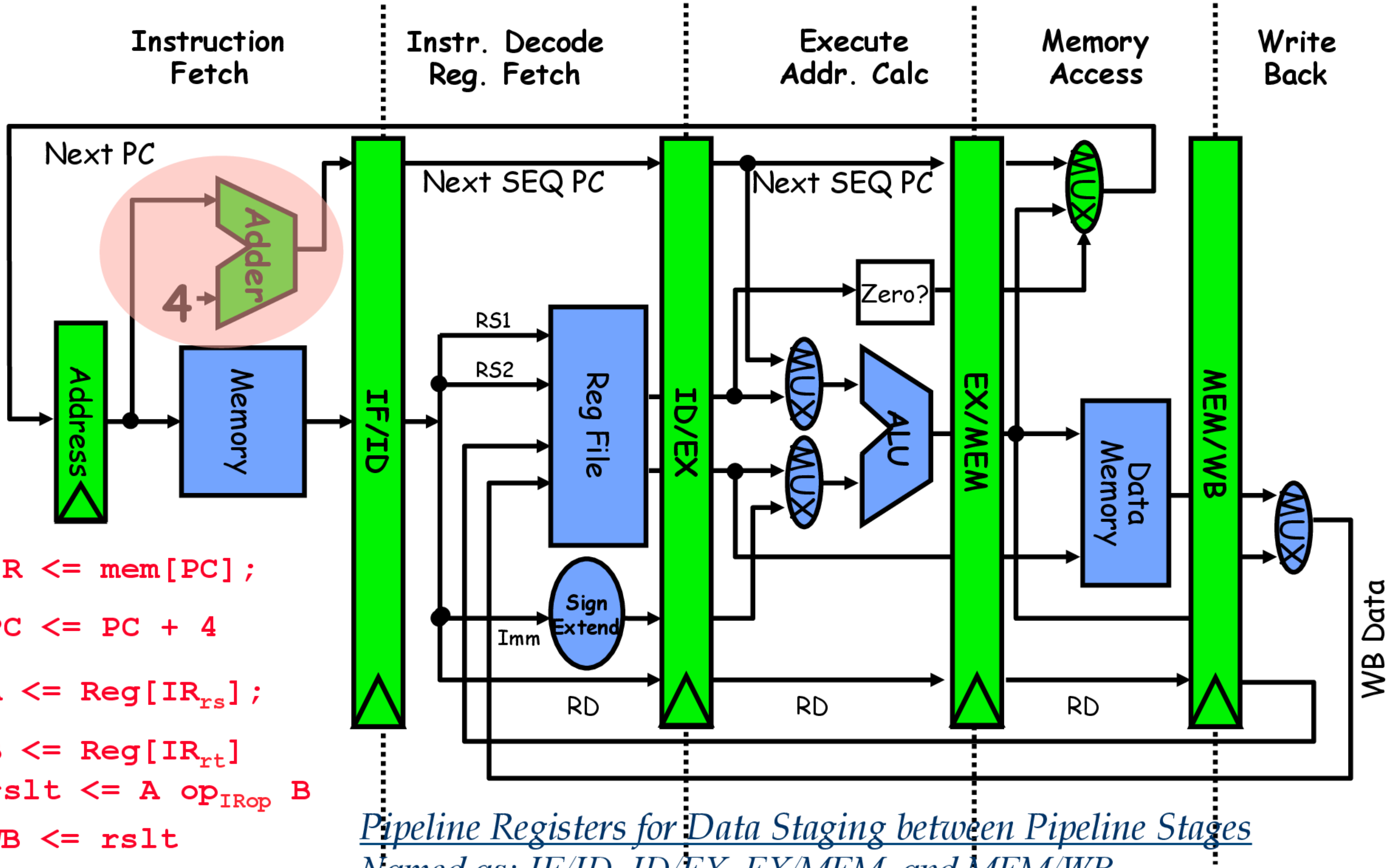


WB for Store


sw
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
0
M
u
x
1
Address
Data memory
Write data
Read data
0
M
u
x
1

Pipeline Registers



IR <= mem[PC];

PC <= PC + 4

A <= Reg[IR_{rs}];

B <= Reg[IR_{rt}]

