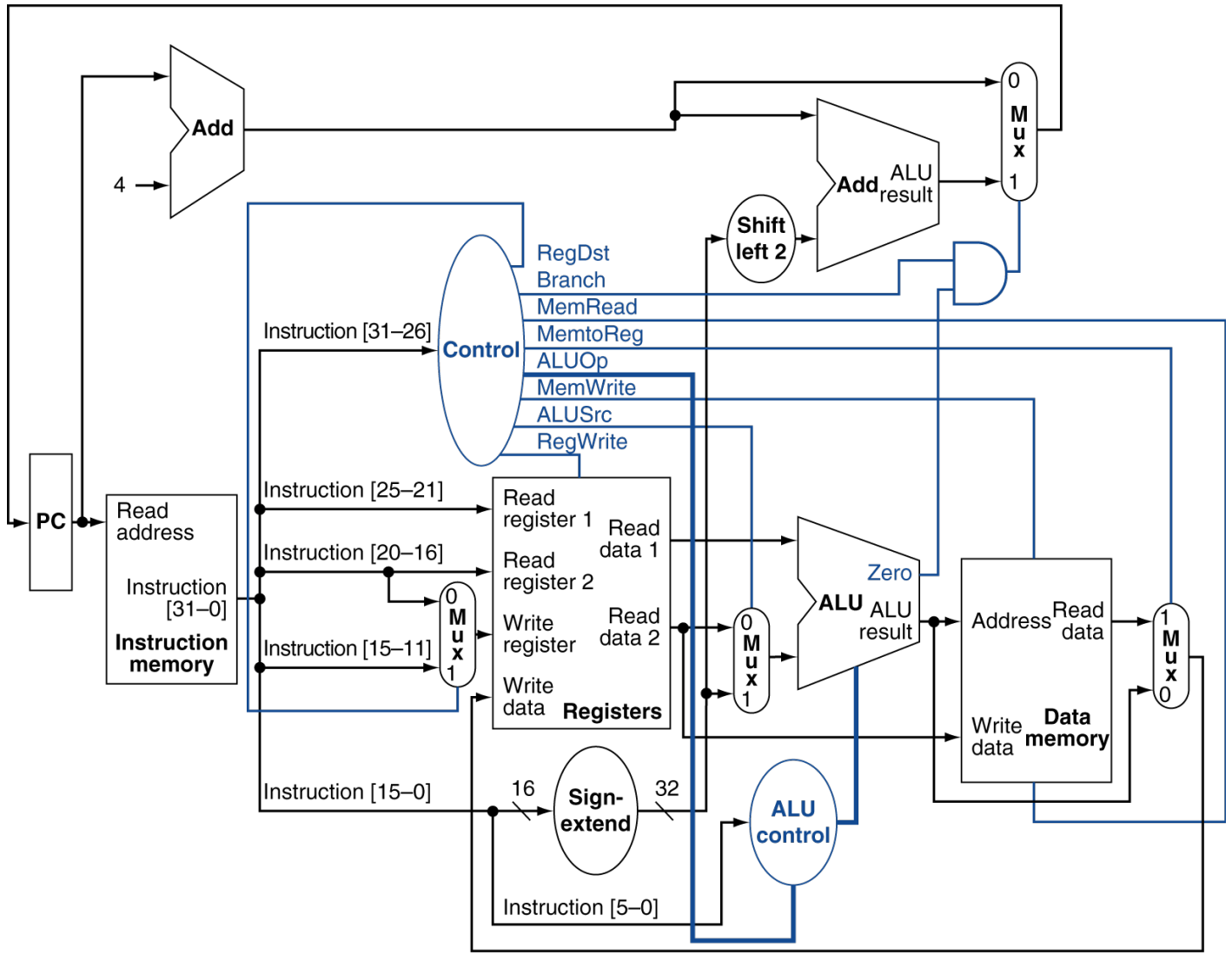
Inf2C - Computer Systems
Lecture 13-14
Processor Design – Multi-Cycle

