CMP301B/CMPN301: Computer Architecture

Pipelining

Mayada Hadhoud
Computer Engineering Department
Cairo University

Agenda

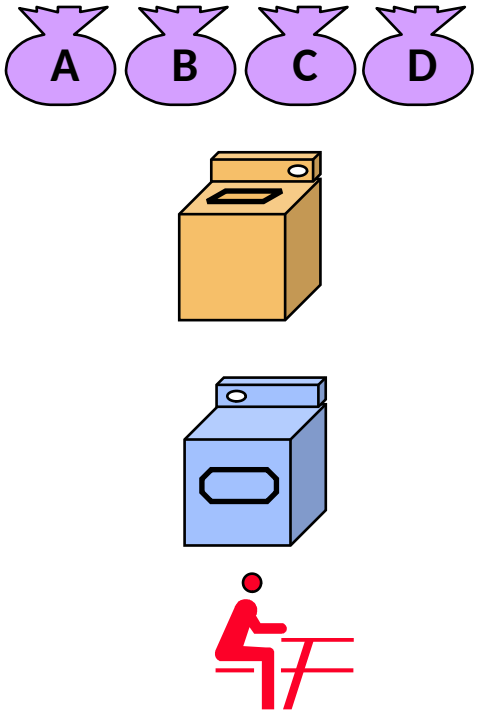
- What is pipelining?
- Characteristics of pipelining
- Pipelining Hazards
  - Structural Hazard
  - Data Hazard
  - Control Hazard

What Is A Pipeline?

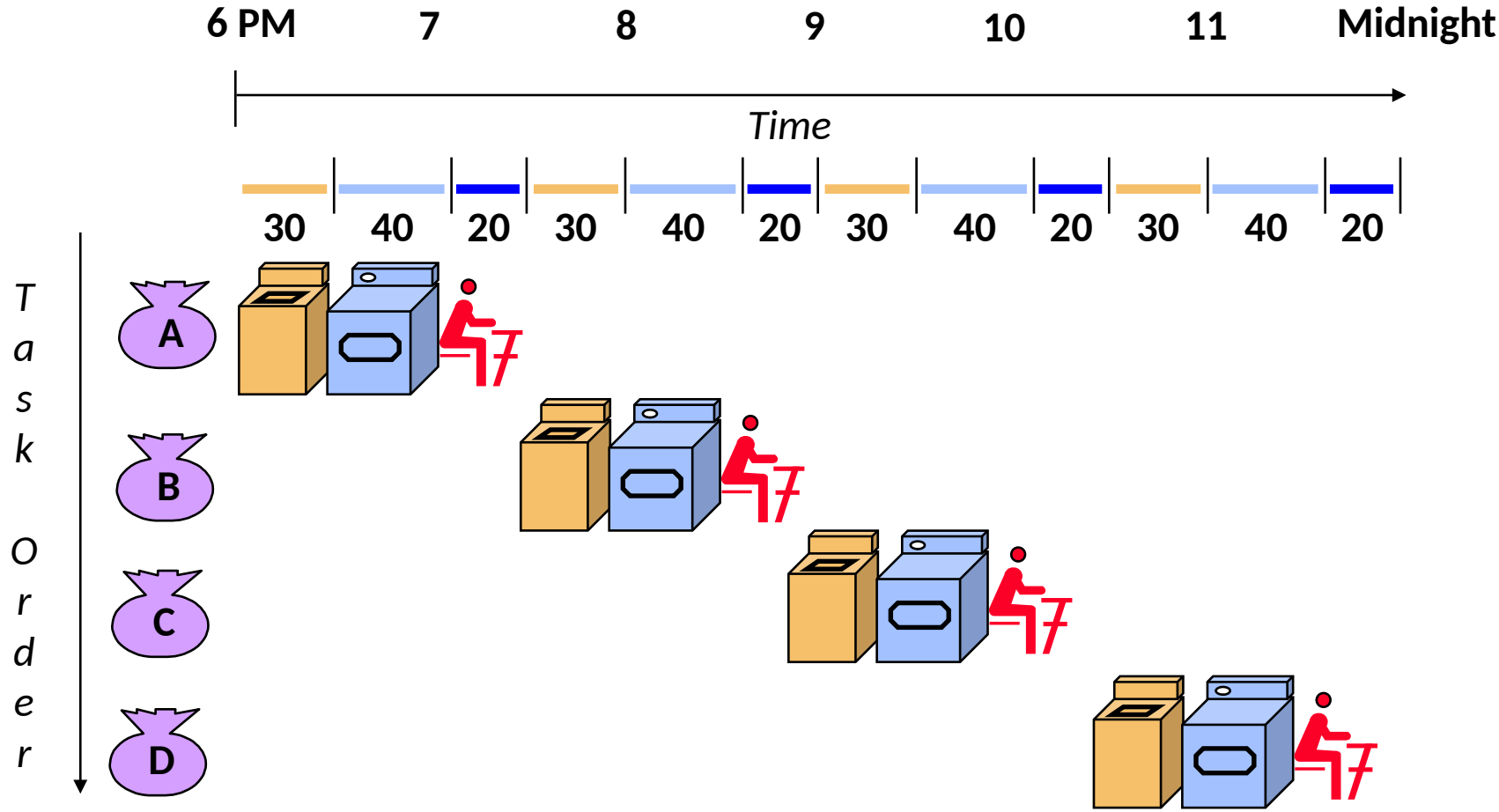
- Pipelining is used by virtually all modern microprocessors to enhance performance by **overlapping the execution of instructions.**

What Is Pipelining

- Laundry Example
- 4 persons each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



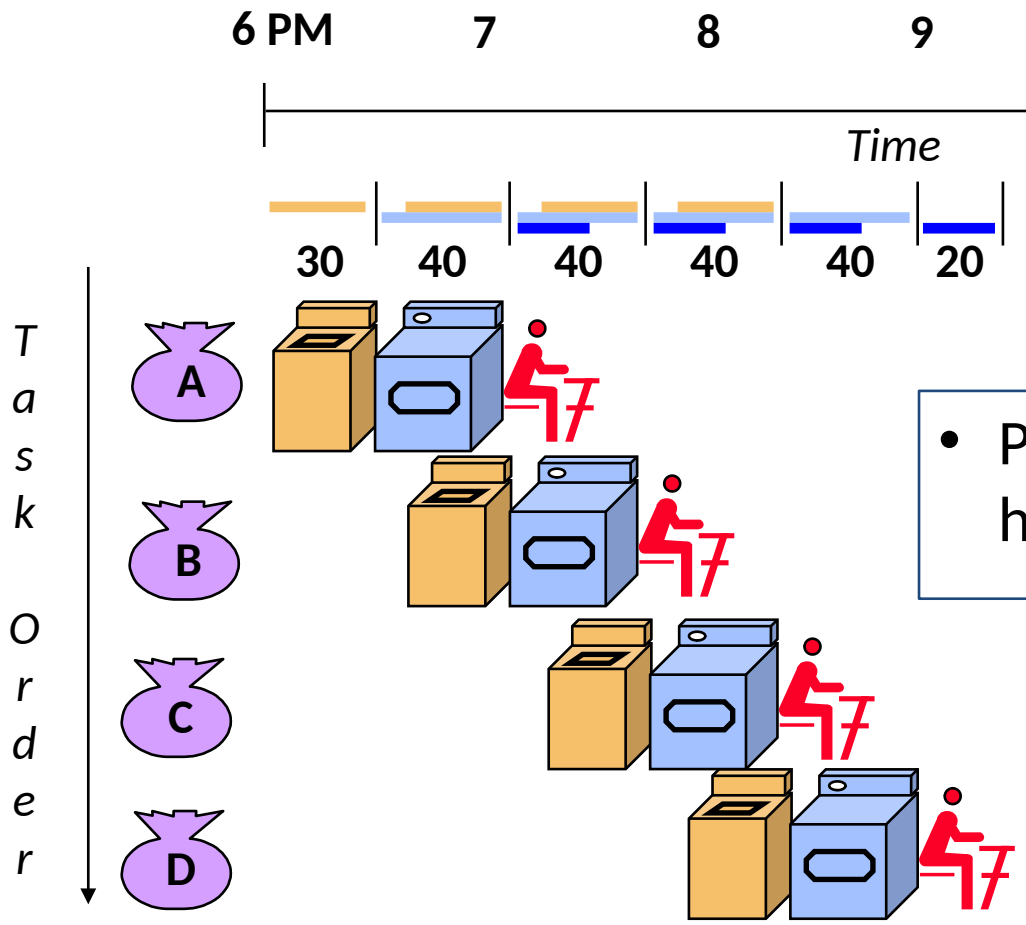
What Is Pipelining



Sequential laundry takes 6 hours for 4 loads

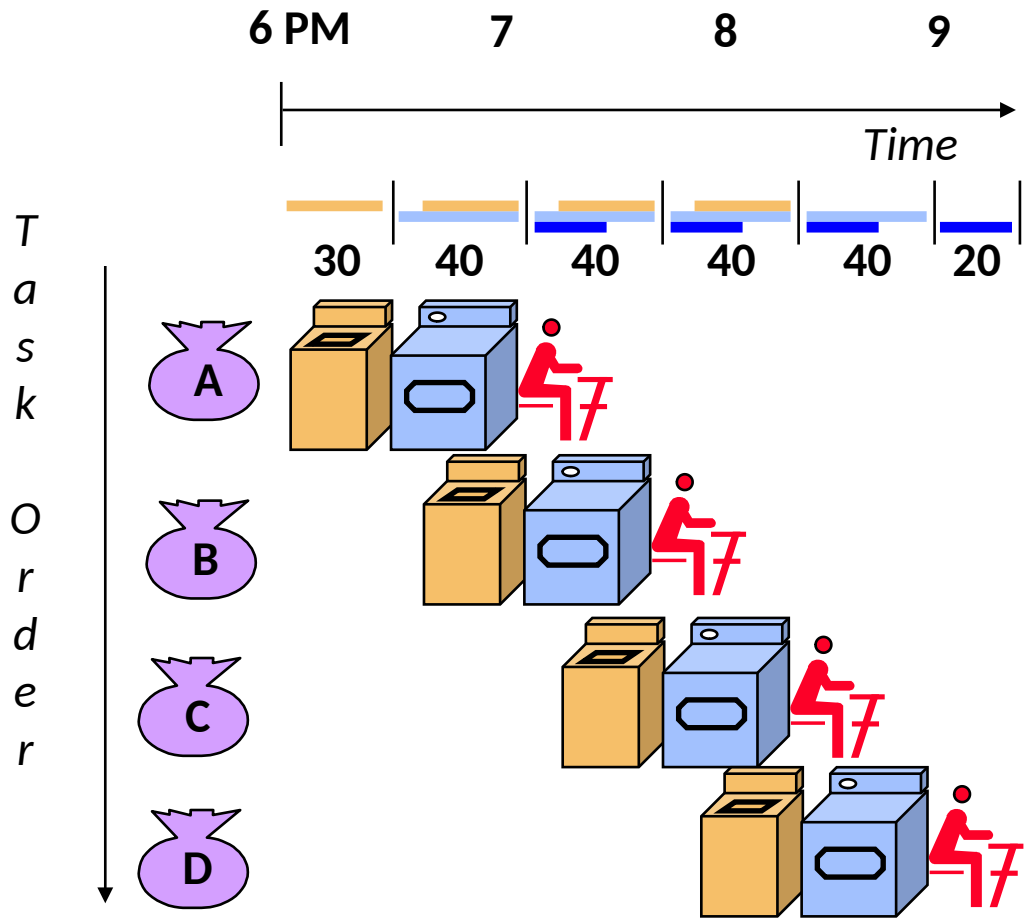
If they learned pipelining, how long would laundry take?