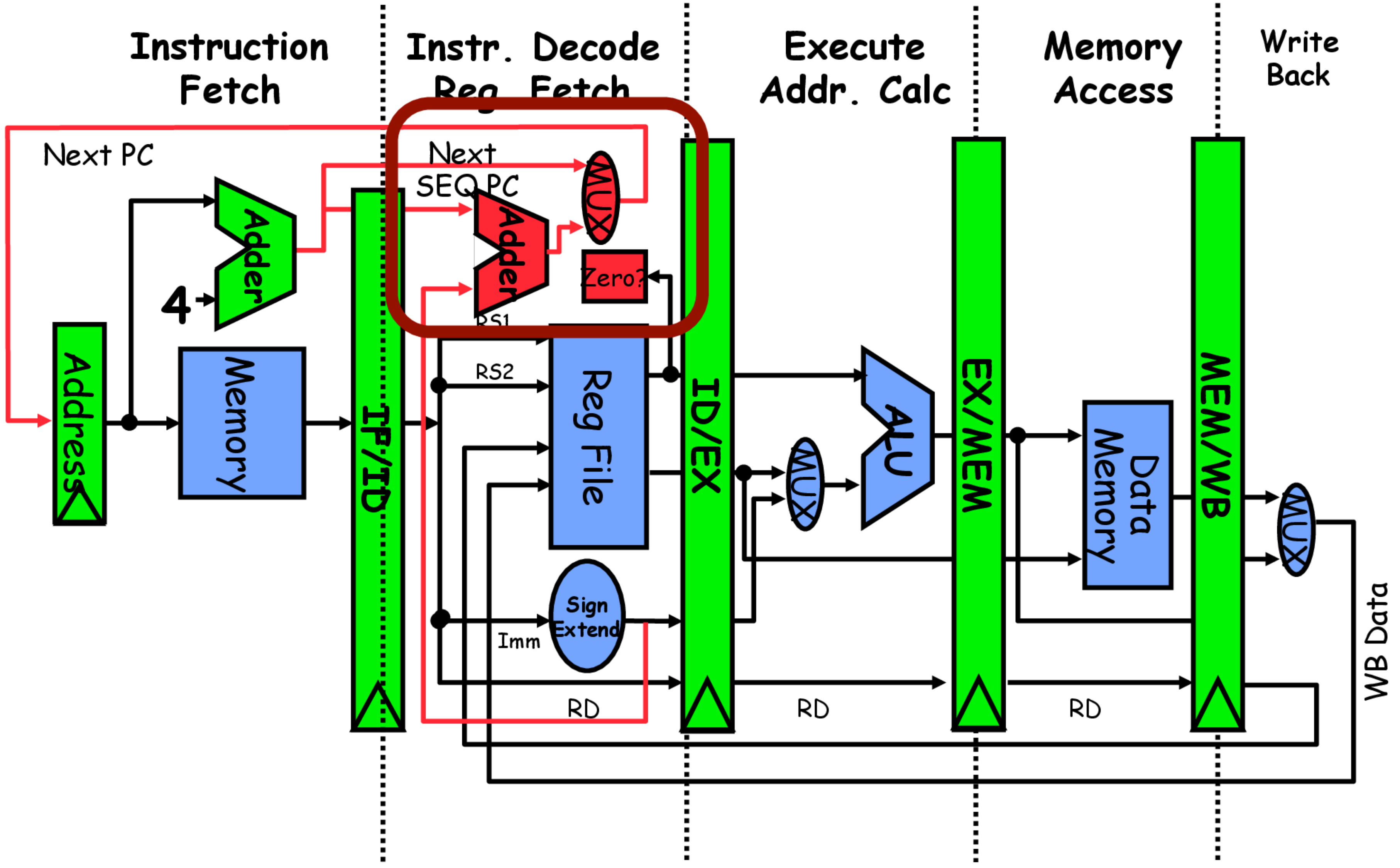
rslt <= A op_{IRop} B

WB <= rslt

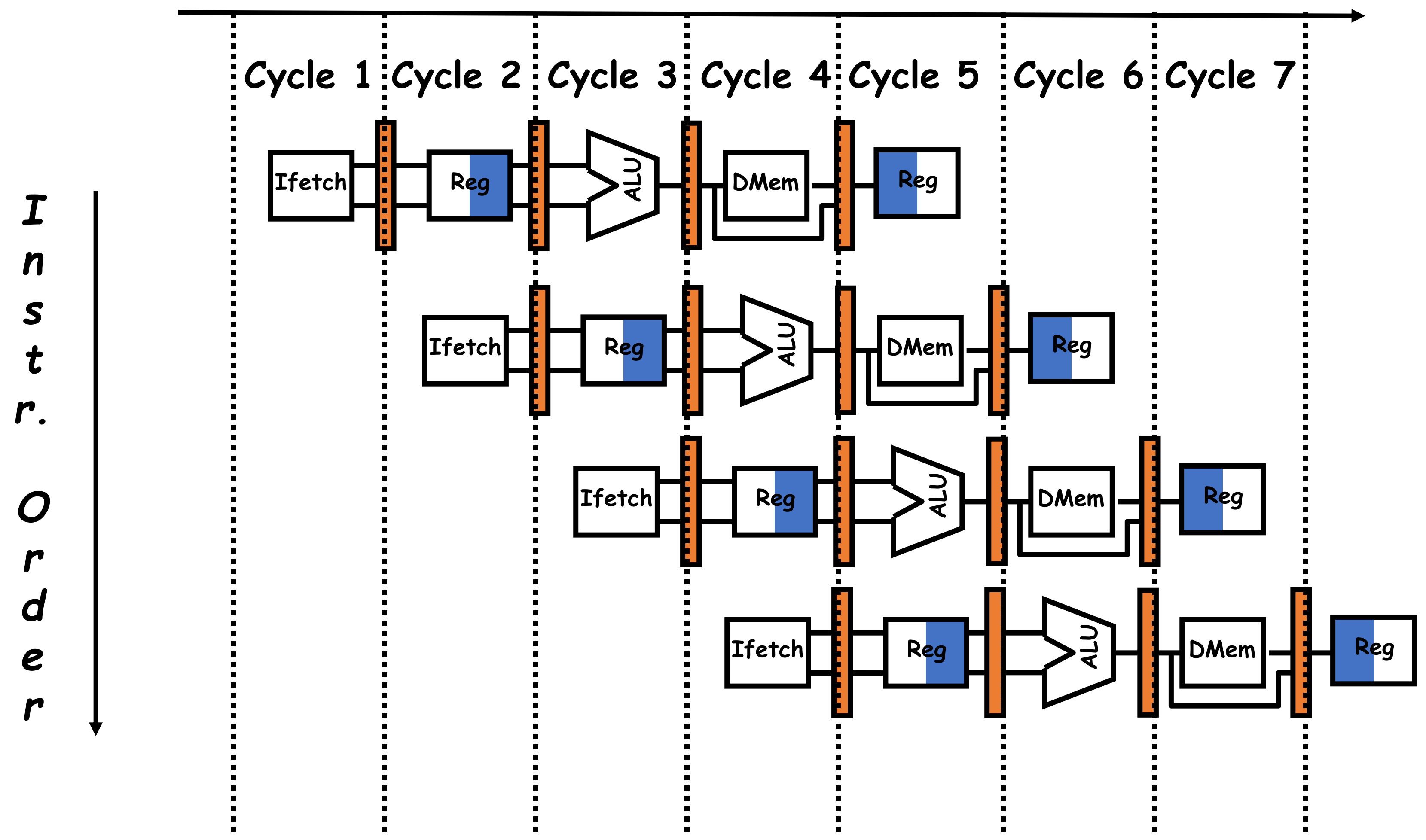
Reg[IR_{rd}] <= WB

Pipeline Registers for Data Staging between Pipeline Stages Named as: IF/ID, ID/EX, EX/MEM, and MEM/WB