Vijay Nagarajan

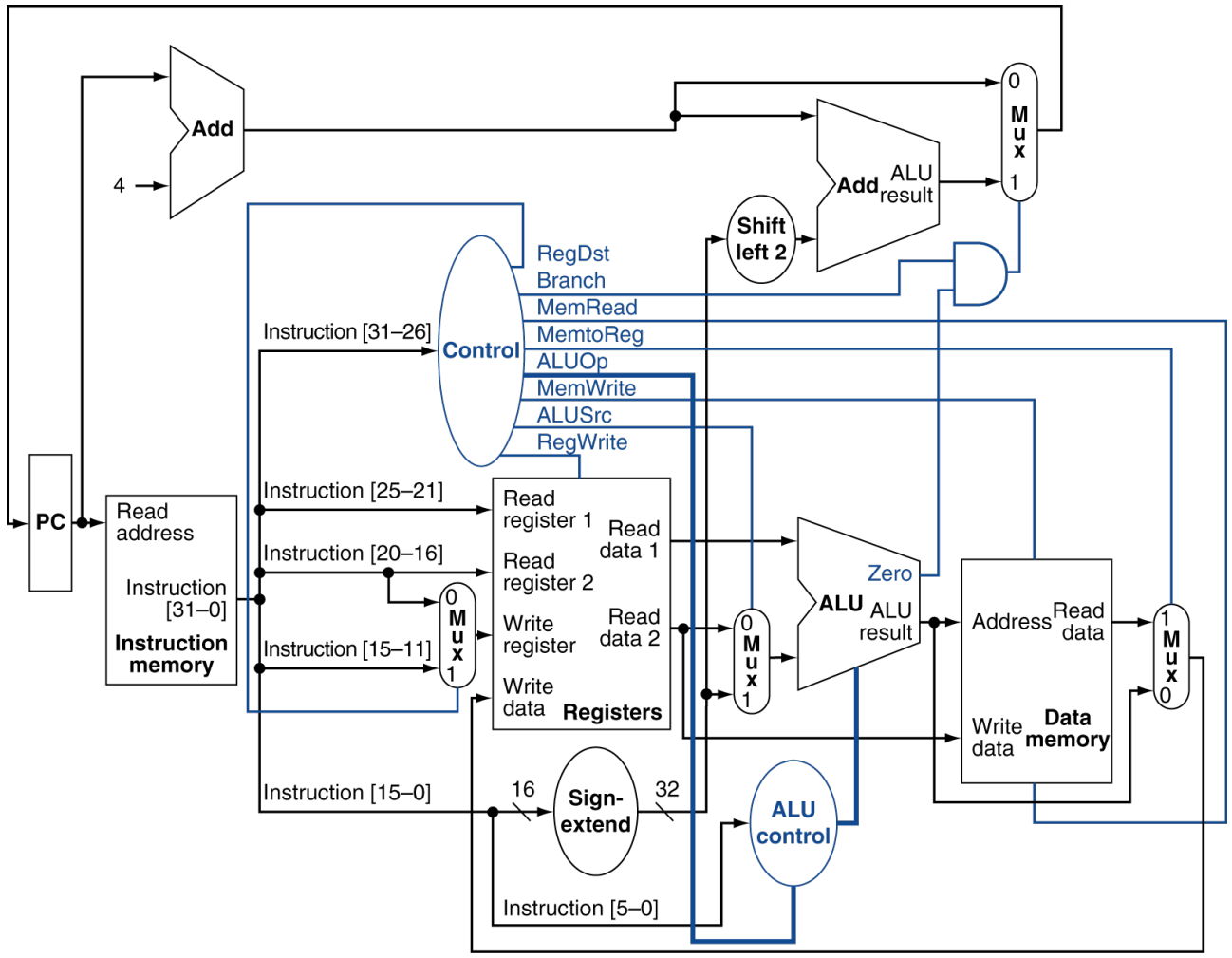
School of Informatics

University of Edinburgh

Previous lecture: single-cycle processor



Question: which instruction takes the most time to complete in a single-cycle processor?



Motivating a multi-cycle processor

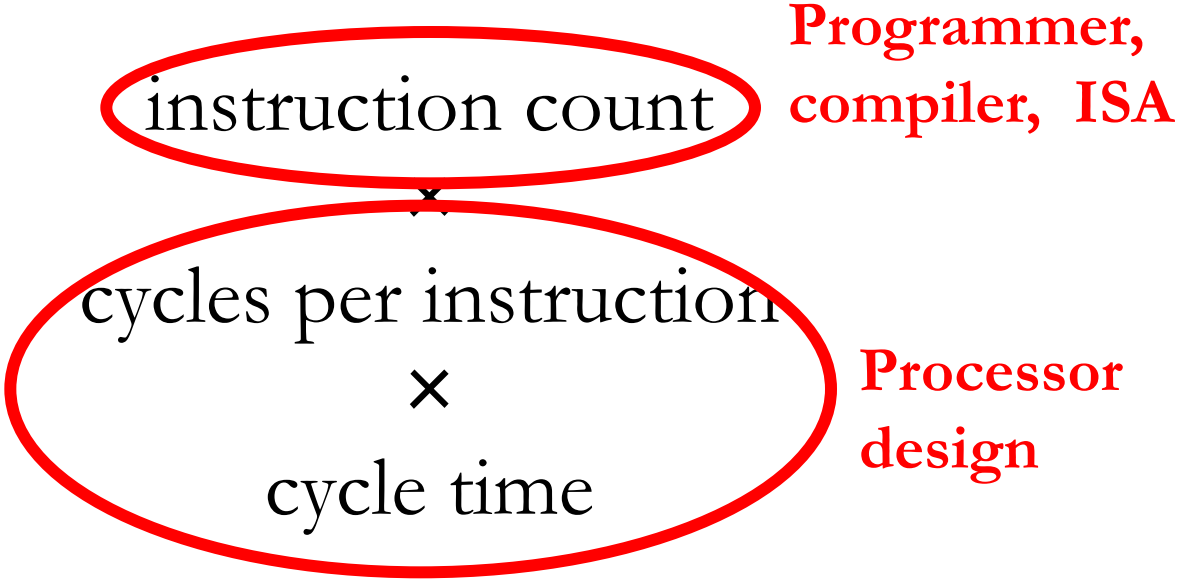
Aren't single cycle processors good enough?

No!

- Speed: cycle time must be long enough for the most complex instruction to complete
  - But the average instruction needs less time
- Cost: functional units (e.g. adders) cannot be re-used within one instruction's execution

Measuring processor speed

Execution time is:



Multi-cycle processor: Basic idea

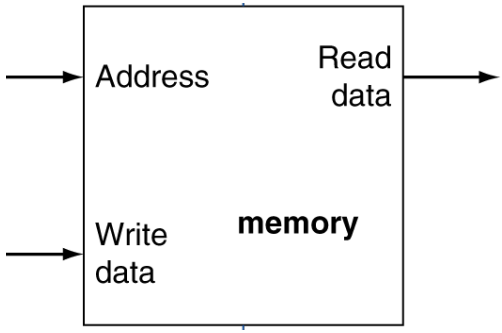
- Break up the execution of each instruction into multiple cycles
- Ensure that the actions performed within each cycle are "generic" – i.e., common to many instructions
- Reuse a common set of datapath and control components across cycles

End result:

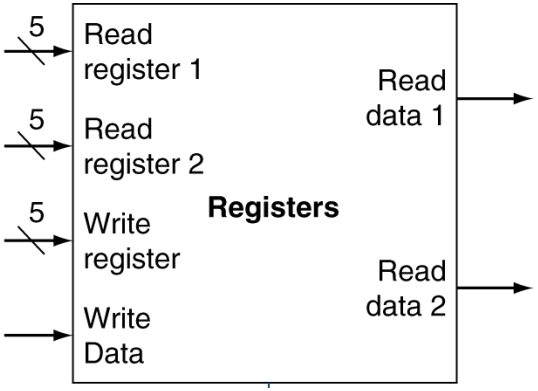
no instruction takes more time than necessary

Multi-cycle processor: Datapath building blocks

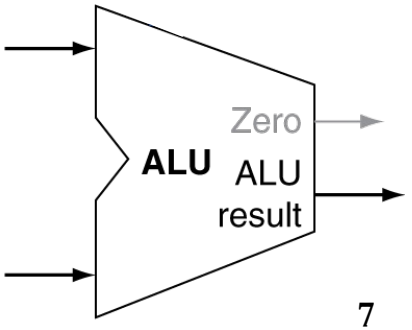
- One memory
  - Shared between instructions and data
  - Common interface



- Registers
  - Read early in instruction execution
  - Written late (if at all) in inst. execution



- One ALU
  - All PC calculations (conditional & unconditional)
  - All arithmetic (inc. branch condition evaluation)