What Is Pipelining
Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

What Is Pipelining

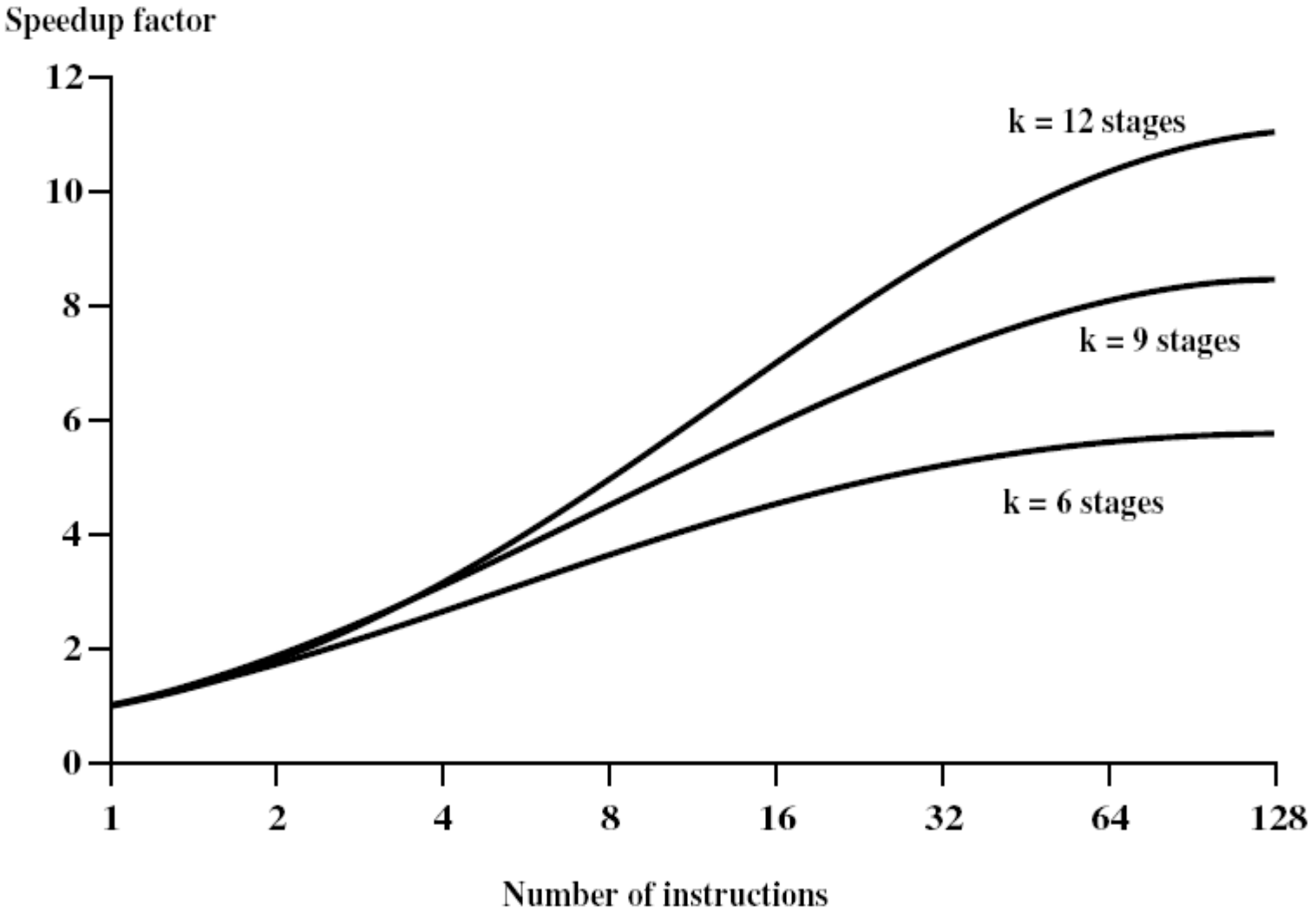


Pipelining Lessons

- 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 pipe stages**
- **Unbalanced** lengths of pipe stages **reduces** speedup
- Time to "fill" pipeline and time to "drain

Pipelining Theoretical Performance

- An ideal pipeline divides a task into **k independent sequential subtasks**
  - Each subtask requires 1 time unit to complete
  - The task itself requires k time units to complete
- For **n** iterations of task, the execution times:
  - **With no pipelining: nk time units**
  - **With pipelining: k + (n-1) time units**
- Speedup of a k-stage pipeline is
  - **S = nk/[k+(n-1)] → = k for large n**


Speedup factor
12
10
8
6
4
2
0
k = 12 stages
k = 9 stages
k = 6 stages
1
2
4
8
16
32
64
128
Number of instructions

Characteristics Of Pipelining

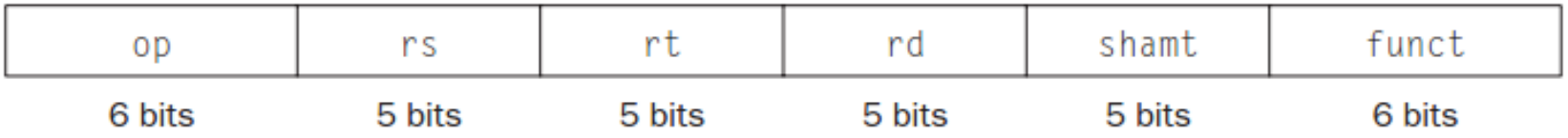
- The previous expression is **ideal**.
- In terms of a CPU, the implementation of pipelining has the effect of **reducing the average instruction time**, therefore **reducing the average CPI.**
- **EX**: If each instruction in a microprocessor takes 5 clock cycles (unpipelined) and we have a 4 stage pipeline, the ideal average CPI with the pipeline will be 1.25 .

RISC Instruction Set Basics (MIPS)

- **Properties of RISC architectures:**
  - All operations on data apply to data in registers and typically change the entire register (32-bits or 64-bits).
  - The only operations that affect memory are load/store operations. Memory to register, and register to memory.
  - Usually instructions are few in number and are typically **one size.**

RISC Instruction Set Basics (MIPS)
Types of Instructions

- **ALU Instructions (R-type)**:
  - Arithmetic operations, take two registers as operands. The result is stored in a third register.
  - Logical operations AND OR, XOR, shift



R-Type Instruction Example

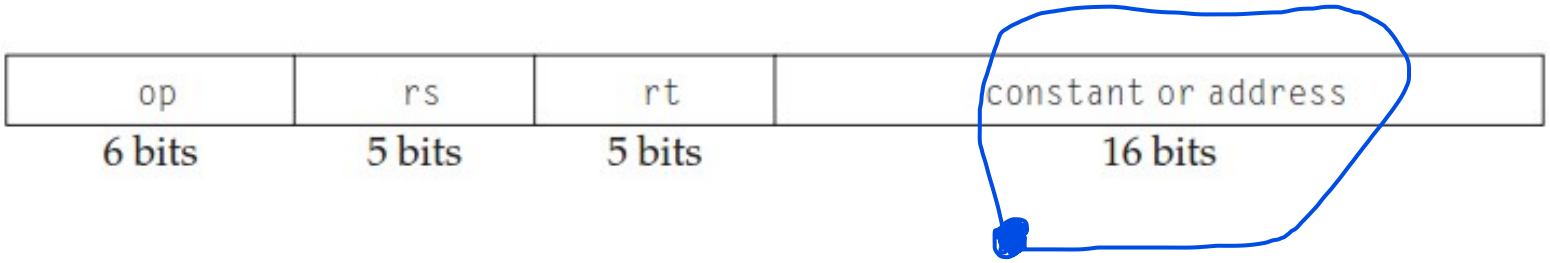
```
add $3,$17,$3
```

MEANING	comp	$17	$18	$3	-	add
Decimal	0	17	18	3	0	32
Binary	00 0000	1 0001	1 0010	0 0011	0 0000	10 0000
Hex	00	11	12	03	00	20

RISC Instruction Set Basics (MIPS)
Types of Instructions

Immediate Format Instructions (I-type):

- Usually take a register (base register) as an operand and a 16-bit immediate value. The sum of the two will create the *effective address*. A second register acts as a source in the case of a load operation.
- In the case of a store operation the second register contains the data to be stored.



I-Type Instruction Example

```
examples:
addi  Rd,Rs,N          ; Rd = Rs + SignExt(N)
ori   Rd,Rs,N          ; Rd = Rs | SignExt(N)
beq   Rs,Rt,Label      ;If(Rs == Rt) goto Label
lw    Rt,N(Rs)         ;Rt = Mem[Rs + SignExt(N)]
sw    Rt,N(Rs)         ;Mem[Rs + SignExt(N)] = Rt
```

```
addi $8,$22,-16
```

MEANING	addi	$22	$8	-16
Decimal	9	22	8	-16
Binary	00 1001	1 0110	0 1000	1111 1111 1111 0000
Hex	09	16	08	FFF0

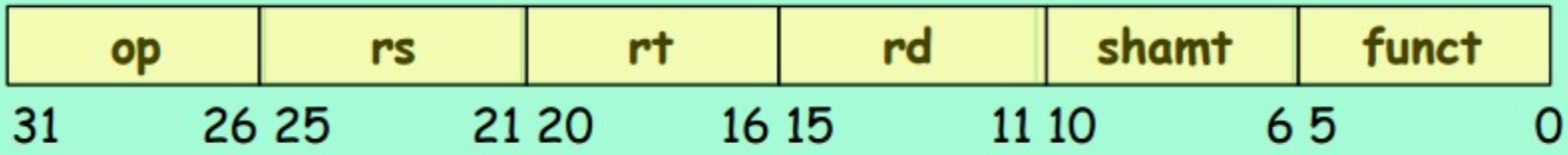
RISC Instruction Set Basics (MIPS)
Types of Instructions

Jump Format (J-type)

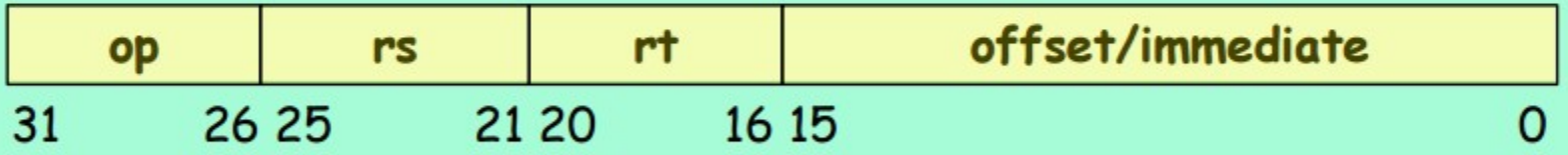
- Conditional branches are transfers of control. As described before, a branch causes an immediate value to be added to the current program counter.

R-, I, and J-type format comparison

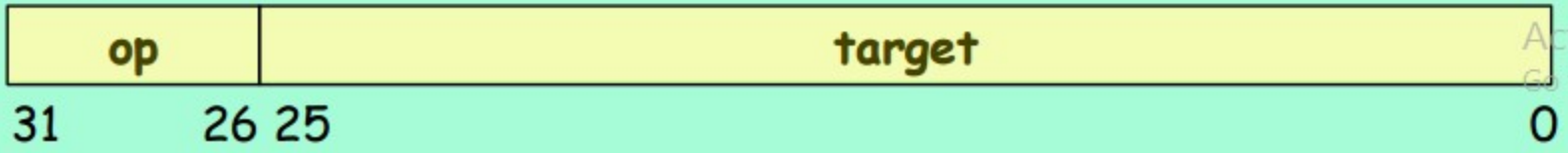
R-type:



I-type:



J-type:

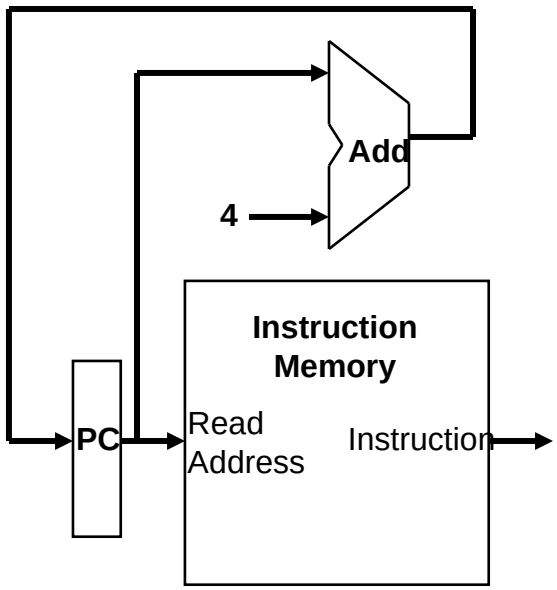


RISC Instruction Set Implementation

- We first need to look at how instructions in the MIPS instruction set are implemented without pipelining. We'll assume that any instruction of the subset of MIPS can be executed in at most 5 clock cycles.
- The five clock cycles will be broken up into the following steps:
    - ***Instruction Fetch Cycle***
    - ***Instruction Decode/Register Fetch Cycle***
    - ***Execution Cycle***
    - ***Memory Access Cycle***
    -