Pipelined MIPS Datapath

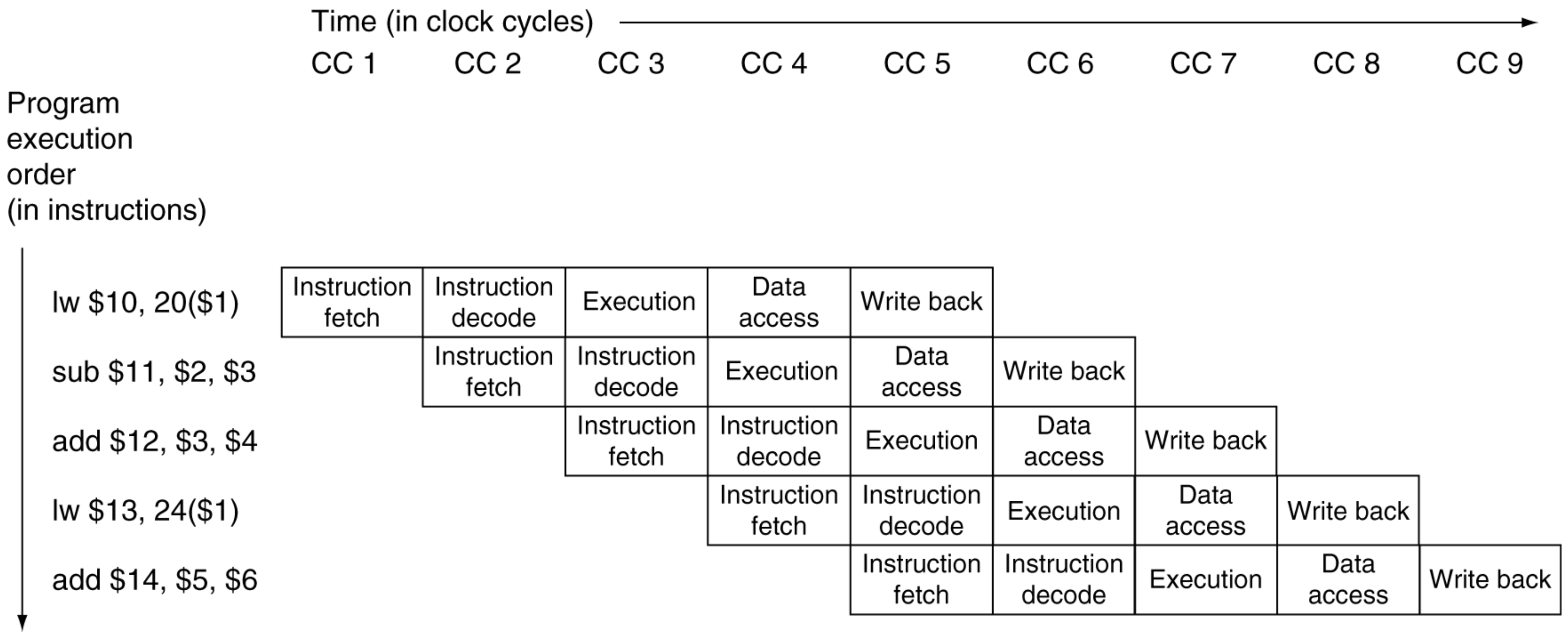


The Basic Pipeline For MIPS – Simplified Diagram



Multi-Cycle Pipeline Diagram

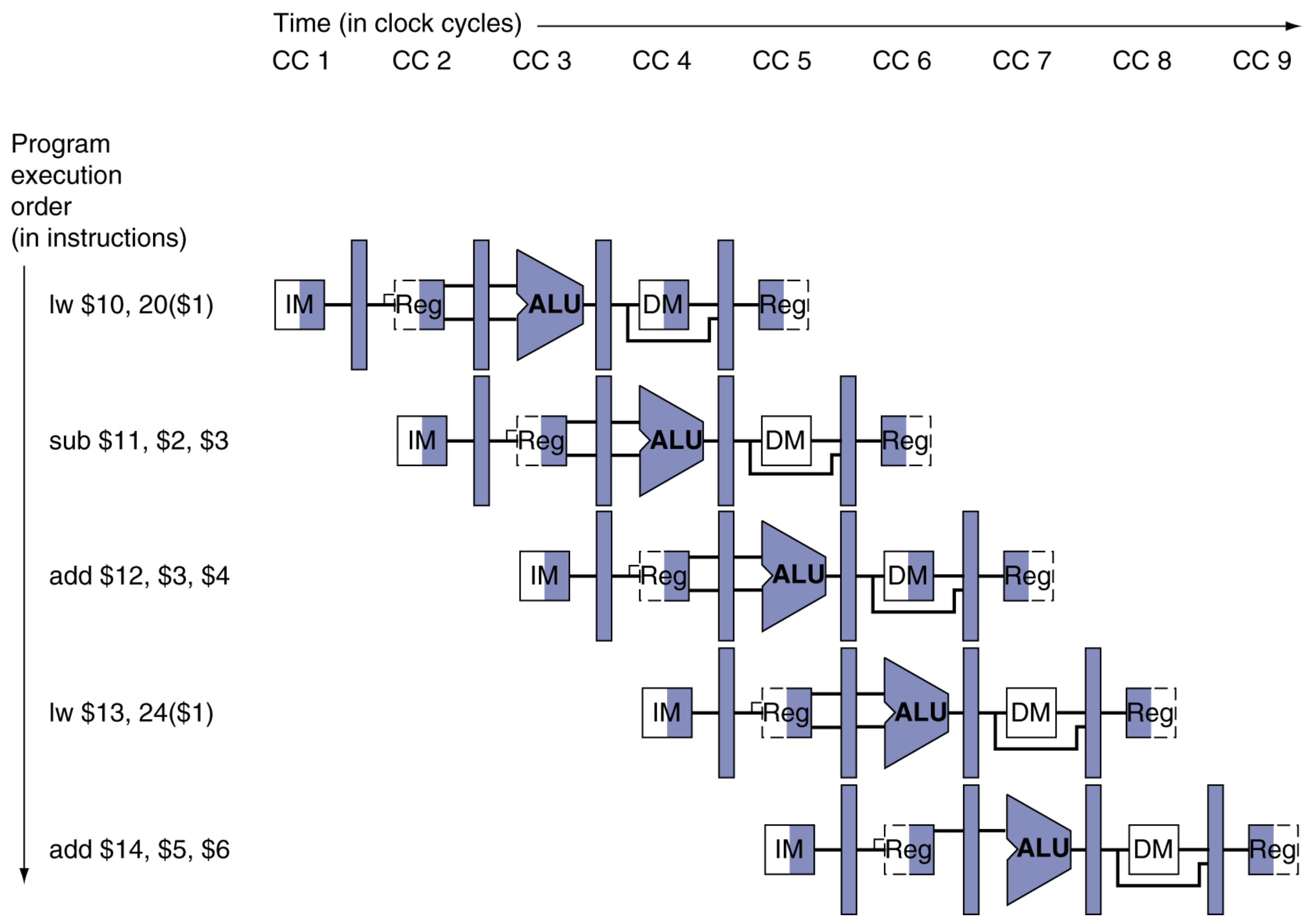
- Traditional form



	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw $10, 20($1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub $11, $2, $3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add $12, $3, $4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw $13, 24($1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add $14, $5, $6					Instruction fetch	Instruction decode	Execution	Data access	Write back