Multi-cycle processor: Basic operation

- Each instruction takes multiple cycles to execute
  - Each cycle, at least one basic function performed (e.g., Fetch or Read Registers)
  - Multiple functions can be performed in one cycle if they use different functional units
    - E.g., Fetch → memory, PC+4 → ALU
- Example execution: SW

  cycle 1: Fetch (access memory)

  cycle 2: Read registers

  cycle 3: Compute address (use ALU)

  cycle 4: Perform store (access memory)

Same memory, accessed in different cycles

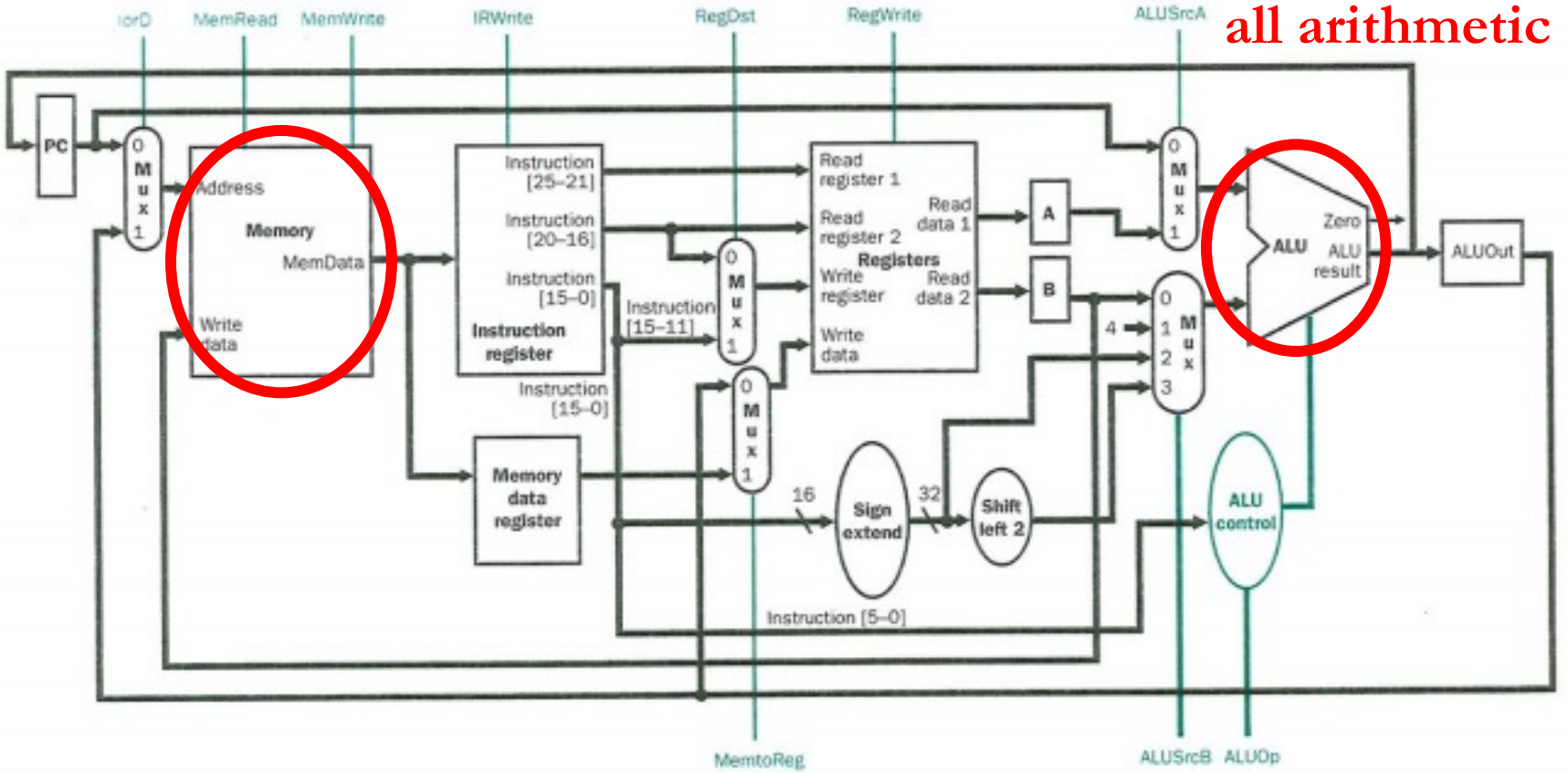
Multi-cycle processor: Details

- Cycle time determined by the propagation delay through the slowest functional unit
- Number of cycles varies for different instructions
- At the end of each cycle, data required in subsequent cycles must be stored somewhere
  - Data used by subsequent instructions stored in memory and registers → same as single-cycle processor
  - Data for the current instruction stored in special registers not visible to the programmer → NEW!

Multi-cycle datapath (overview)