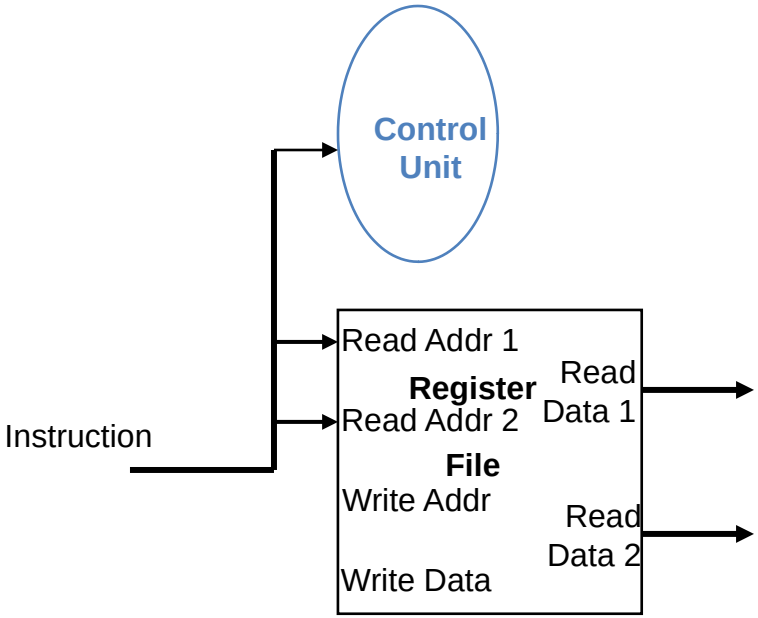
Fetching Instructions (IF)

- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC to hold the address of the next instruction
  - PC is updated every cycle, so it does not need an explicit write control signal
  - Instruction Memory is read every cycle, so it doesn't need an explicit read control signal



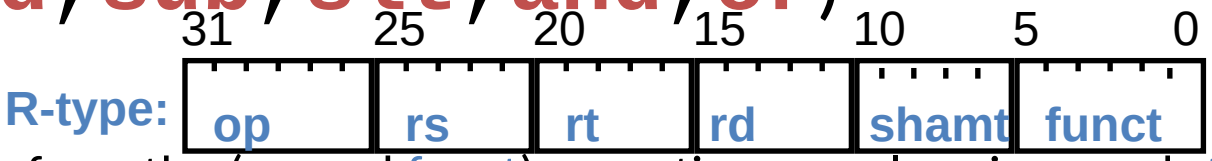
Decoding Instructions (ID)

- Decoding instructions involves
  - sending the fetched instruction's opcode and function field bits to the control unit
  - reading two values from the Register File
    - Register File addresses are contained in the instruction

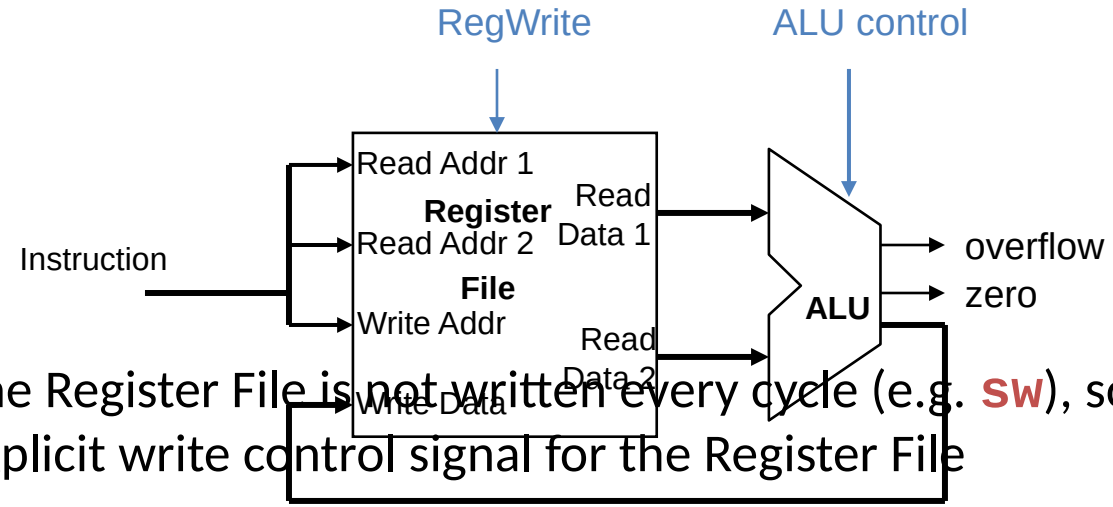


Executing R Format Operations (IE)

- R format operations (`add`, `sub`, `slt`, `and`, `or`)



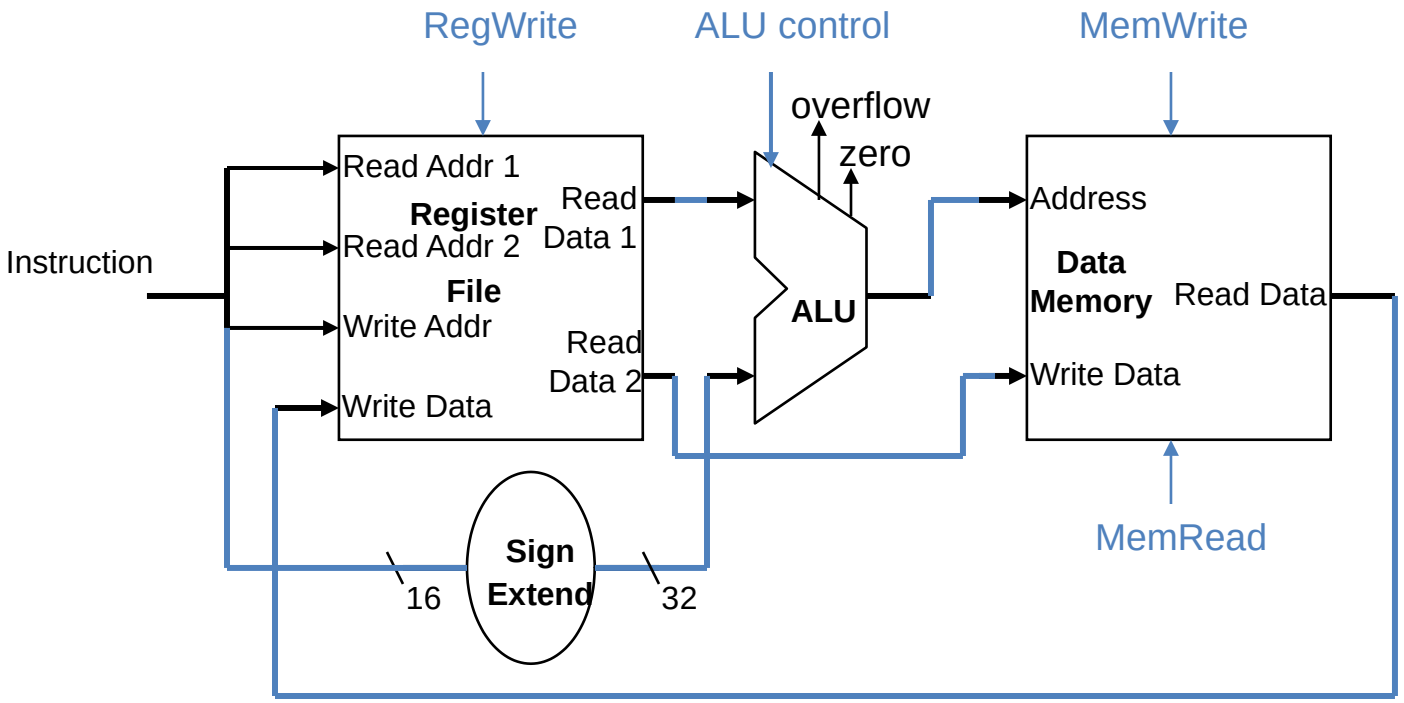
  - perform the (op and funct) operation on values in rs and rt
  - store the result back into the Register File (into location rd)



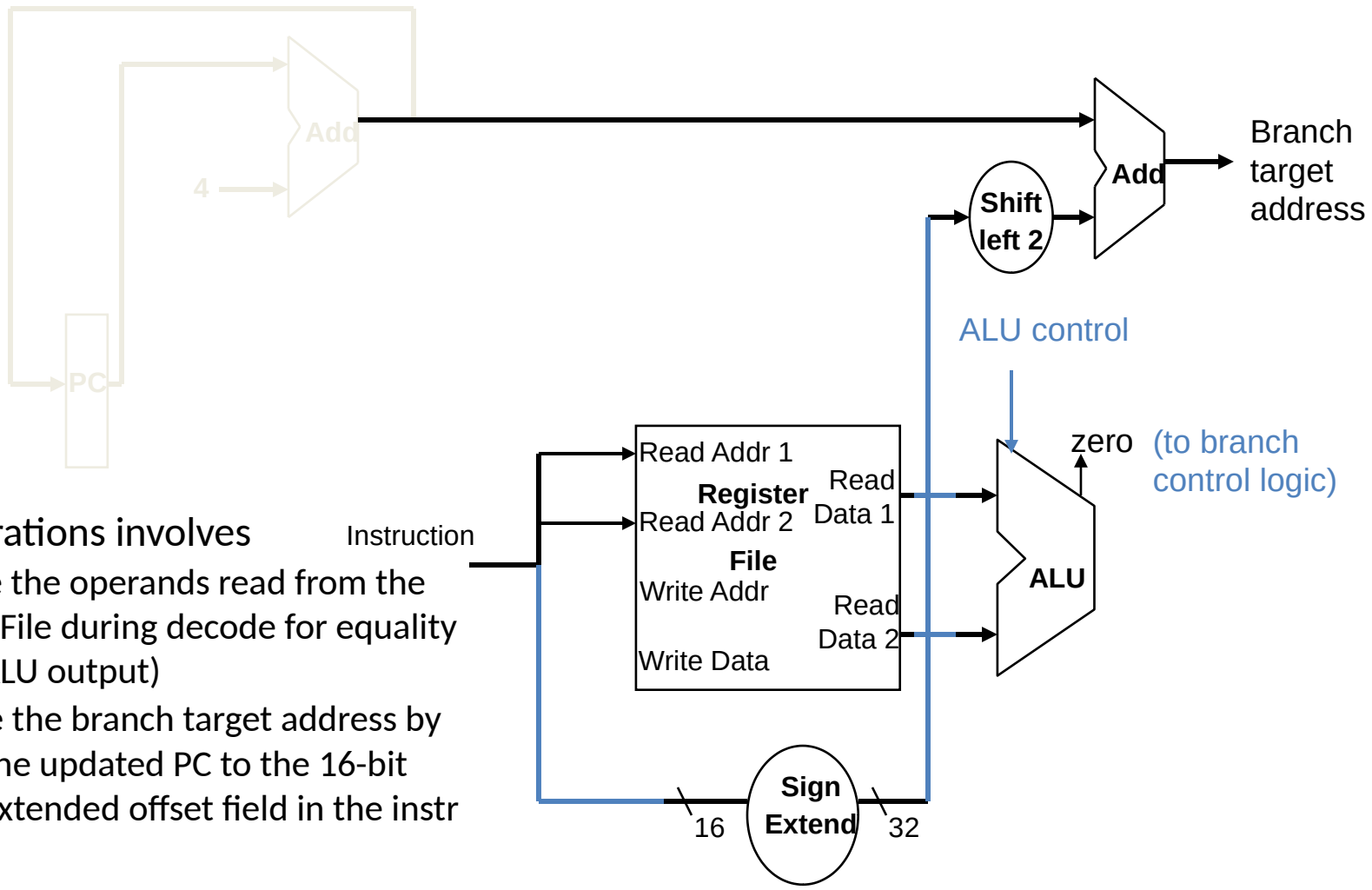
  - The Register File is not written every cycle (e.g. `sw`), so we need an explicit write control signal for the Register File

Executing Load and Store Operations (IE)