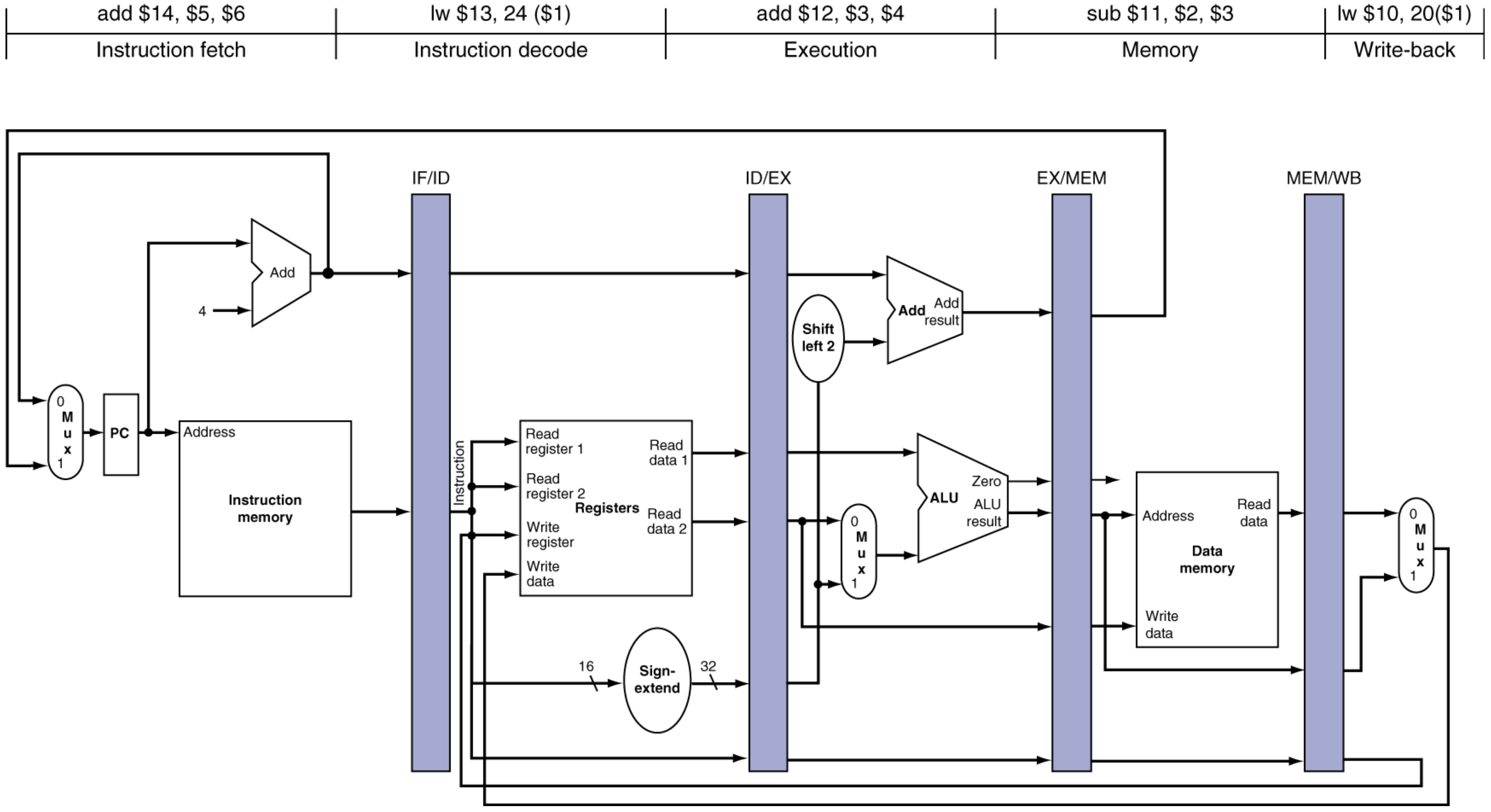
Multi-Cycle Pipeline Diagram

- Form showing resource usage for different instructions



Single-Cycle Pipelined Diagram

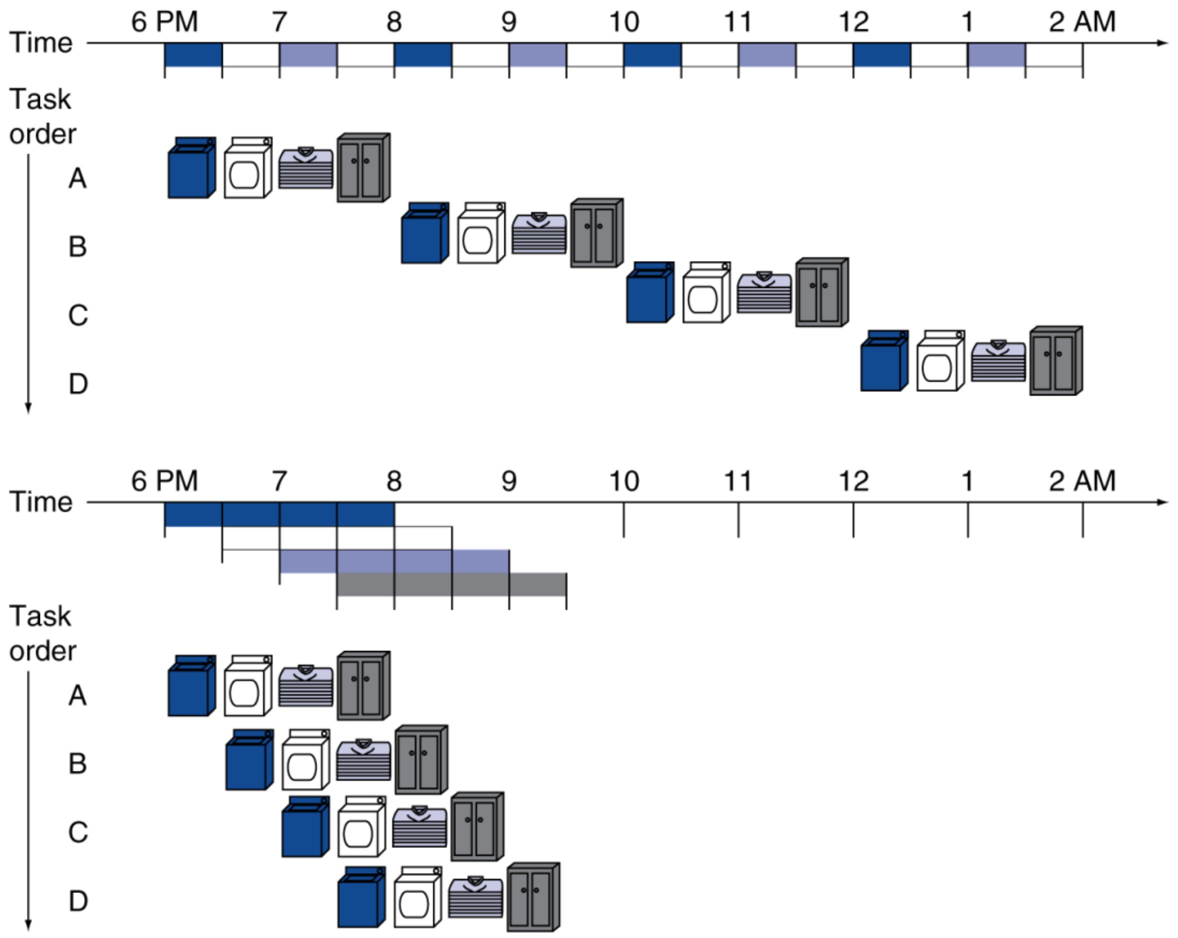
- State of pipeline in a given cycle



Pipeline Performance Estimation

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:
  - Speedup = 8/3.5 = 2.3

Pipelined Architecture

- Example program

CT=1.5ns; CPU Time = 9 x 1 x 1.5ns = 13.5ns

What is the cost of pipelining?