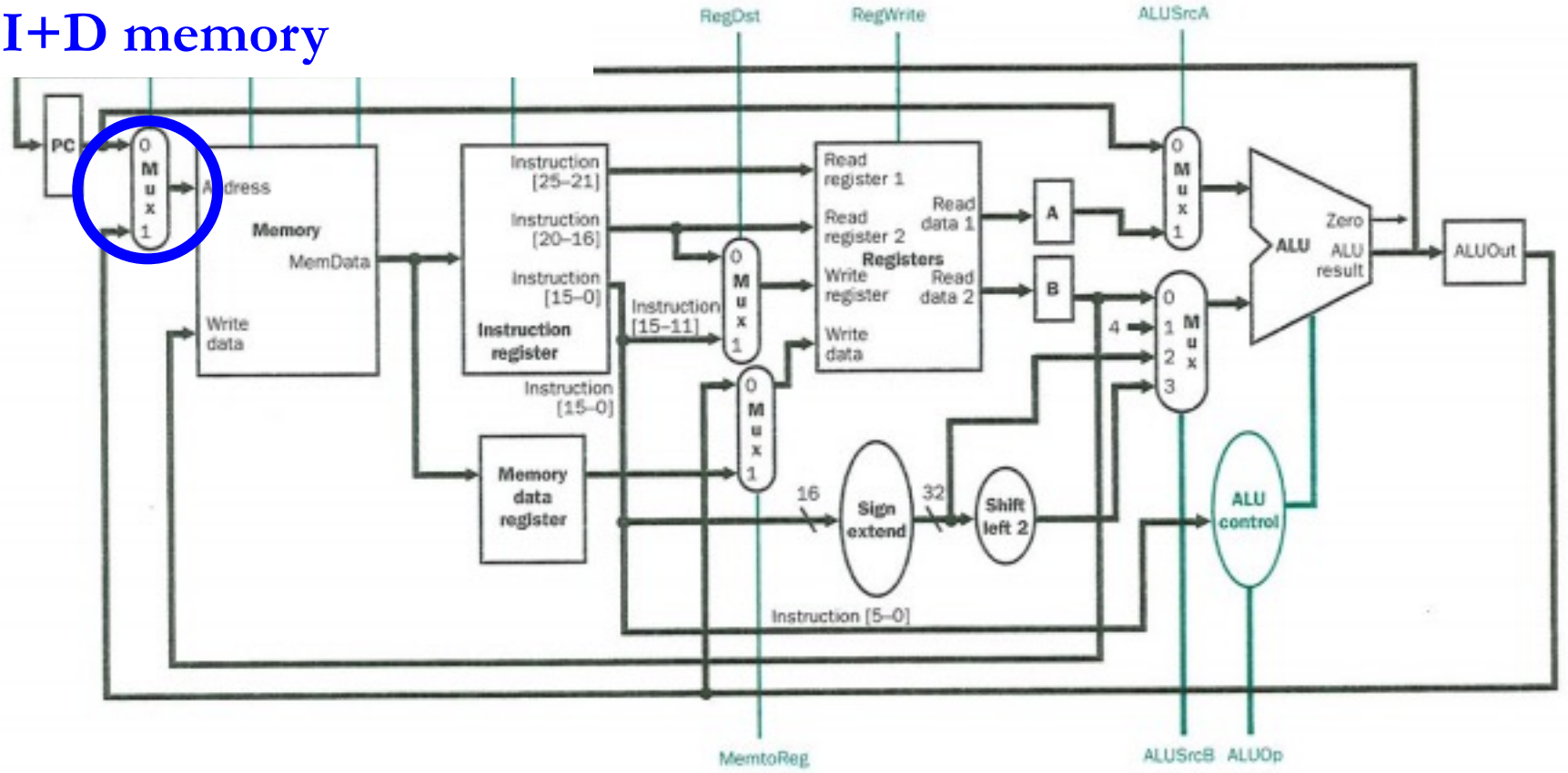
One memory for both Instruction & Data

One ALU for all arithmetic



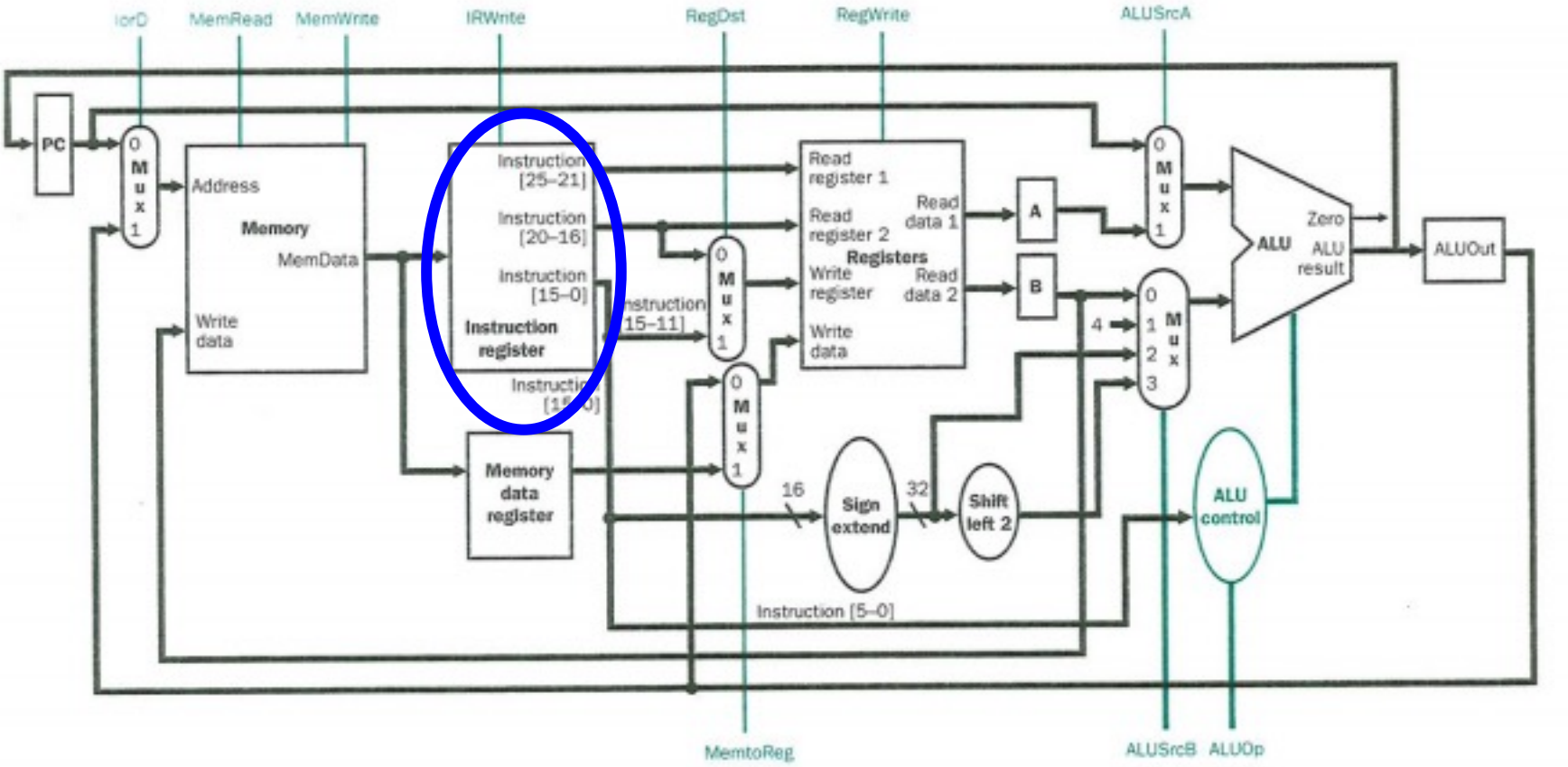
Multi-cycle datapath (overview)