- **Load and store operations** involve
  - compute memory address by adding the base register (read from the Register File during decode) to the 16-bit signed-extended offset field in the instruction
  - store value (read from the Register File during decode) written to the Data Memory
  - load value, read from the Data Memory, written to the Register File



Executing Branch Operations (IE)



- Branch operations involves
  - compare the operands read from the Register File during decode for equality (`zero` ALU output)
  - compute the branch target address by adding the updated PC to the 16-bit signed-extended offset field in the instr

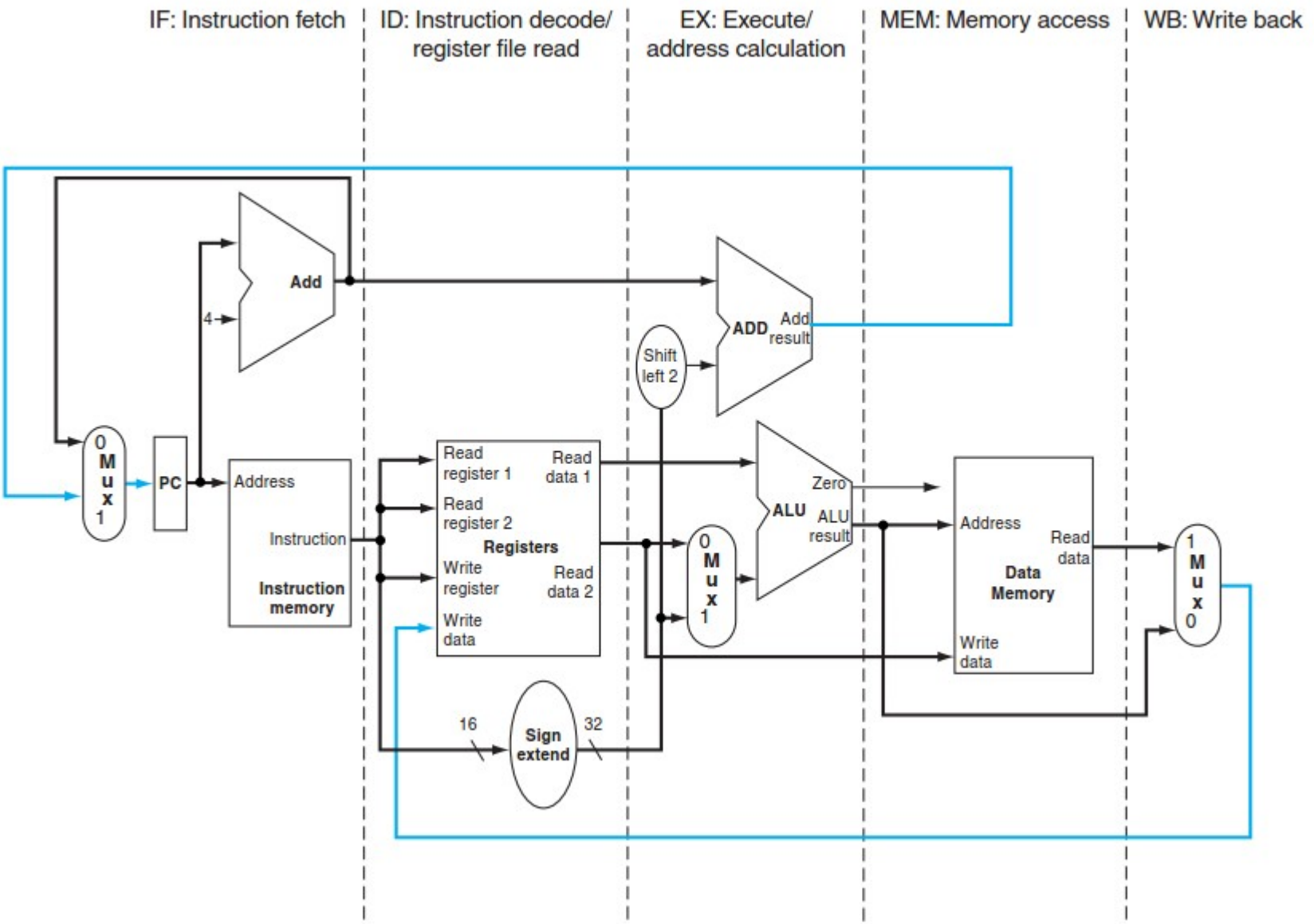
Memory Access (MEM) Cycle

- If a **load**, the effective address computed from the previous cycle is referenced and the memory is read. **The actual data transfer to the register does not occur until the next cycle.**
- If a store, the data from the register is written to the effective address in memory.

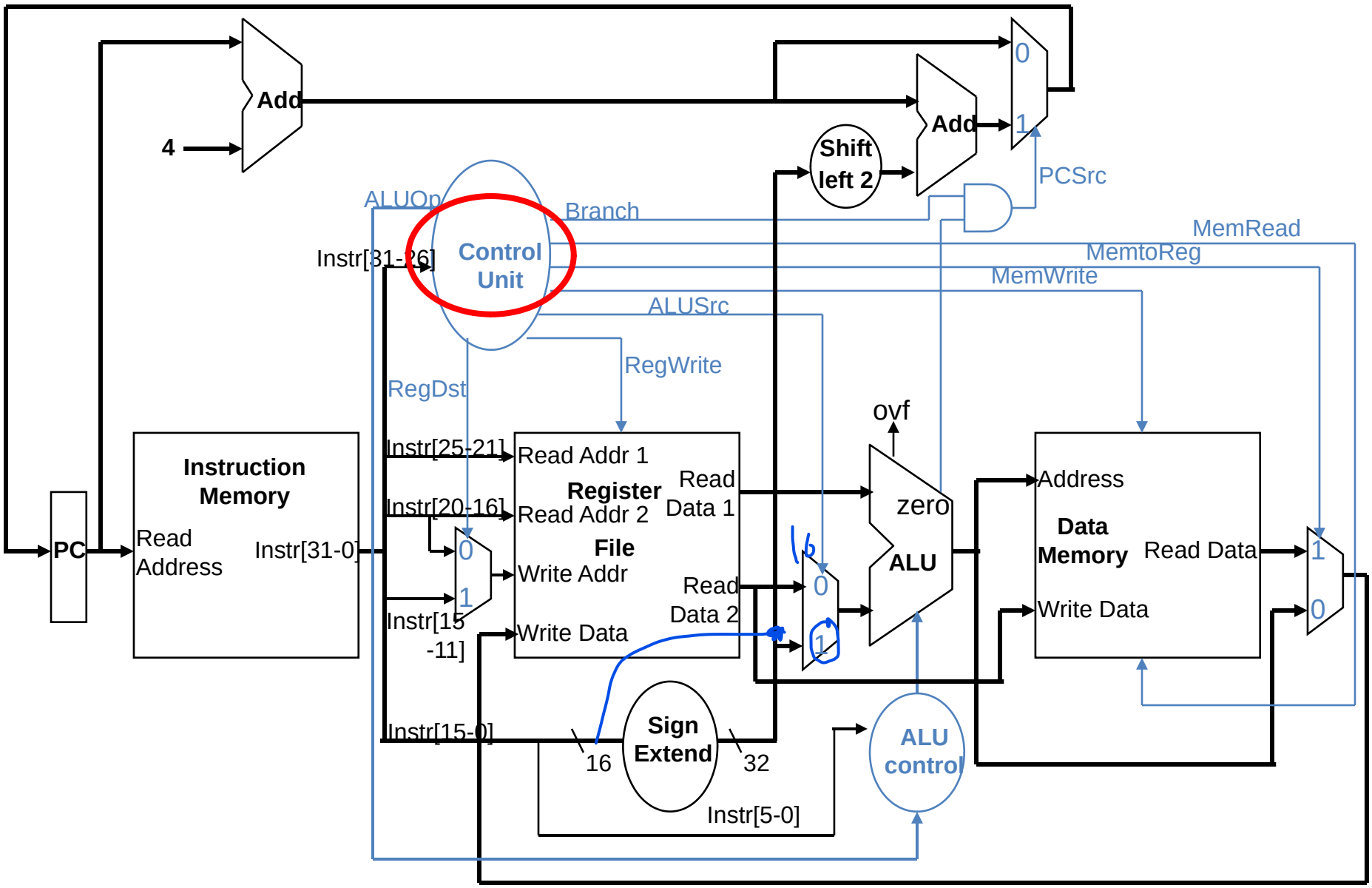
Write-Back (WB) Cycle

- Occurs with **Register-Register ALU** instructions or **load instructions**.
- Simple operation whether the operation is a register-register operation or a memory load operation, the resulting data is written to the appropriate register.