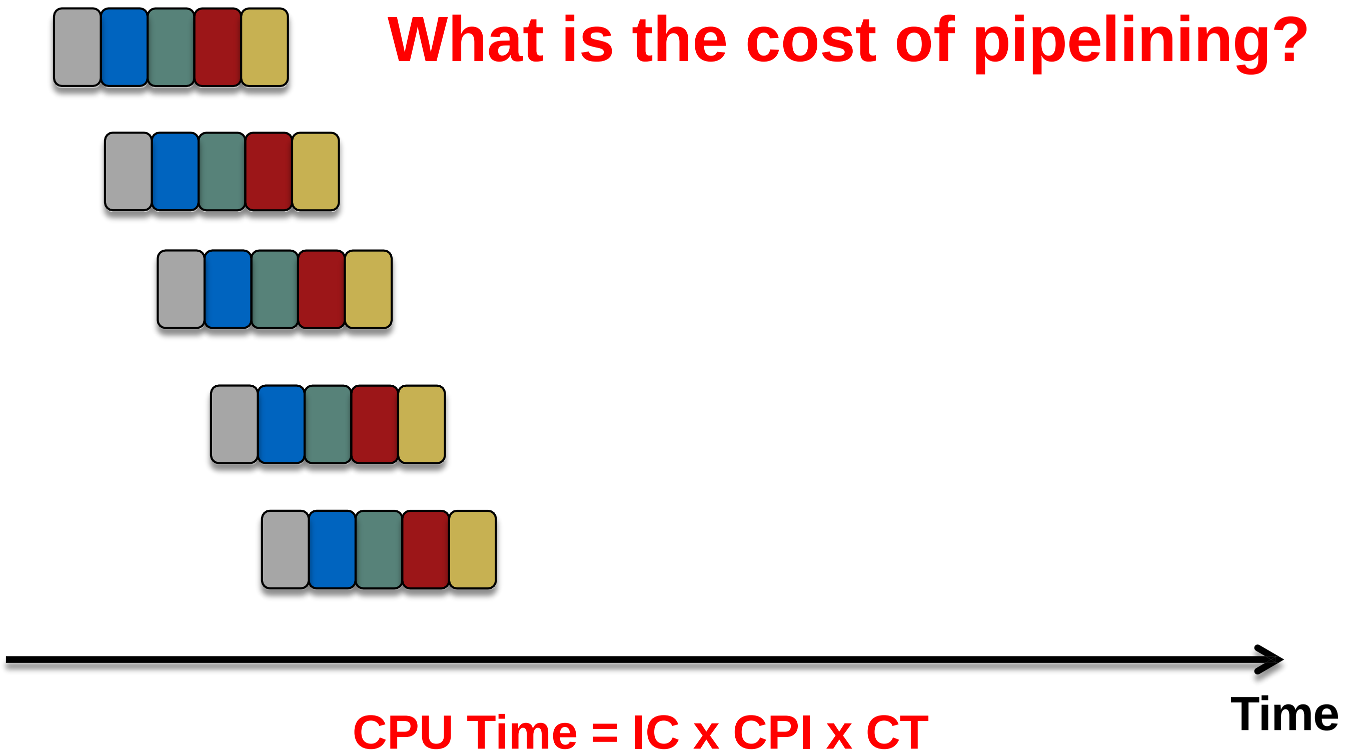
AND R1,R2,R3

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6



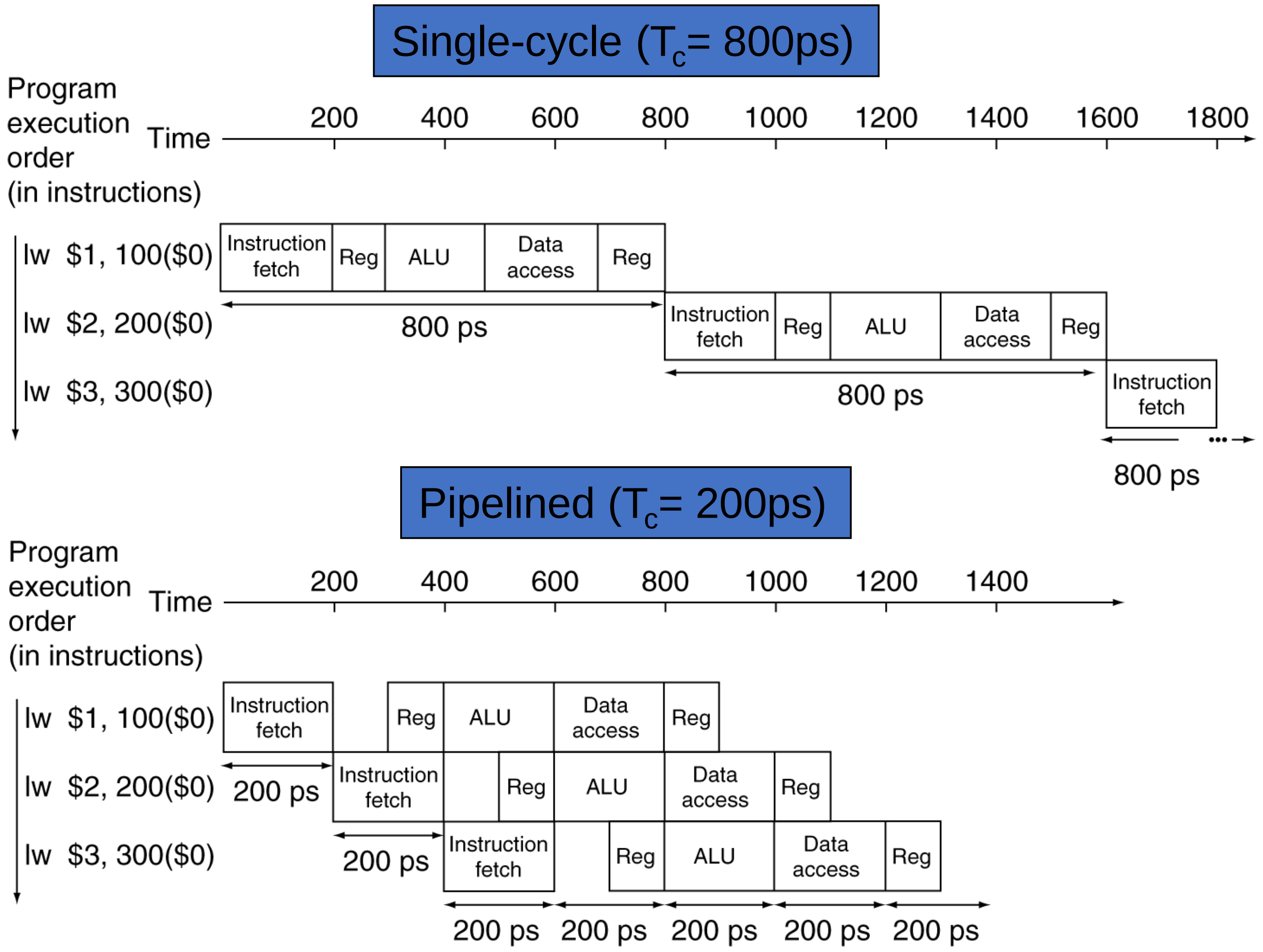
CPU Time = IC x CPI x CT

Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance



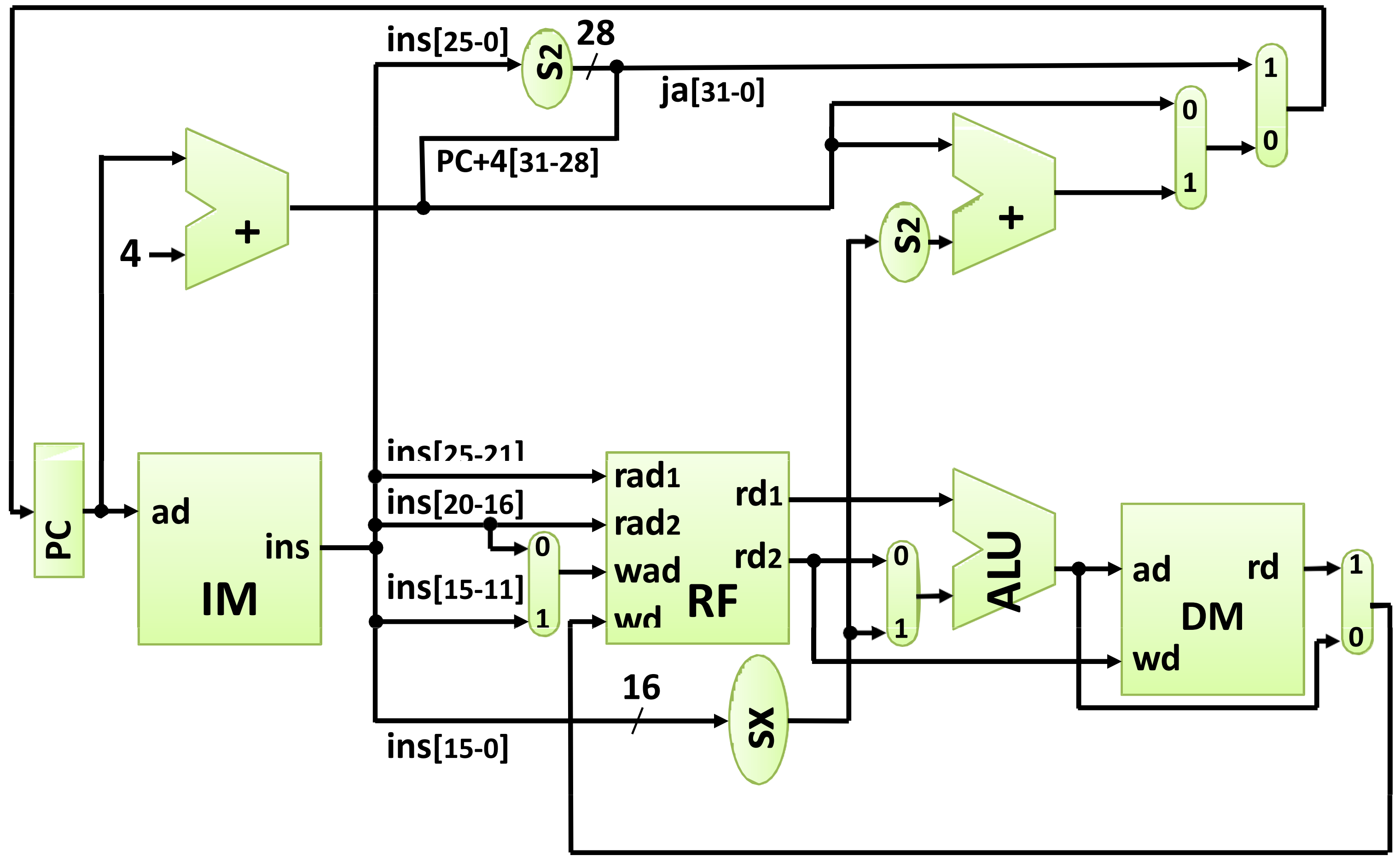
Improving Resource Utilization