Select address input to I+D memory



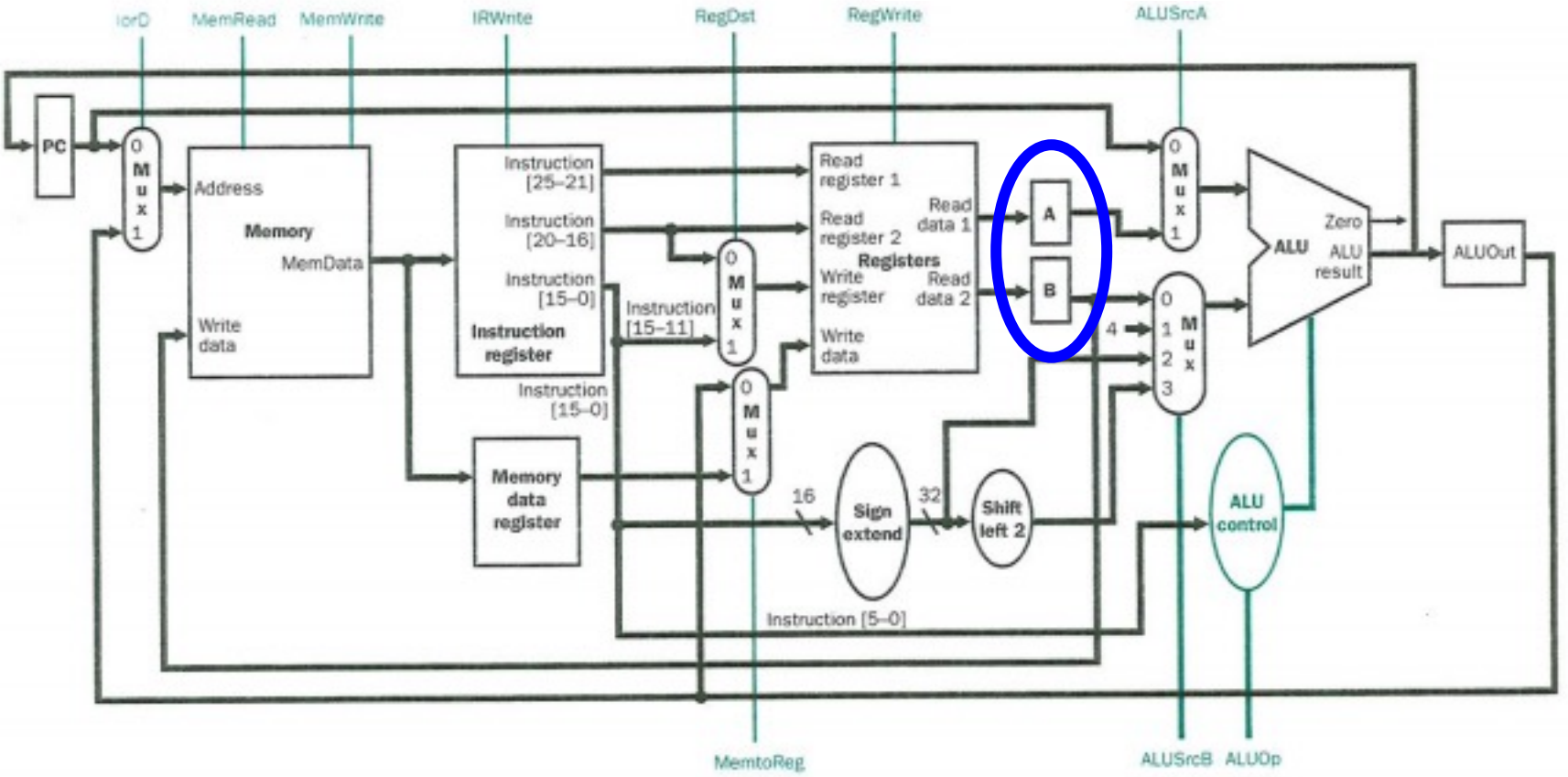
Multi-cycle datapath (overview)

Instruction Register (IR)