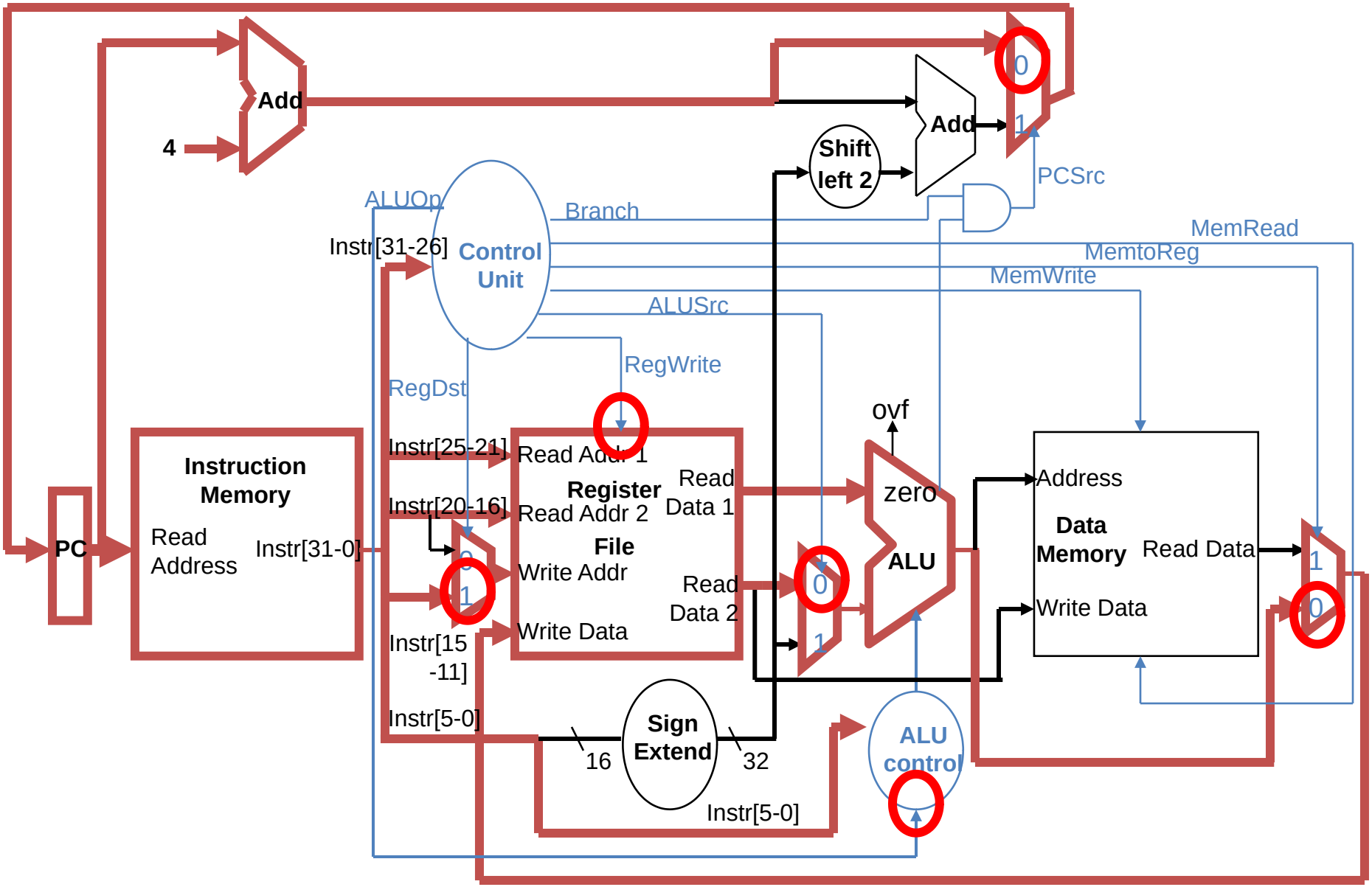
The single cycle datapath



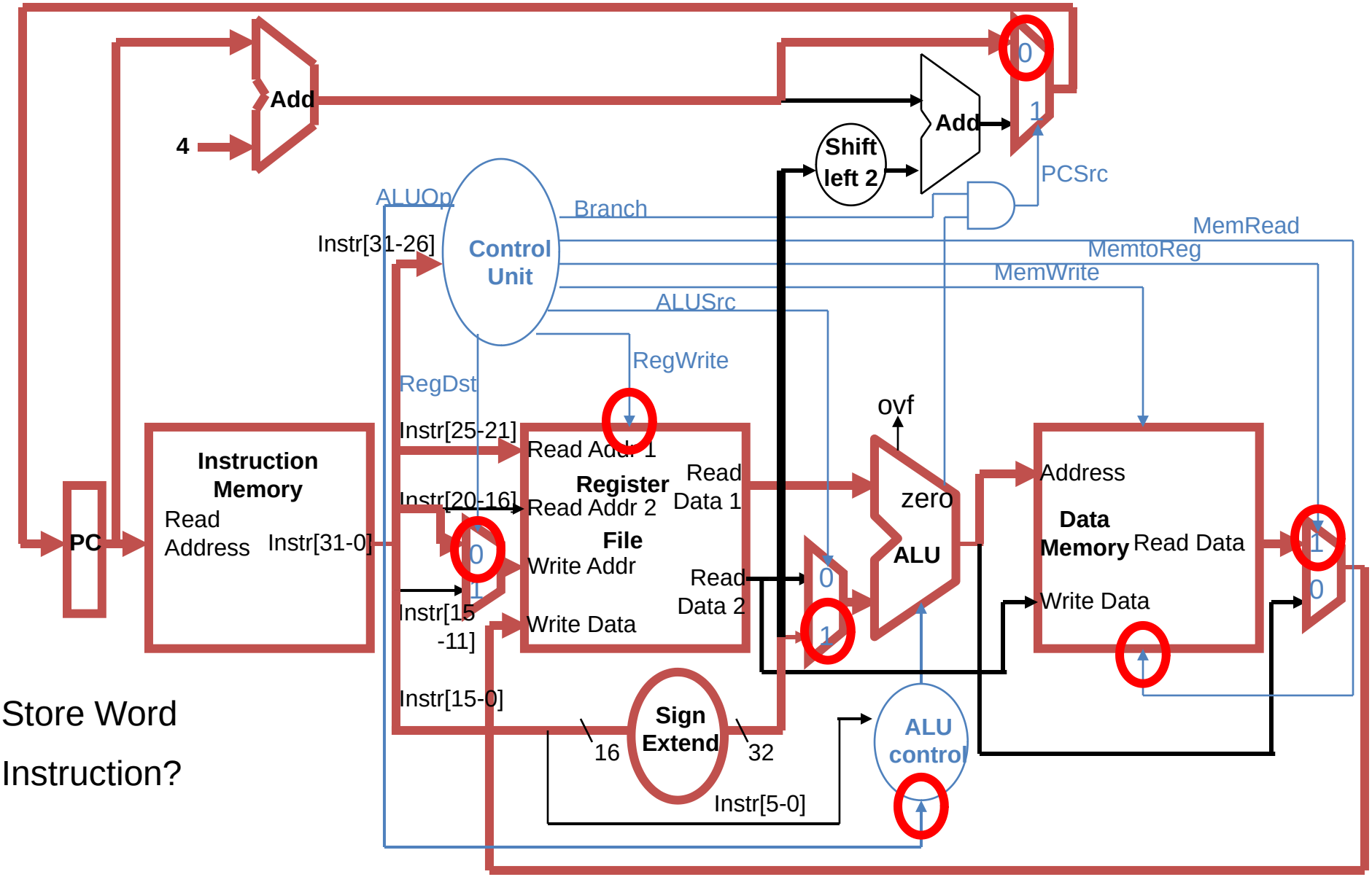
Single Cycle Datapath with Control Unit



R-type Instruction Data/Control Flow

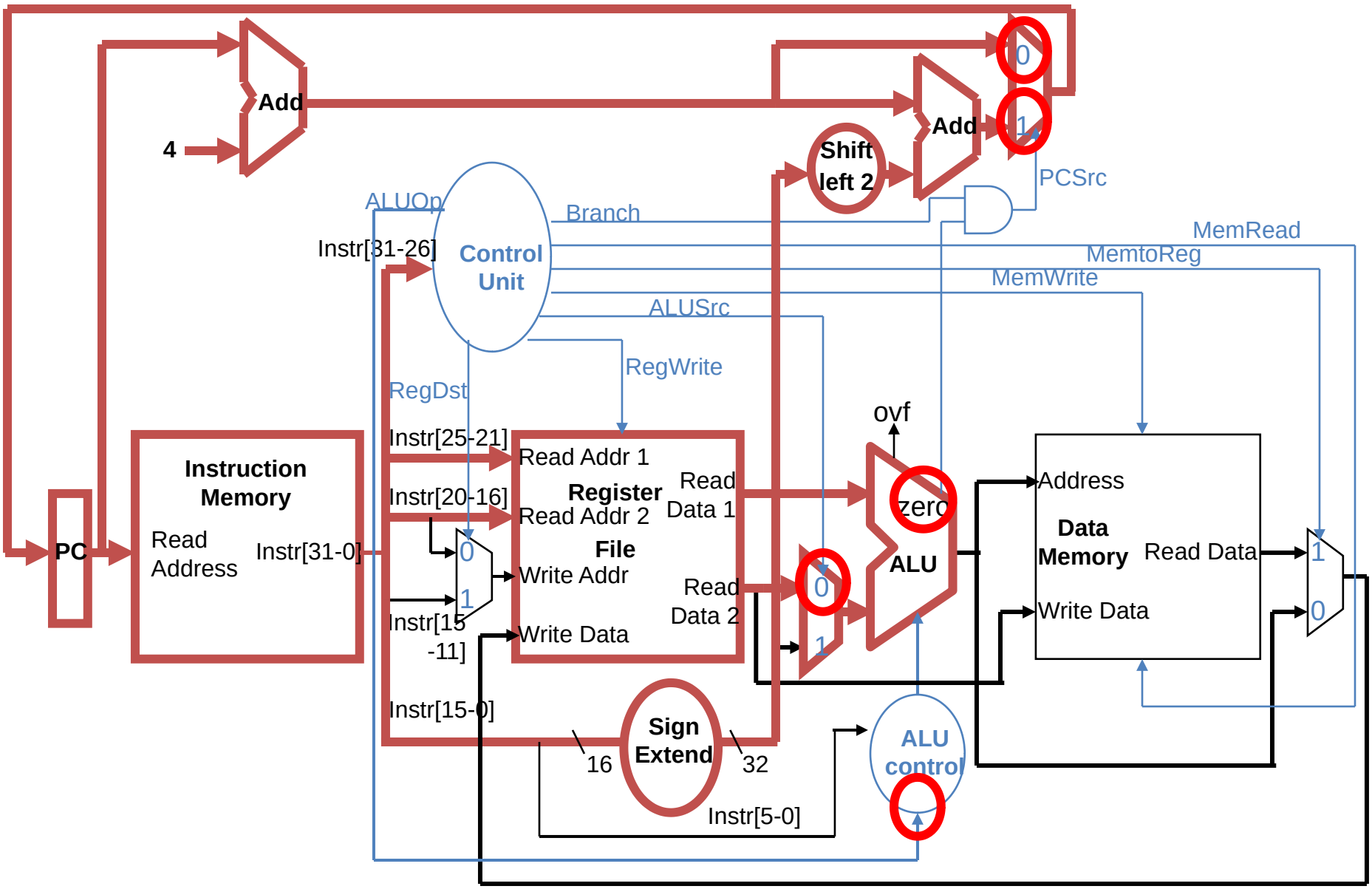


Load Word Instruction Data/Control Flow

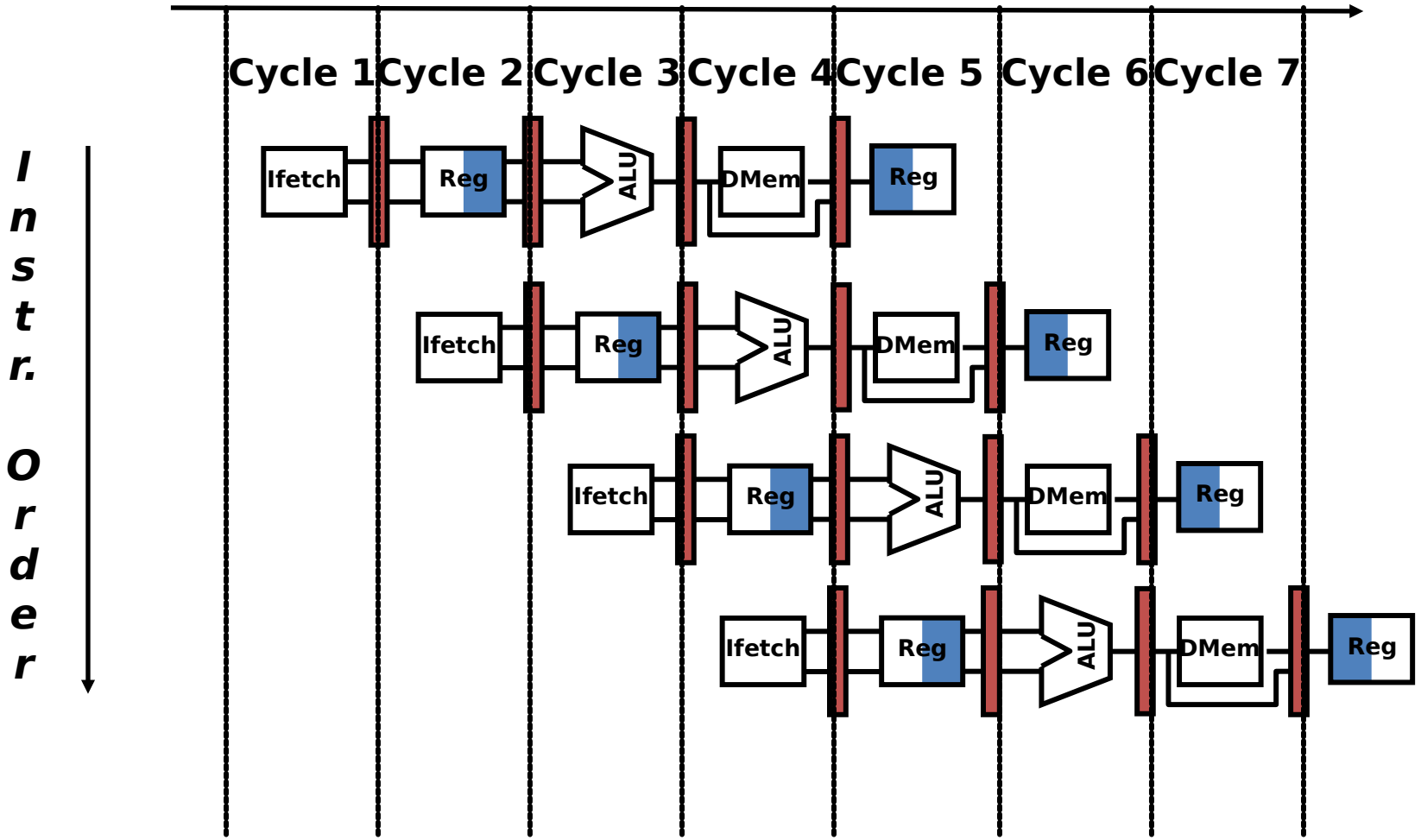


Store Word Instruction?