Improving CPU Resource Utilization

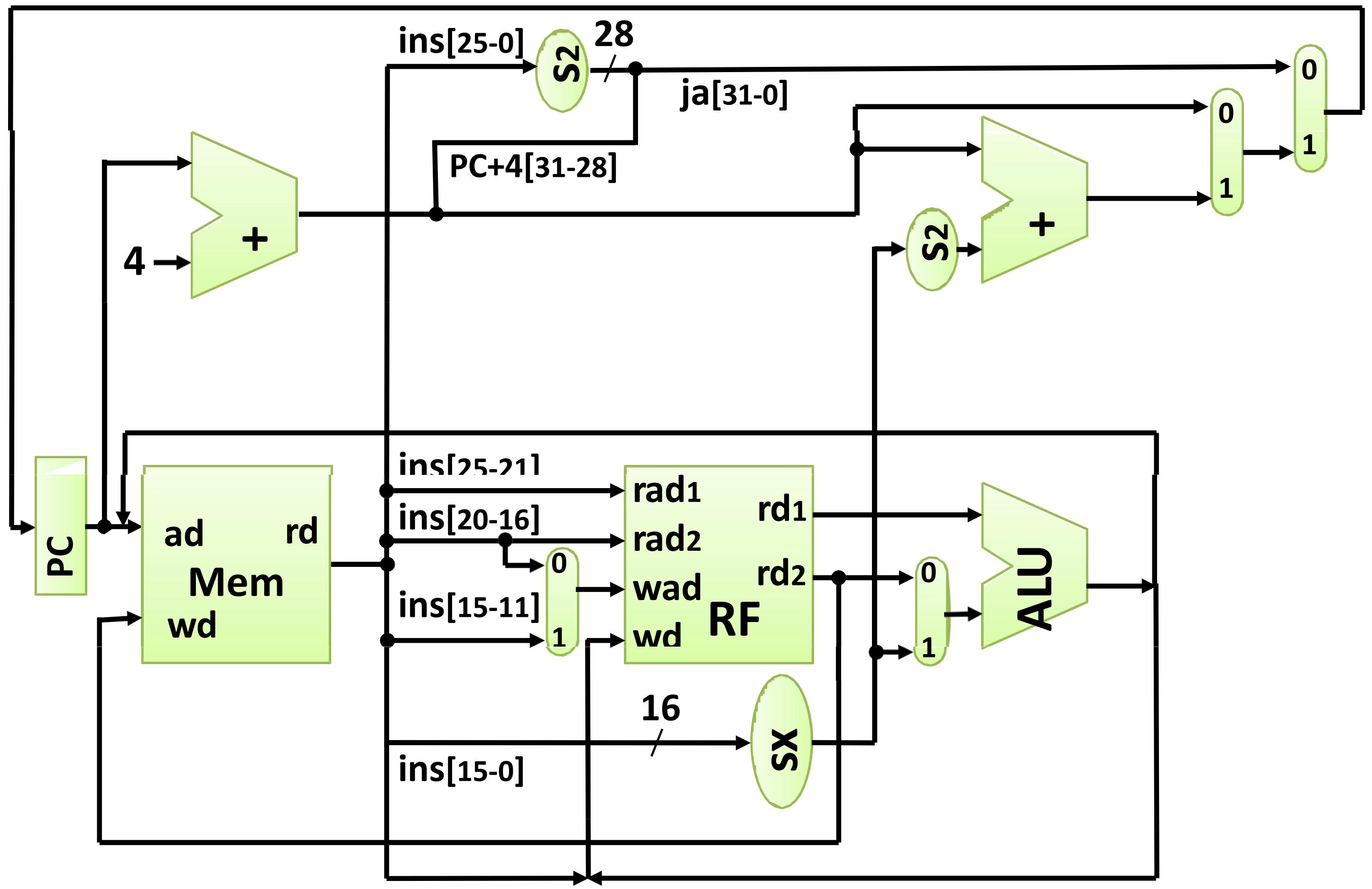
Efficient MIPS CPU Design

- Can we eliminate two adders?
- How to share (or reuse) a resource (say ALU) in different clock cycles?
- Store results in registers.
- Of course, more multiplexing may be required!
- Resources in this design: RF, ALU, MEM.