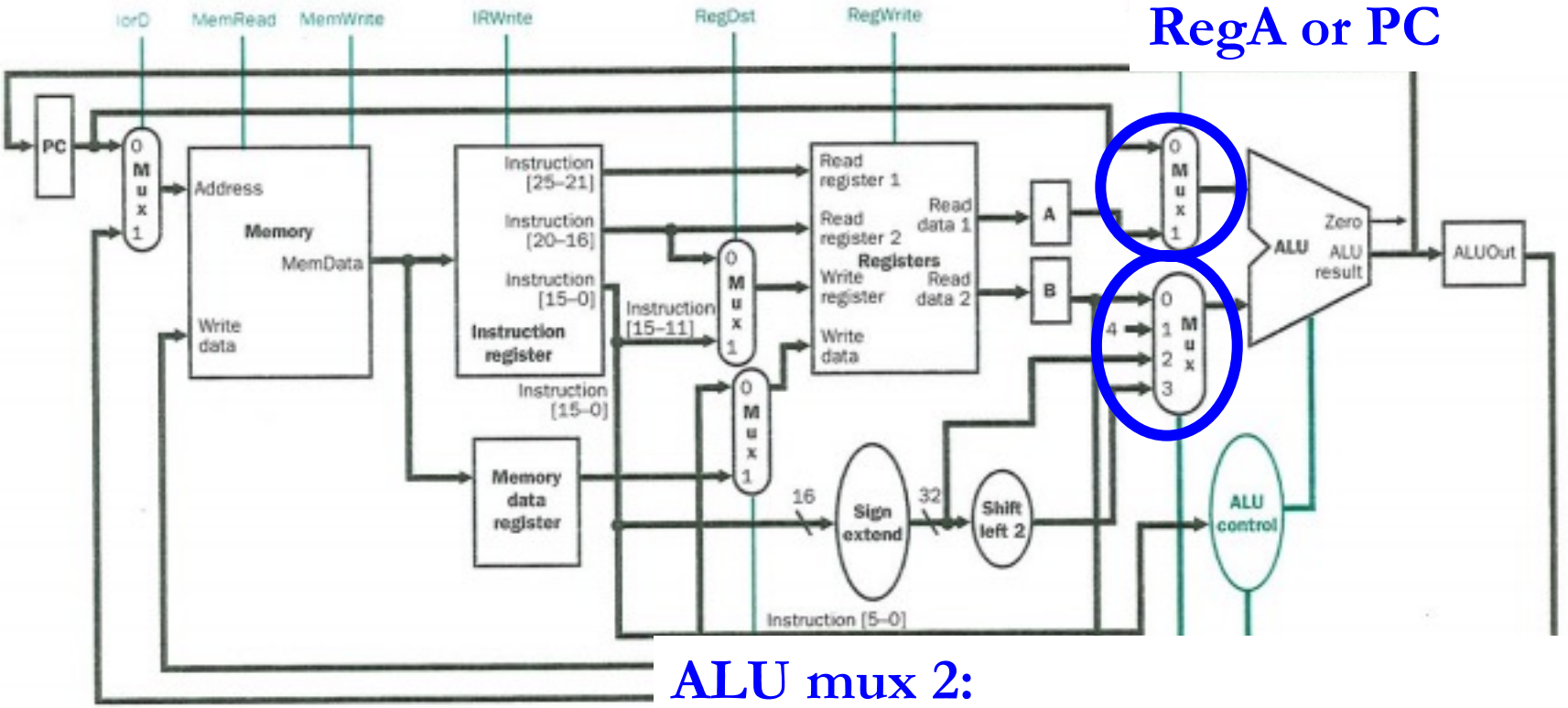


Multi-cycle datapath (overview)

Read Registers A & B



Multi-cycle datapath (overview)



ALU mux 1:
RegA or PC

ALU mux 2:

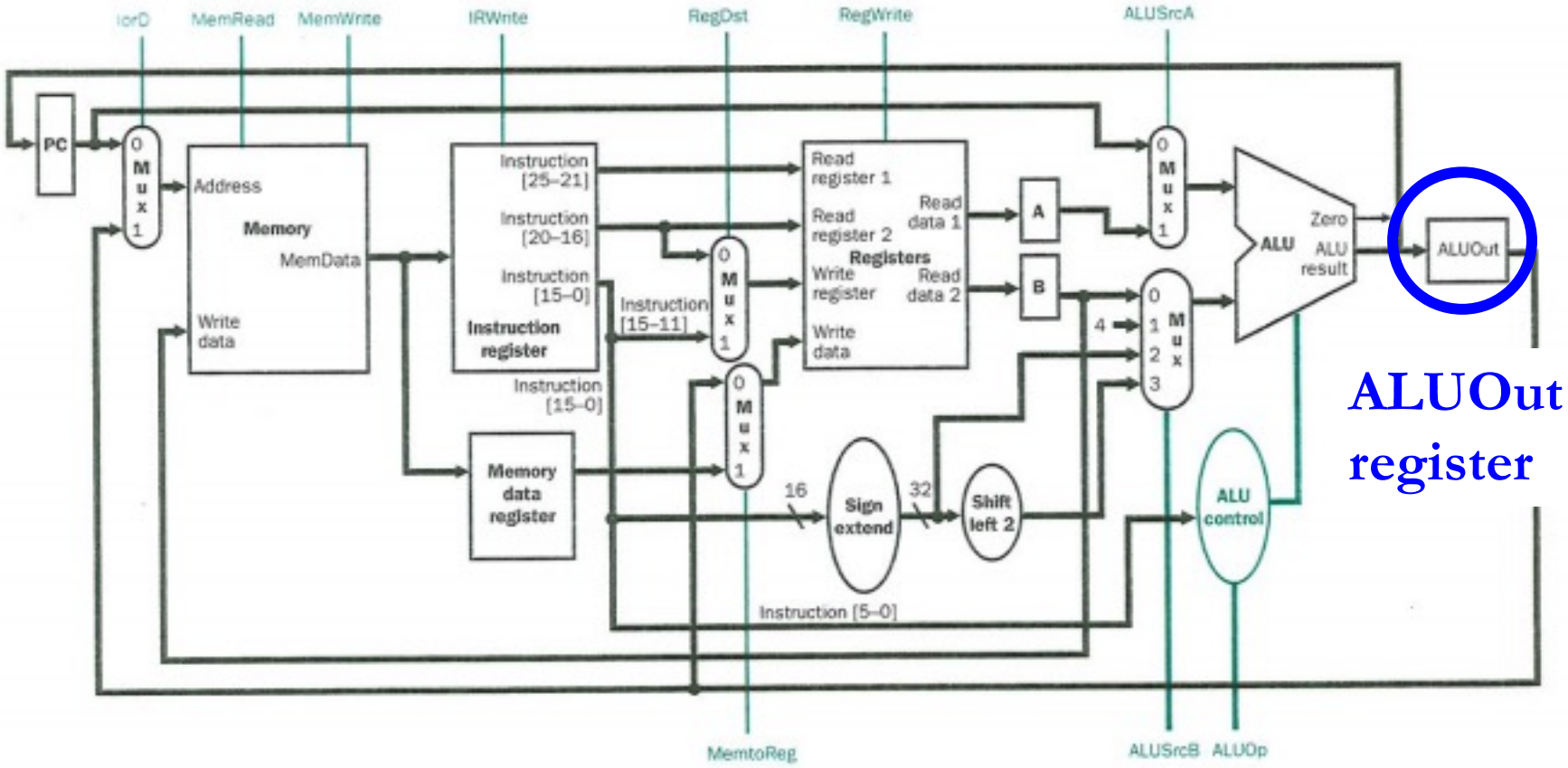
(1) RegB

(2) +4

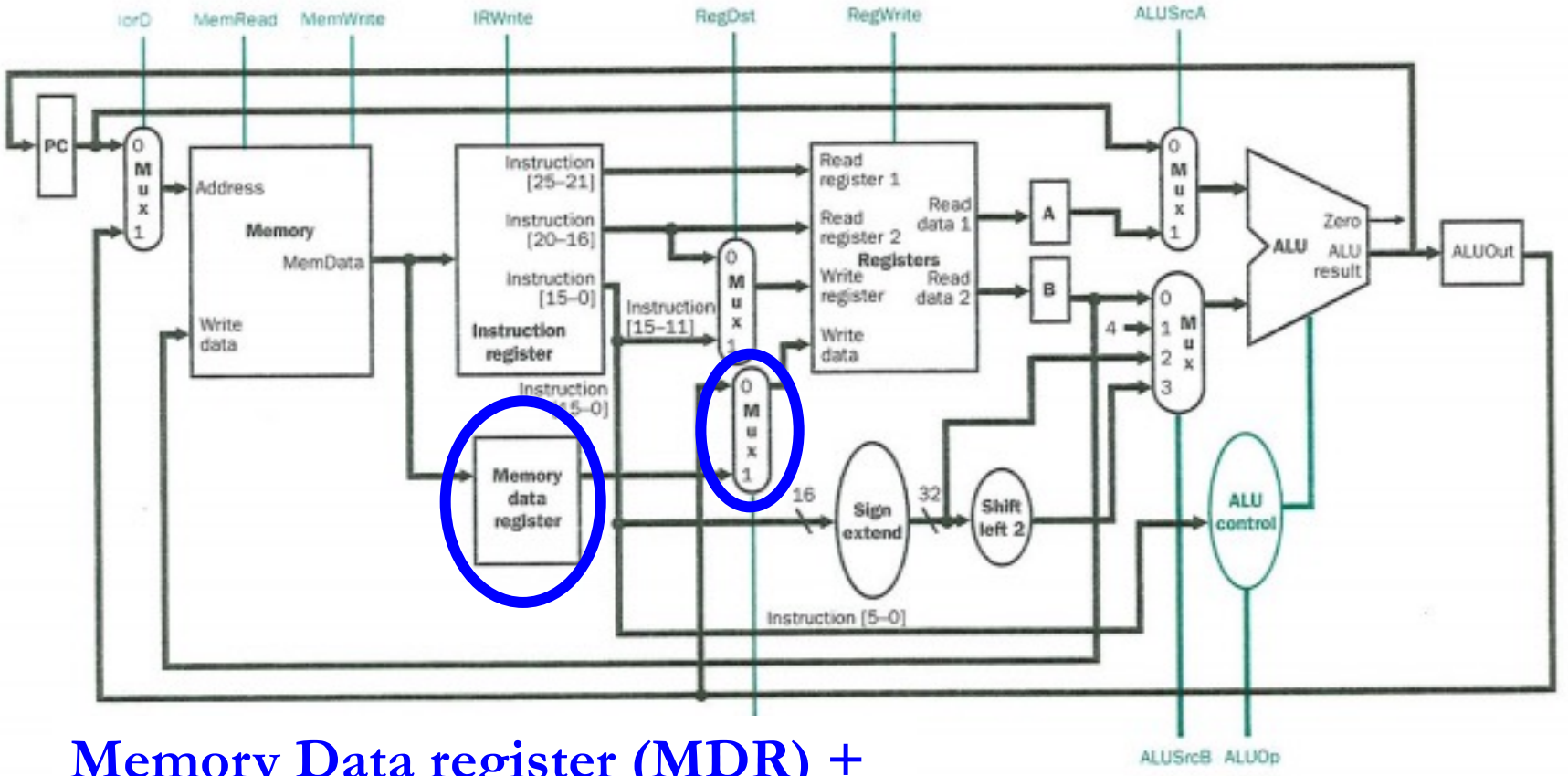
(3) sign-extended immediate

(4) sign-extended shifted immediate

Multi-cycle datapath (overview)

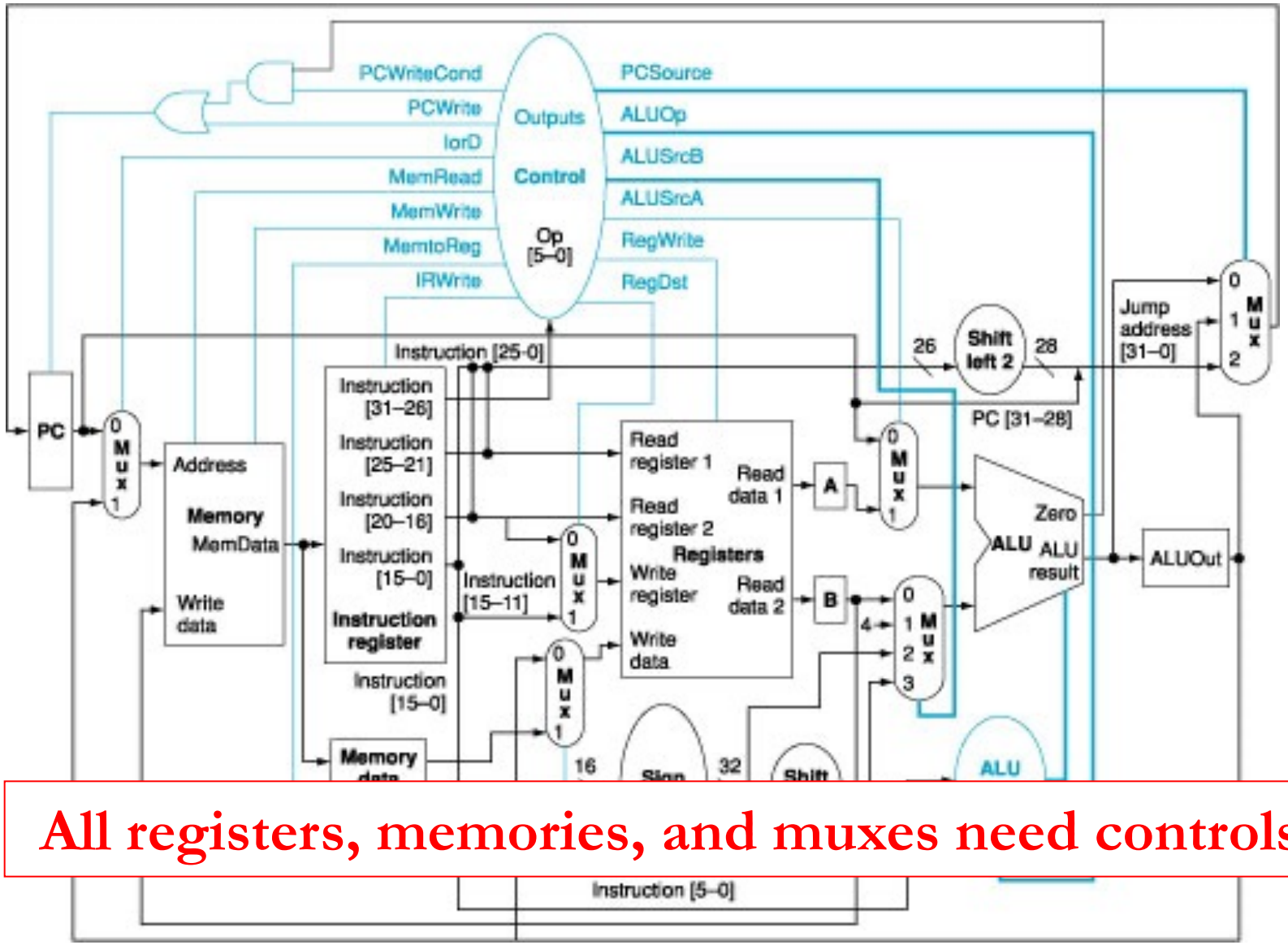


Multi-cycle datapath (overview)