Branch Instruction Data/Control Flow

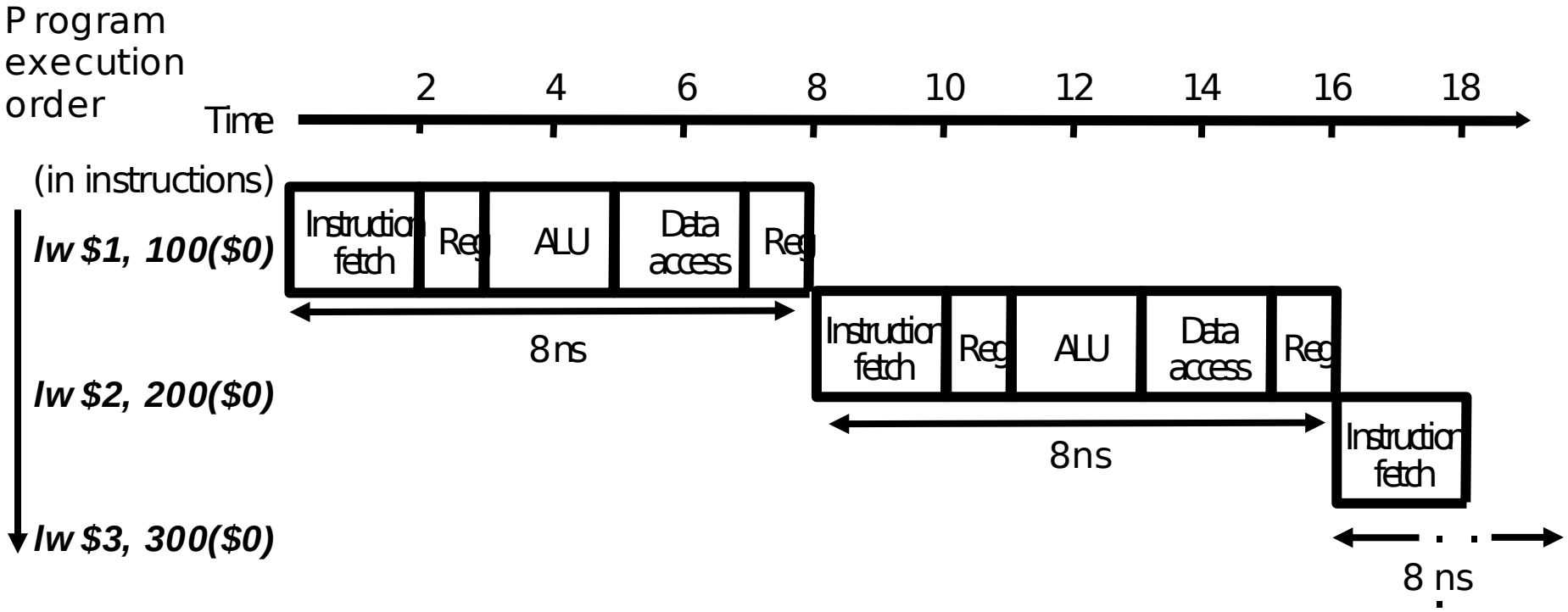


The Basic Pipeline For MIPS



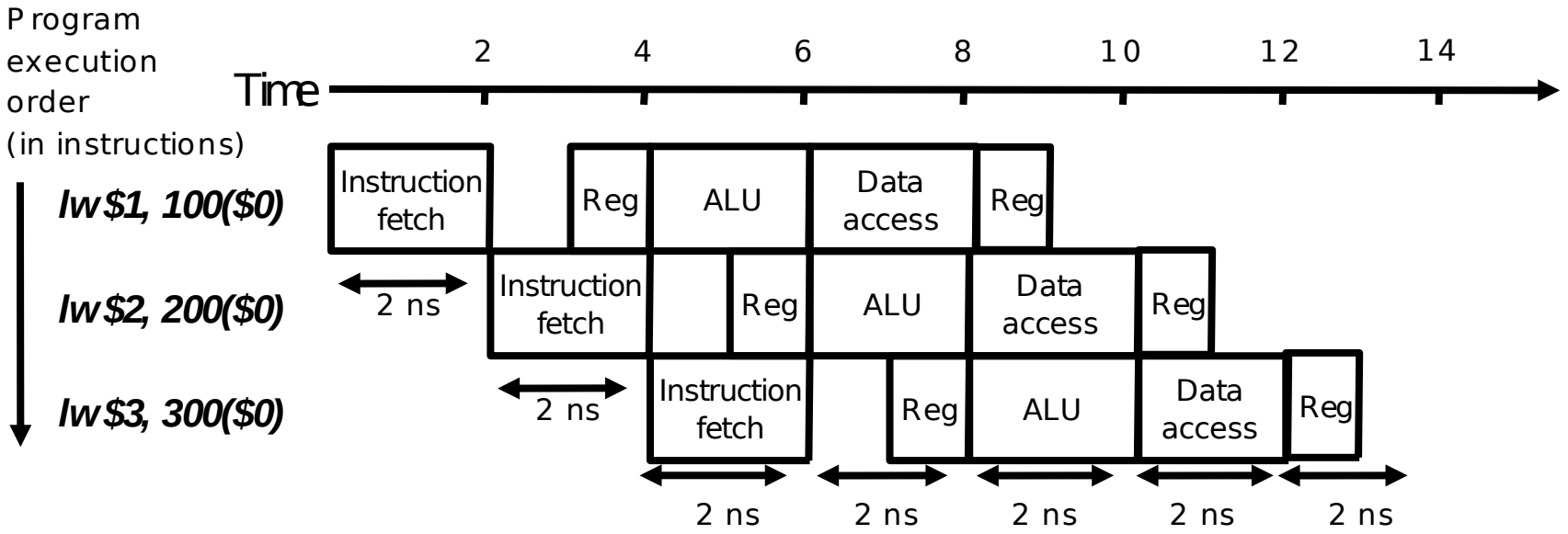
CPU Pipelining: Example

- Example : Single-Cycle, non-pipelined execution
  - Total time for 3 instructions: 24 ns



CPU Pipelining: Example

- Single-cycle, pipelined execution
  - Improve performance by increasing instruction throughput
  - Total time for 3 instructions = 14 ns
  - Each instruction adds 2 ns to total execution time
  - Stage time limited by slowest resource (2 ns)
  - Assumptions:
    - Write to register occurs in 1st half of clock
    - Read from register occurs in 2nd half of clock

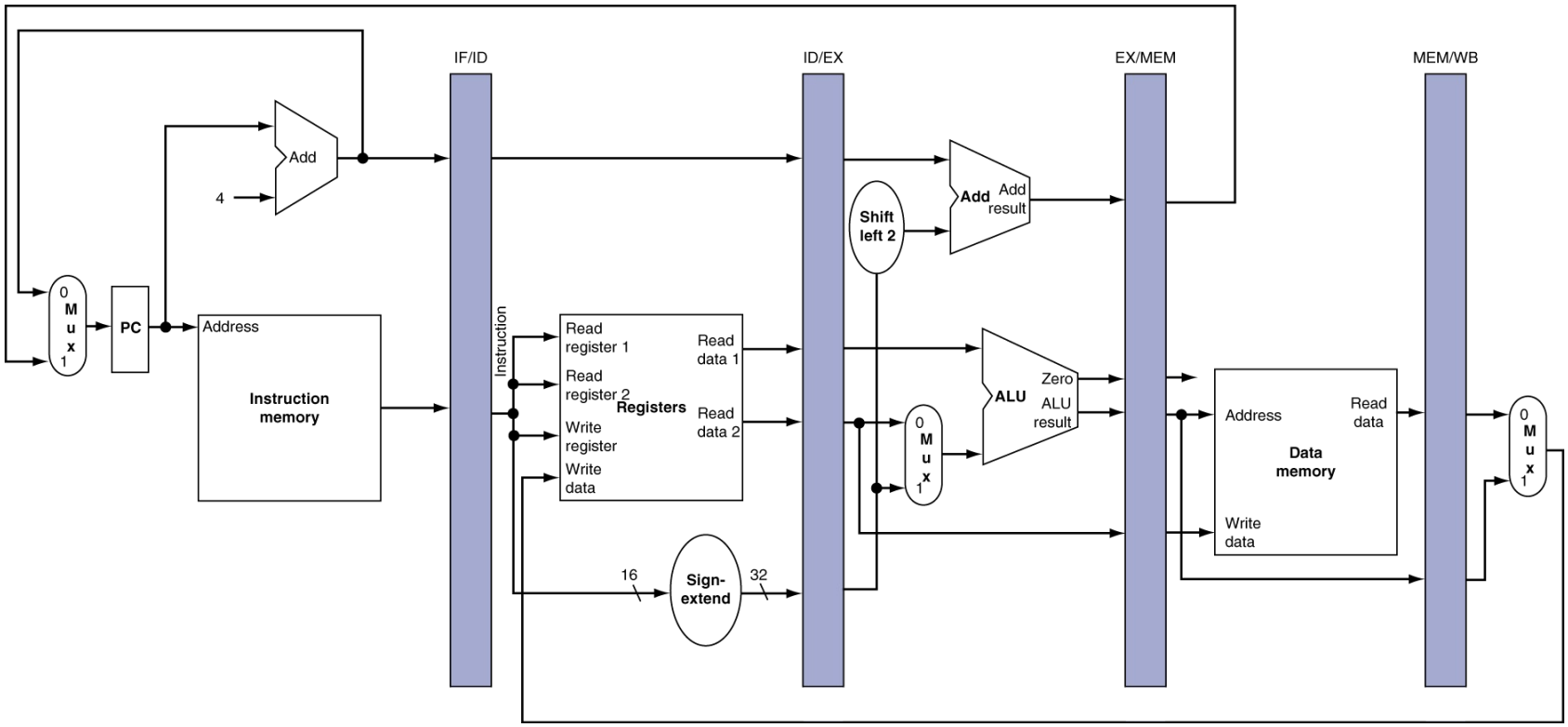


CPU pipelining: Example

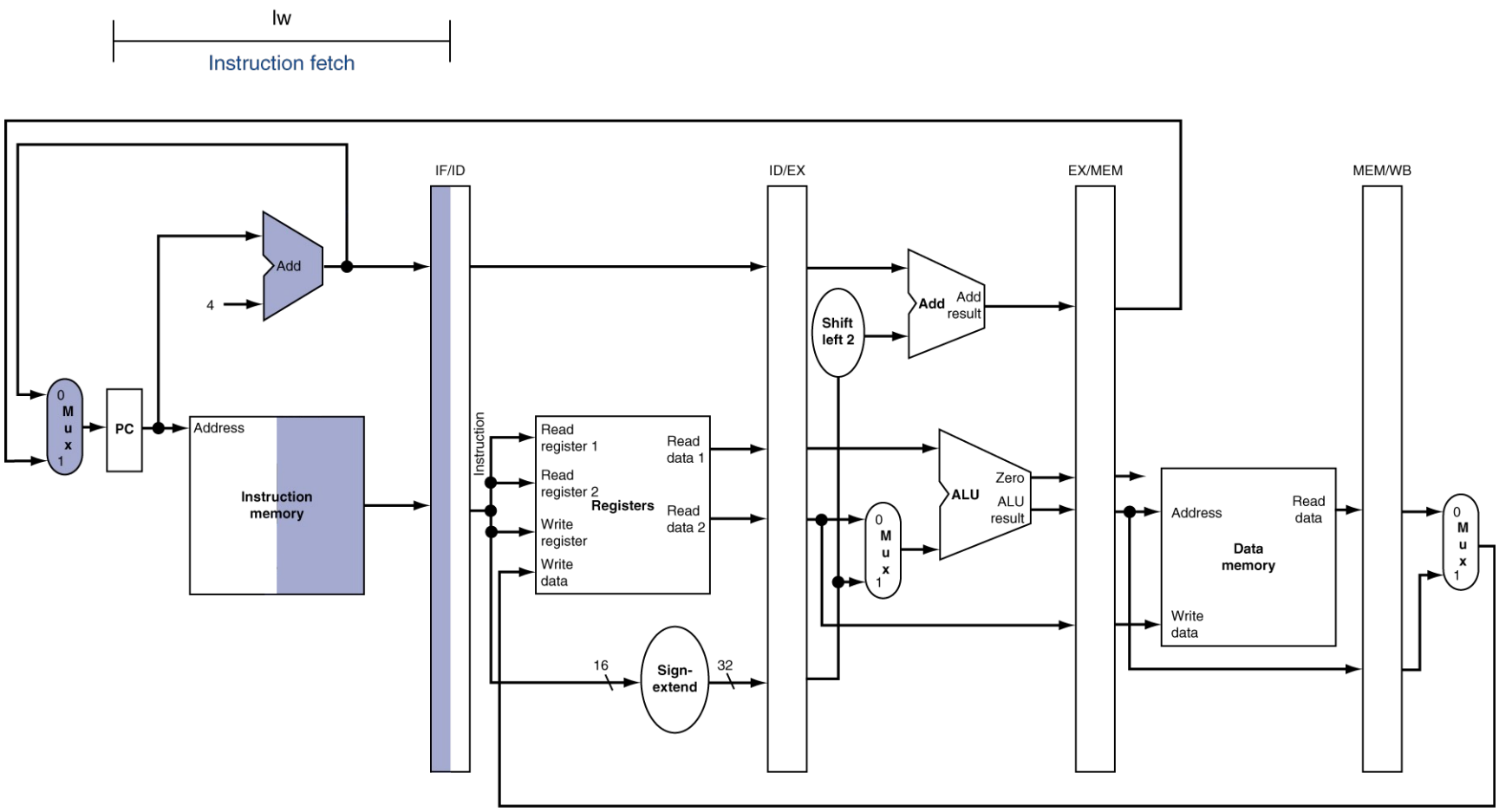
- Time without pipelining = 24 ns
- Time with pipelining = 14 ns (**not = 24/5**), WHY???
  - ***Number of instructions is not large***
- Let's increase the number of instructions
  - If number of instructions = 1,000,000 instruction , the total time with pipelining = 1,000,000 X 2 ns = 2,000,000 ns
  - Time without pipelining = 1,000,000 X 8ns = 8,000,000 ns
  - The speed up = 4 (increased)