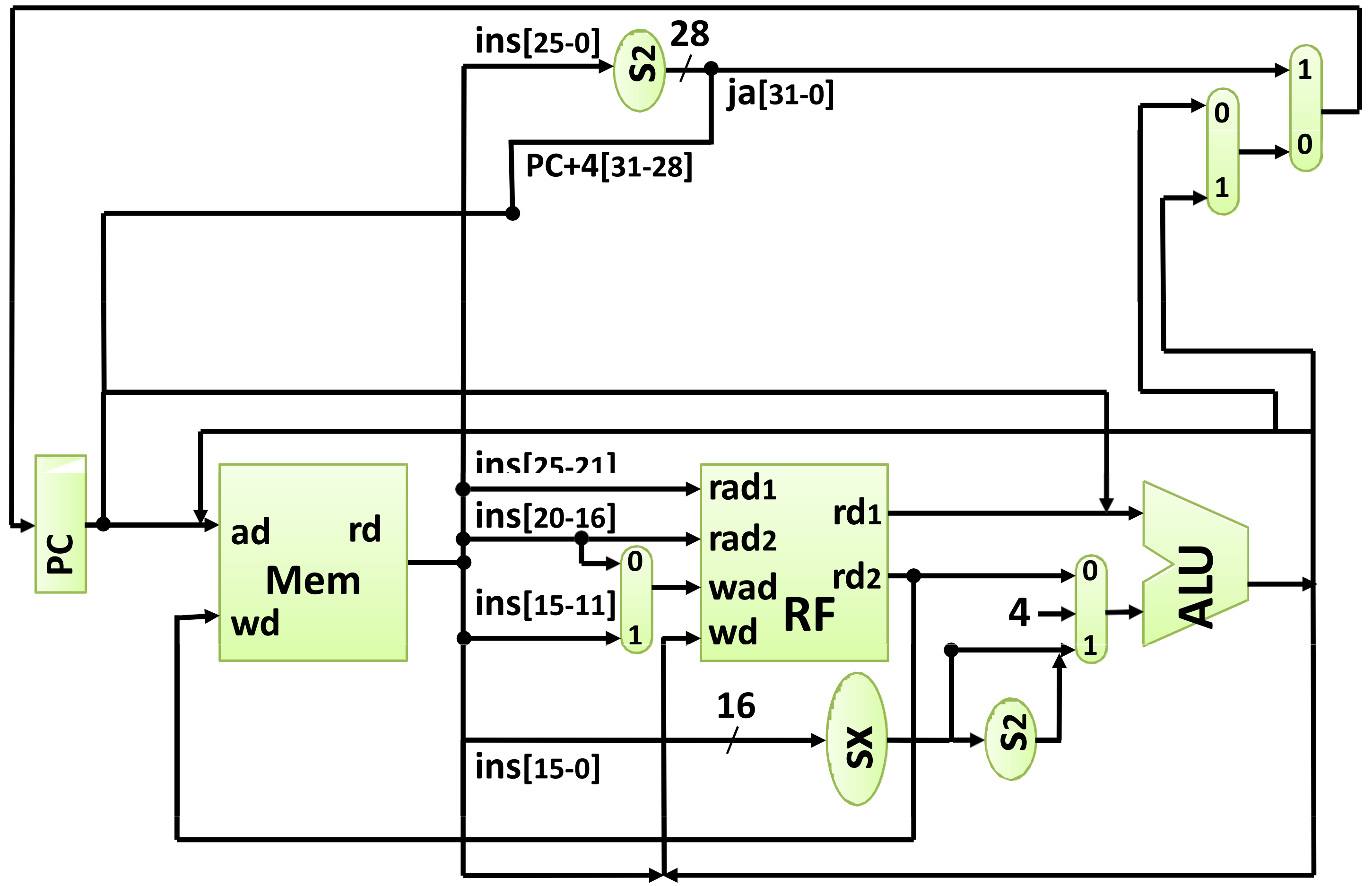
Single Cycle Datapath



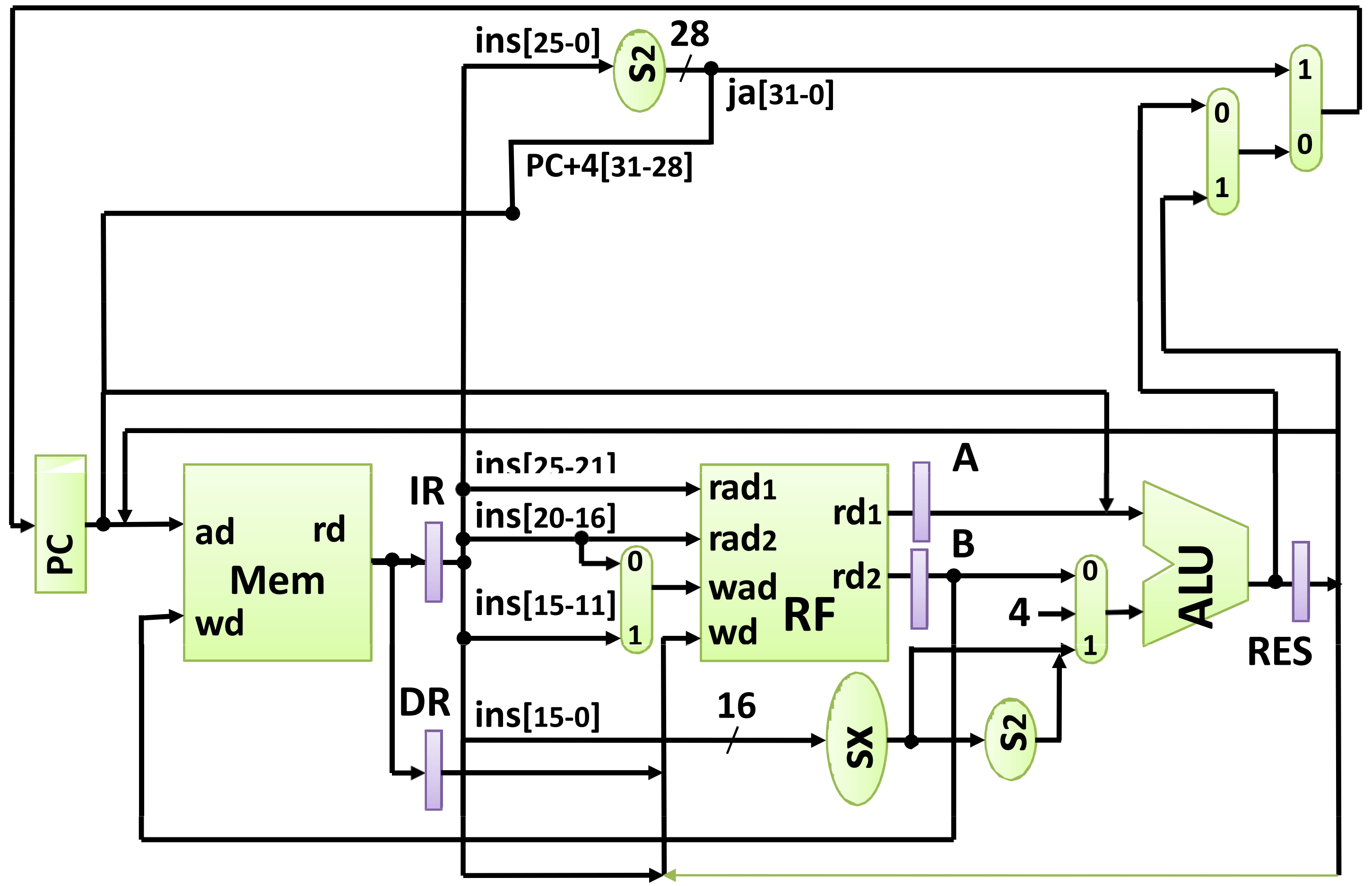
Merge IM and DM, Rearrange Diagram



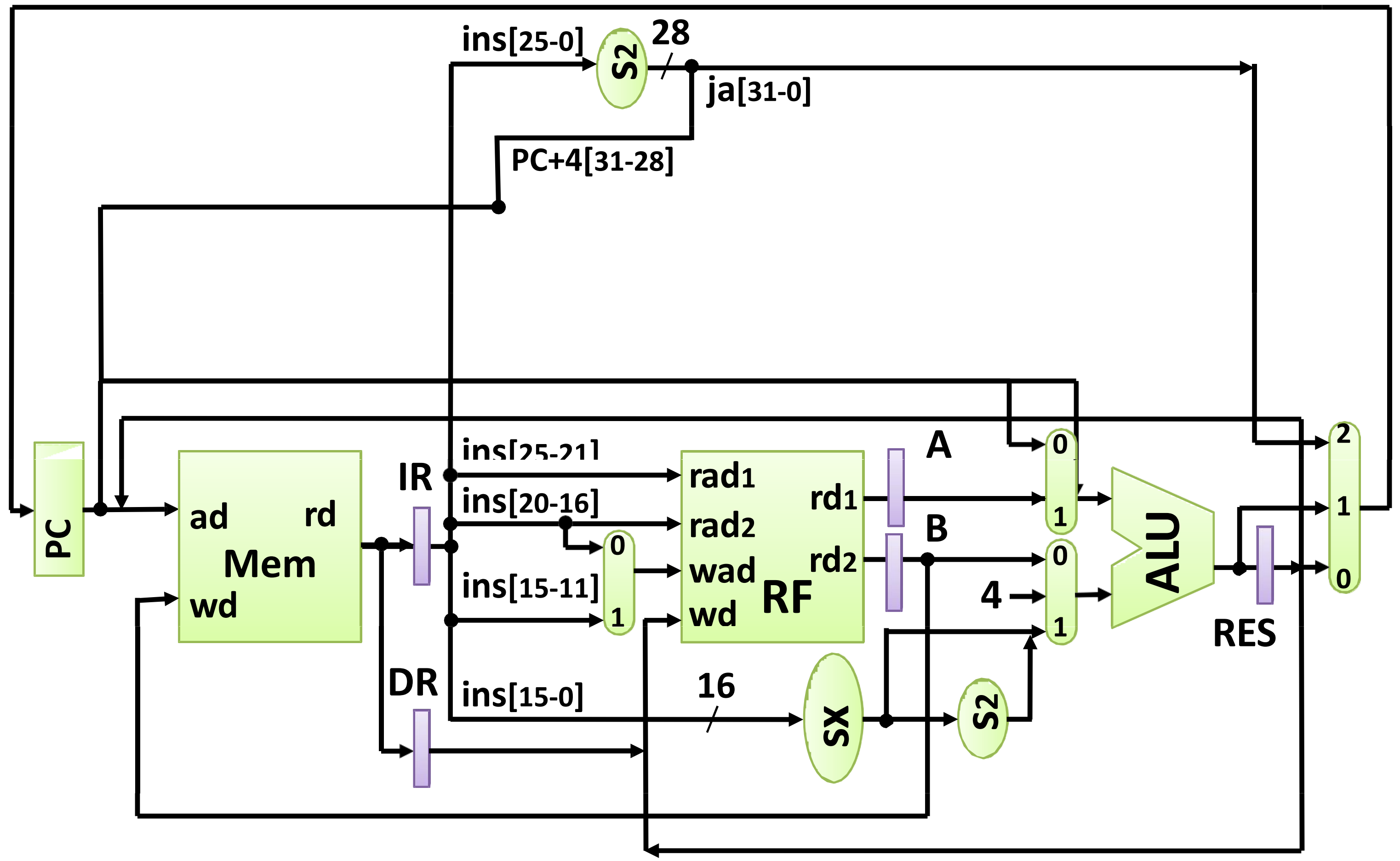
Combine Adders - Rearrange Diagram



Introduce registers

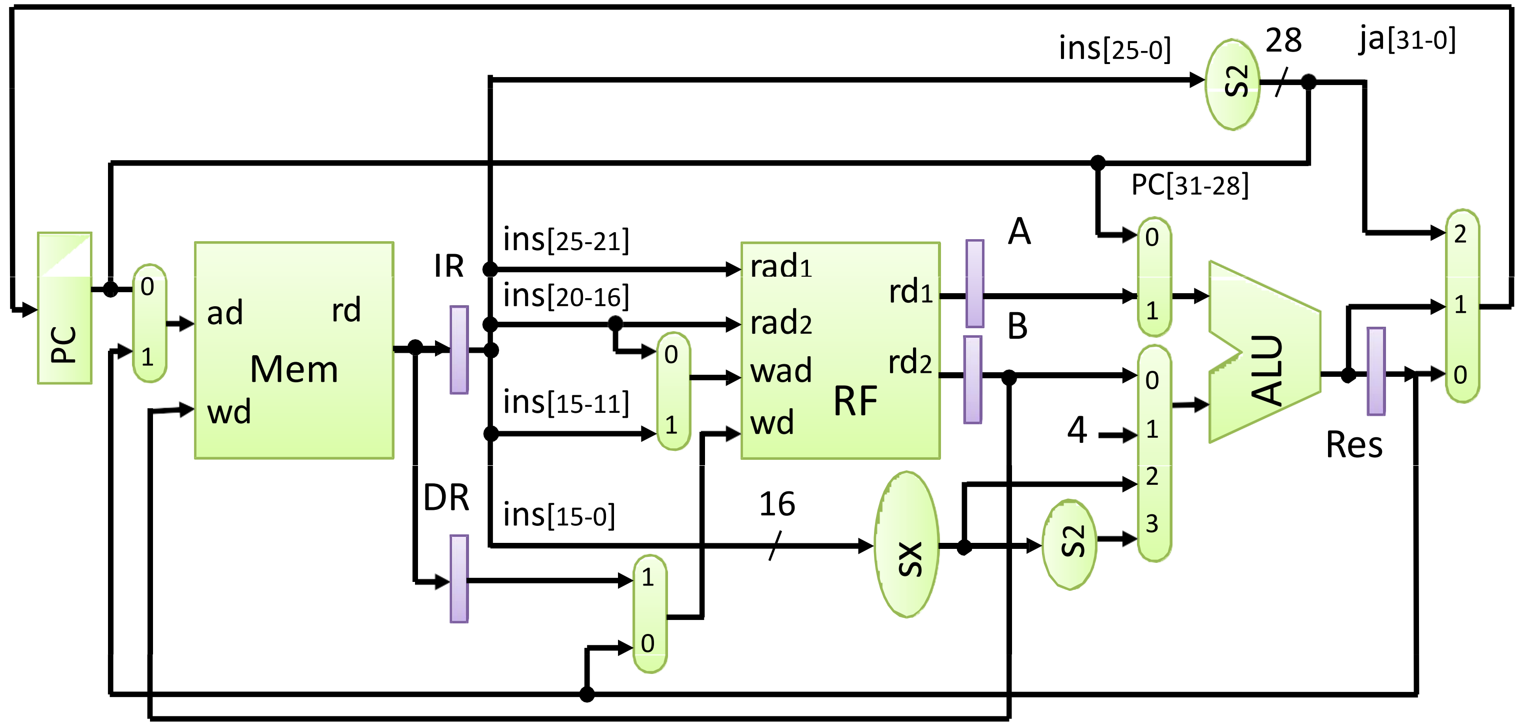


Introduce ALU Multiplexers at Input and Output



Final: Efficient Multi-Cycle Data Path

- Mux at Memory Input
- Mux at Register File Input
- One memory for lw / sw
- One ALU for EXE and Address Calculation