The pipelined version of MIPS Datapath

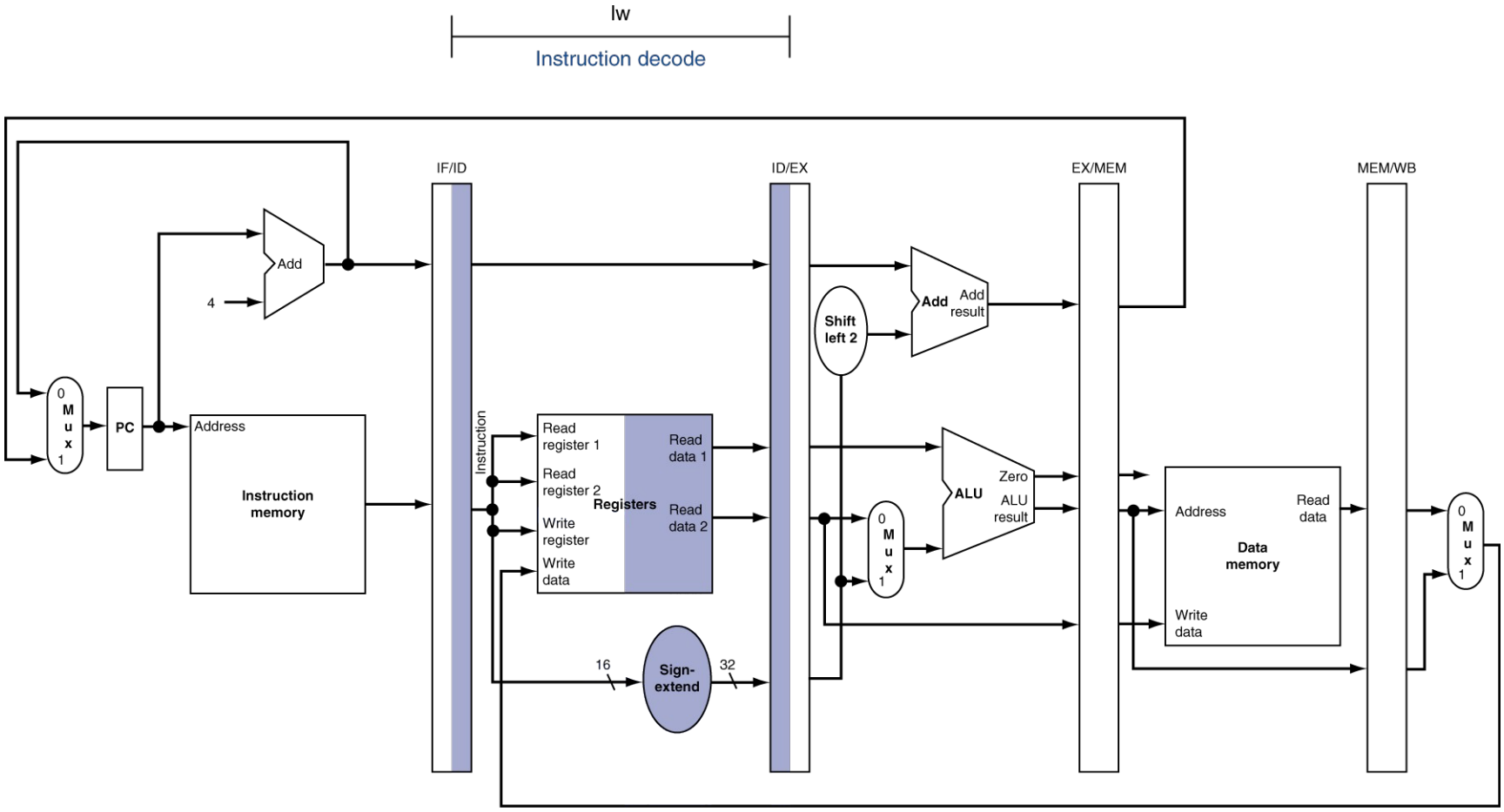
- Need registers between stages
  -



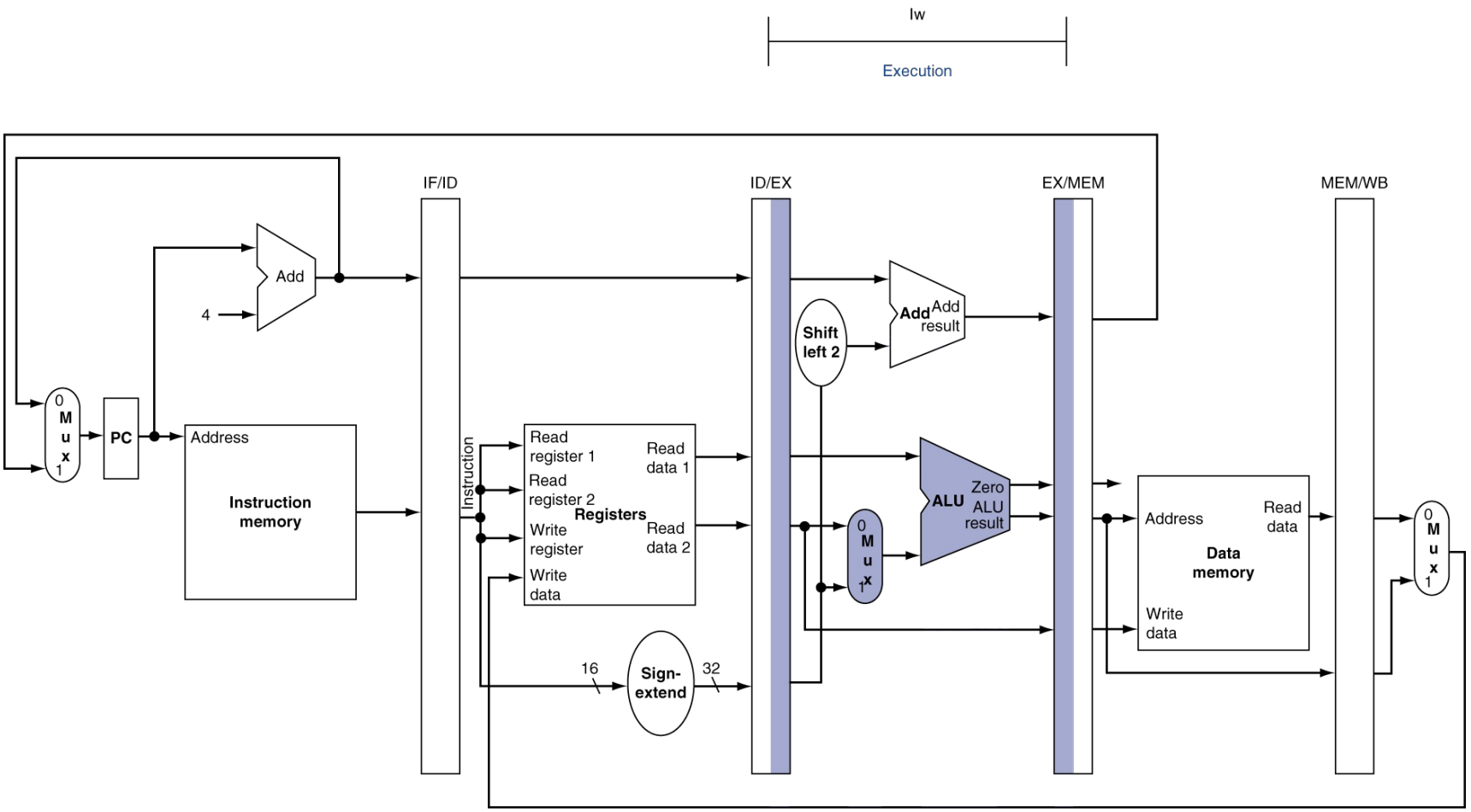
IF


lw
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

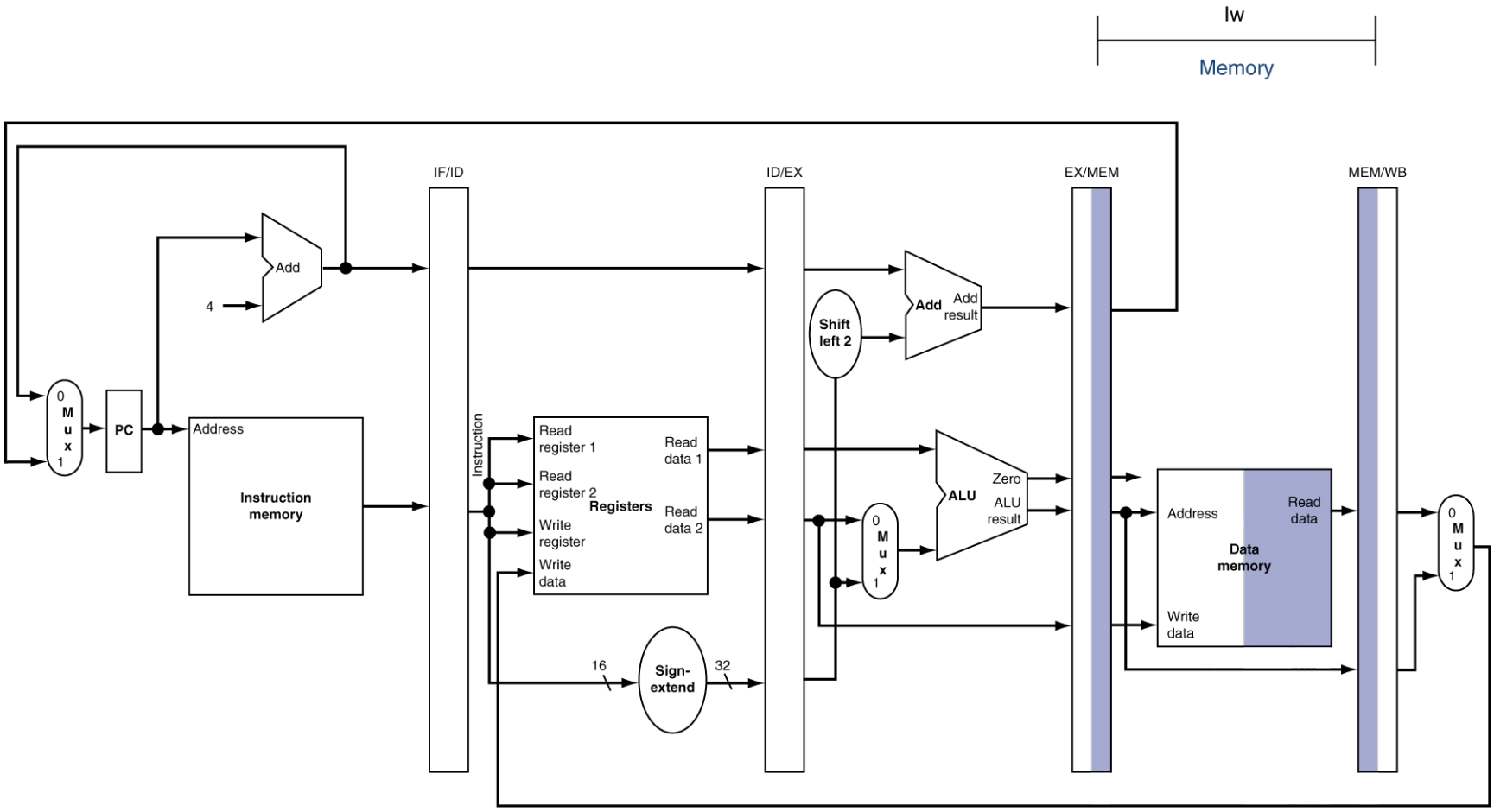
ID


lw
Instruction decode
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

EX for Load

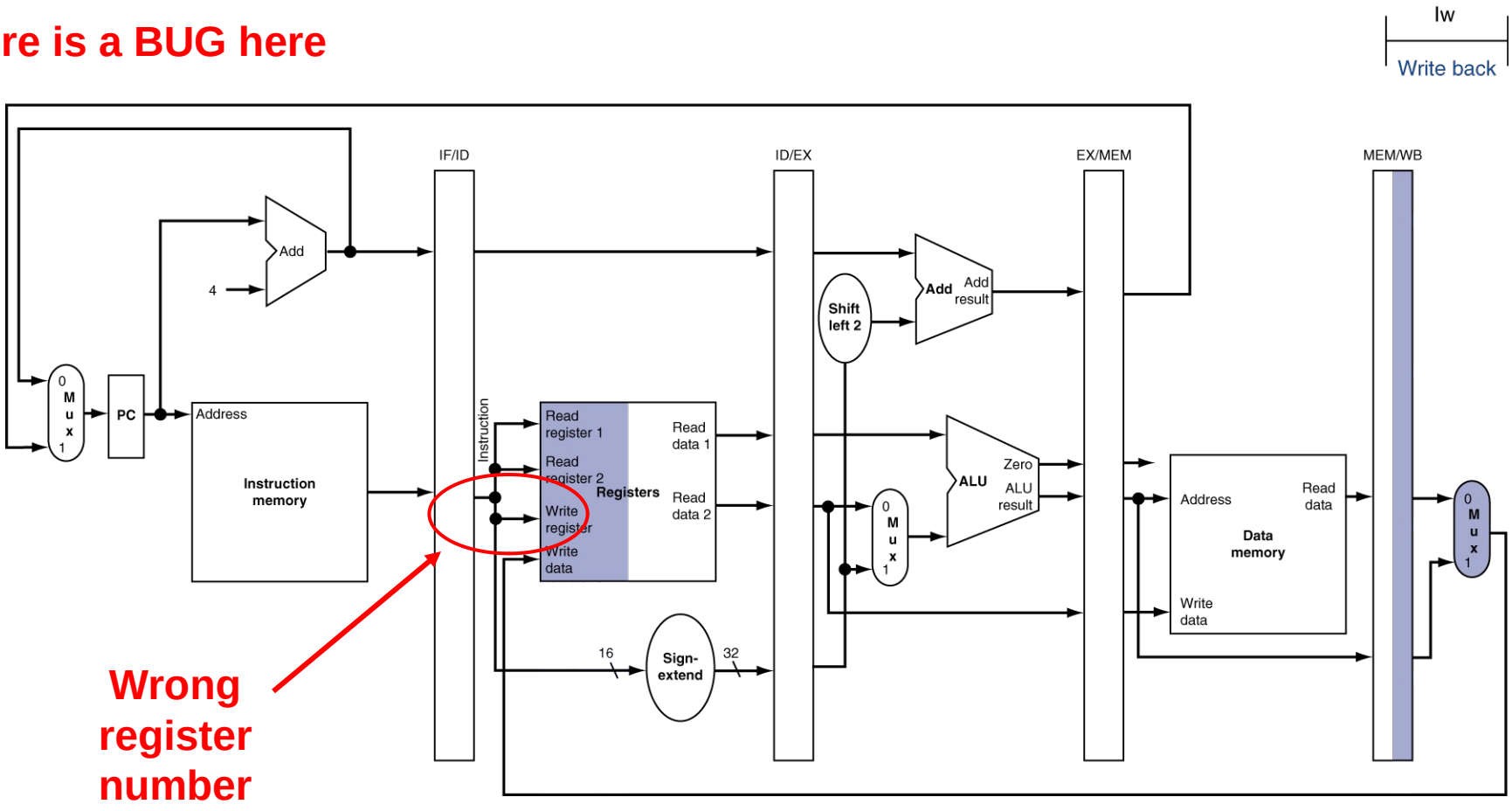

lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign-extend
32

MEM for Load