- Several MUXes introduced



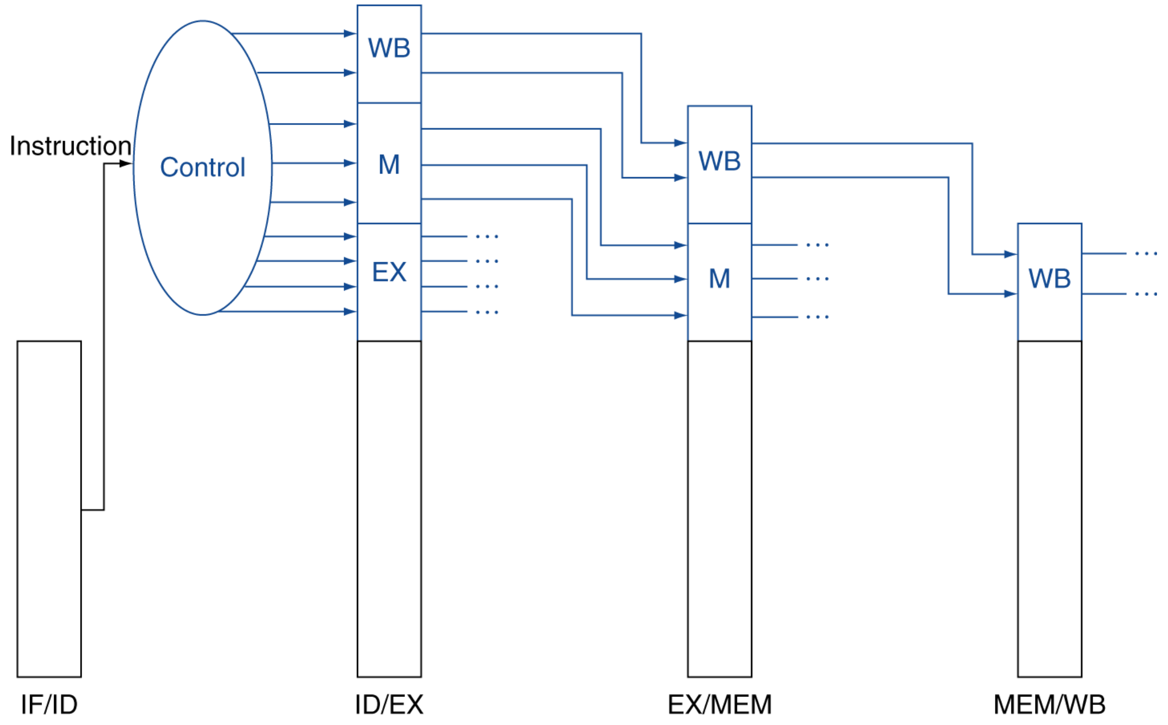
Multi-cycle Control Path

Multi-cycle Control Design

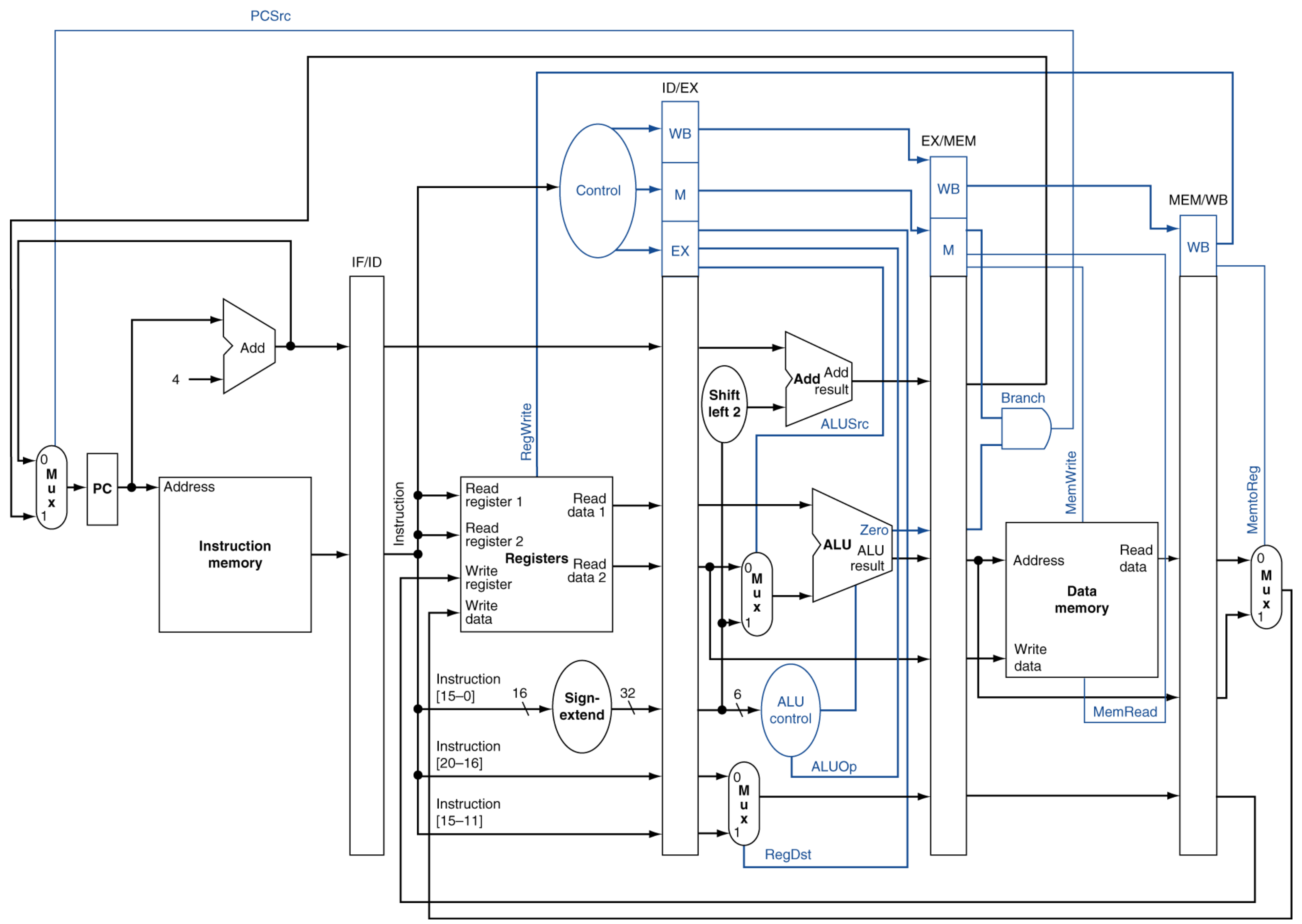
- Break instructions into cycles
- Put cycle sequences together
- Control signal groups and micro operations
- Control states and signal values
- Control state transitions

Pipelined Control

- Control signals derived from instruction
    - As in single-cycle implementation



Pipelined Control with Connections



Readings

- Chap 4 of P&H Textbook