lw
Memory
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
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

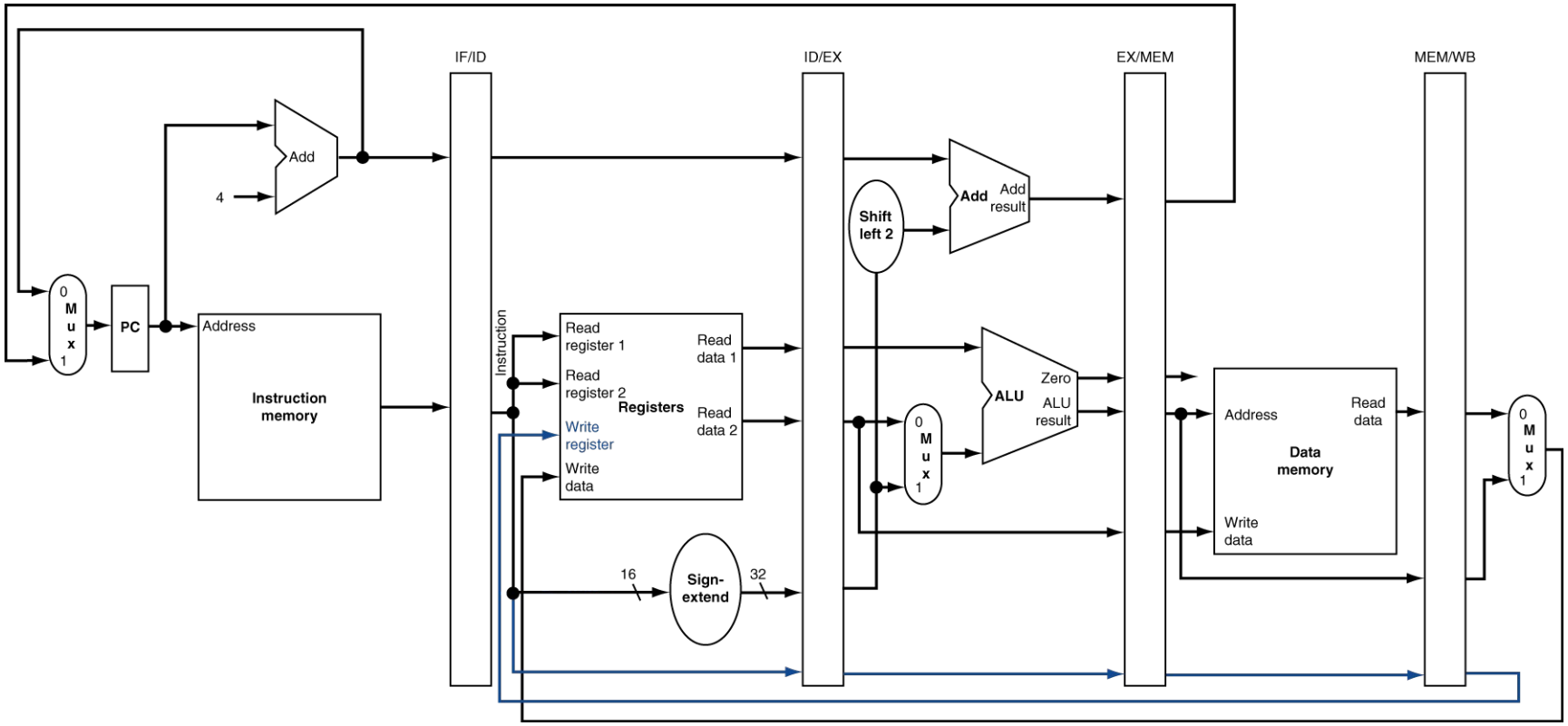
WB for Load

There is a BUG here

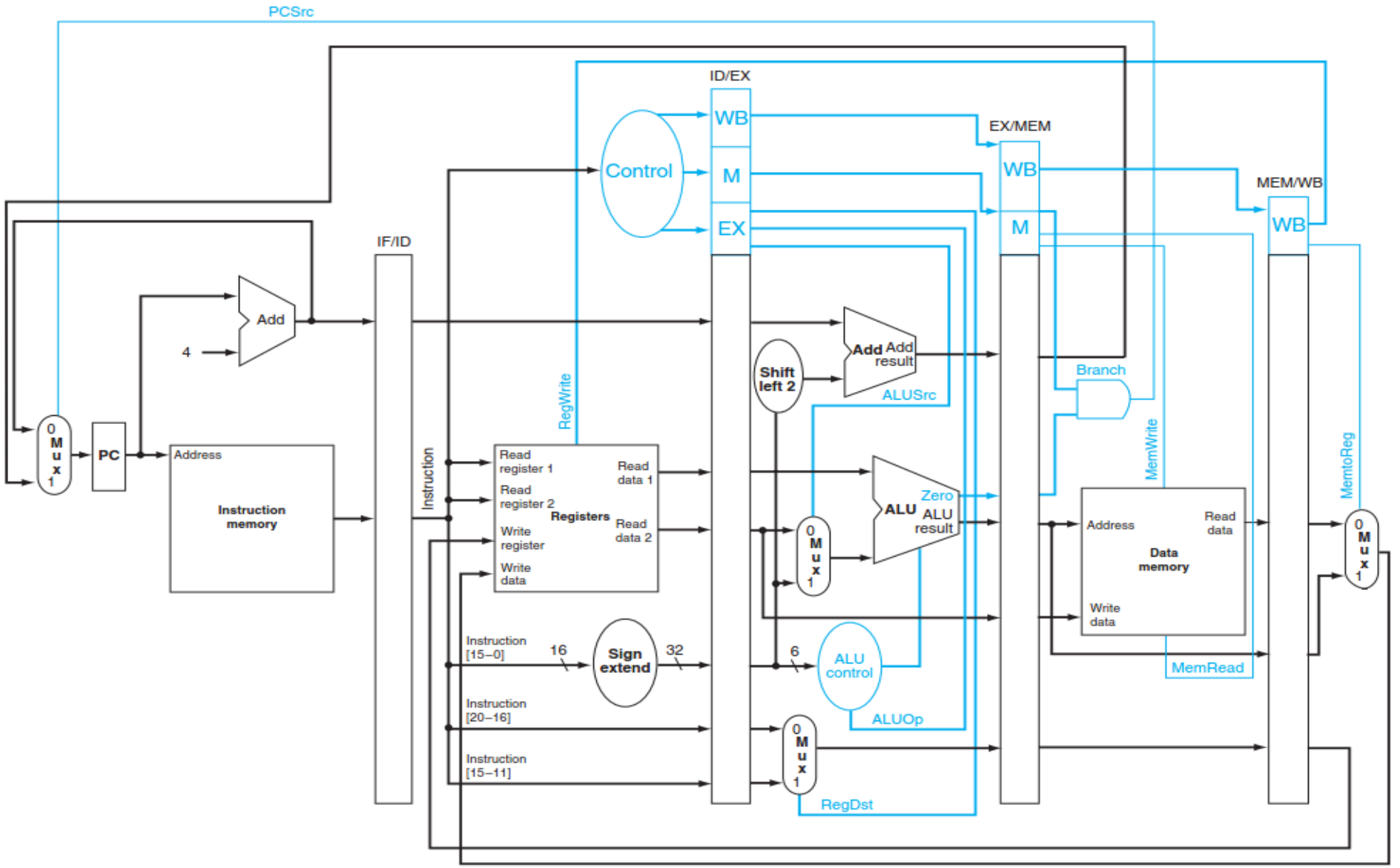


Wrong register number

Corrected Datapath for Load



The pipelined data path with control signals



Control Signals

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
R-format	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