Memory Data register (MDR) +
Write Data mux

Multi-cycle datapath (with control)



All registers, memories, and muxes need controls

Multi-cycle processor control signals (1-bit)

Signal name	Function
RegDst	Write register in the register file (*rt* or *rd*)
RegWrite	Write the register selected via RegDst?
ALUSrcA	Select ALU input A (*PC* or *A* register)
MemRead	Read the memory (could be for instruction fetch or data load)
MemWrite	Write the memory
MemtoReg	Select the source of da or *MDR*)
IorD	Select the source of the address for the memory unit (*PC* or *ALUout*)
IRWrite	Write the IR with output of the memory unit
PCWrite	Write the PC register (source controlled by *PCSource*)
PCWriteCond	Write the PC if *Zero* output from the ALU is active

New in multi-cycle processor

Multi-cycle processor control signals (2-bit)

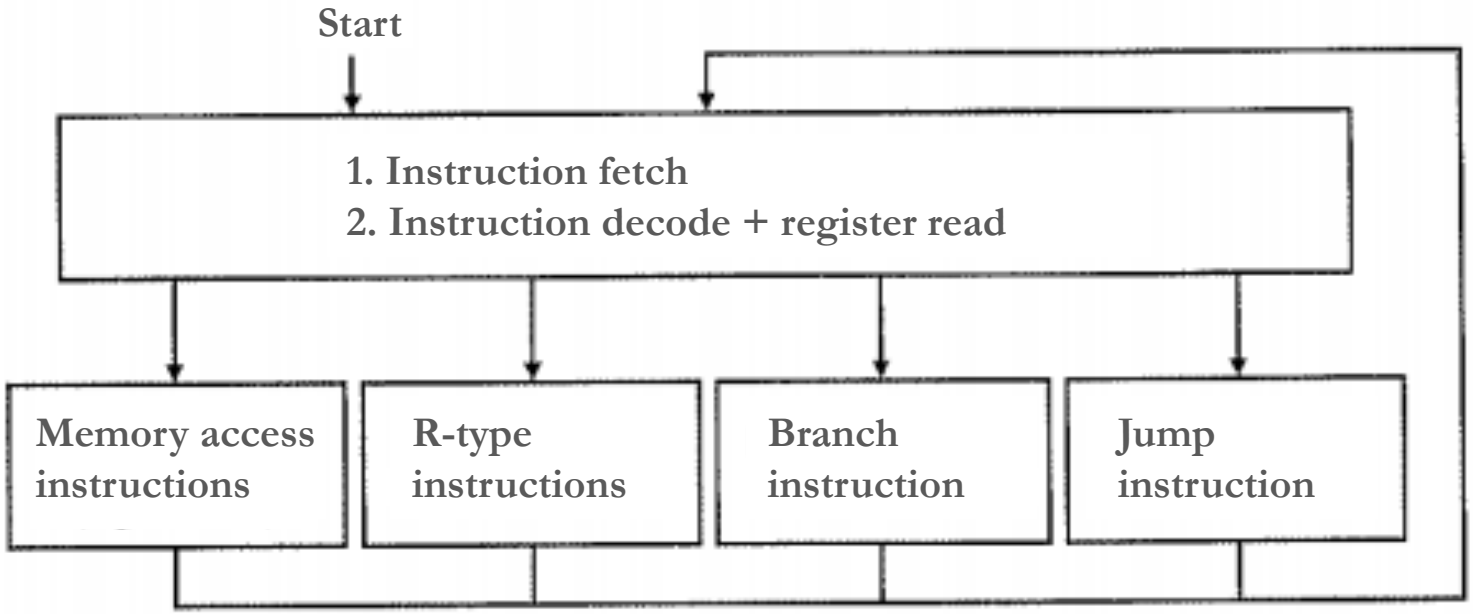
Signal name	Value	Function
ALUSrcB	00	Second ALU input comes from B register
	01	Second ALU input is a constant 4
	10	Second ALU input is sign-extended lower 16 bits of IR
	11	Second ALU input is sign-extended lower 16-bits of IR shifted left by 2
ALUOp	00	ALU performs an Add
	01	ALU performs a Subtract
	10	ALU action determined by FUNCT field
PCSource	00	Output of the ALU (PC+4)
	01	ALUout (branch target address)
	10	Jump target address (IR[25-0] shifted left 2 bits and combined with PC+4[31-28])

How to design the control

- The control unit of a multicycle processor is an FSM

- For a given instruction type, determines the sequence of control signals on a cycle-by-cycle basis
  - On a given cycle: outputs the control signals and next FSM state

Control FSM overview

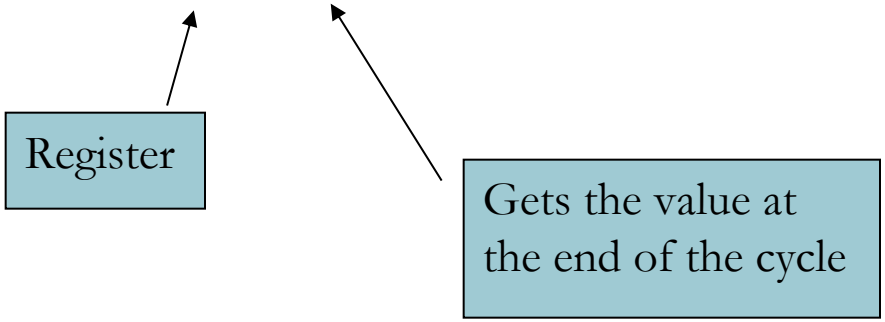


- Fetch & Decode common for all insts
  - 2 cycles (Fetch, Decode)
- Rest: depends on the inst type
  - 1 to 3 additional cycles

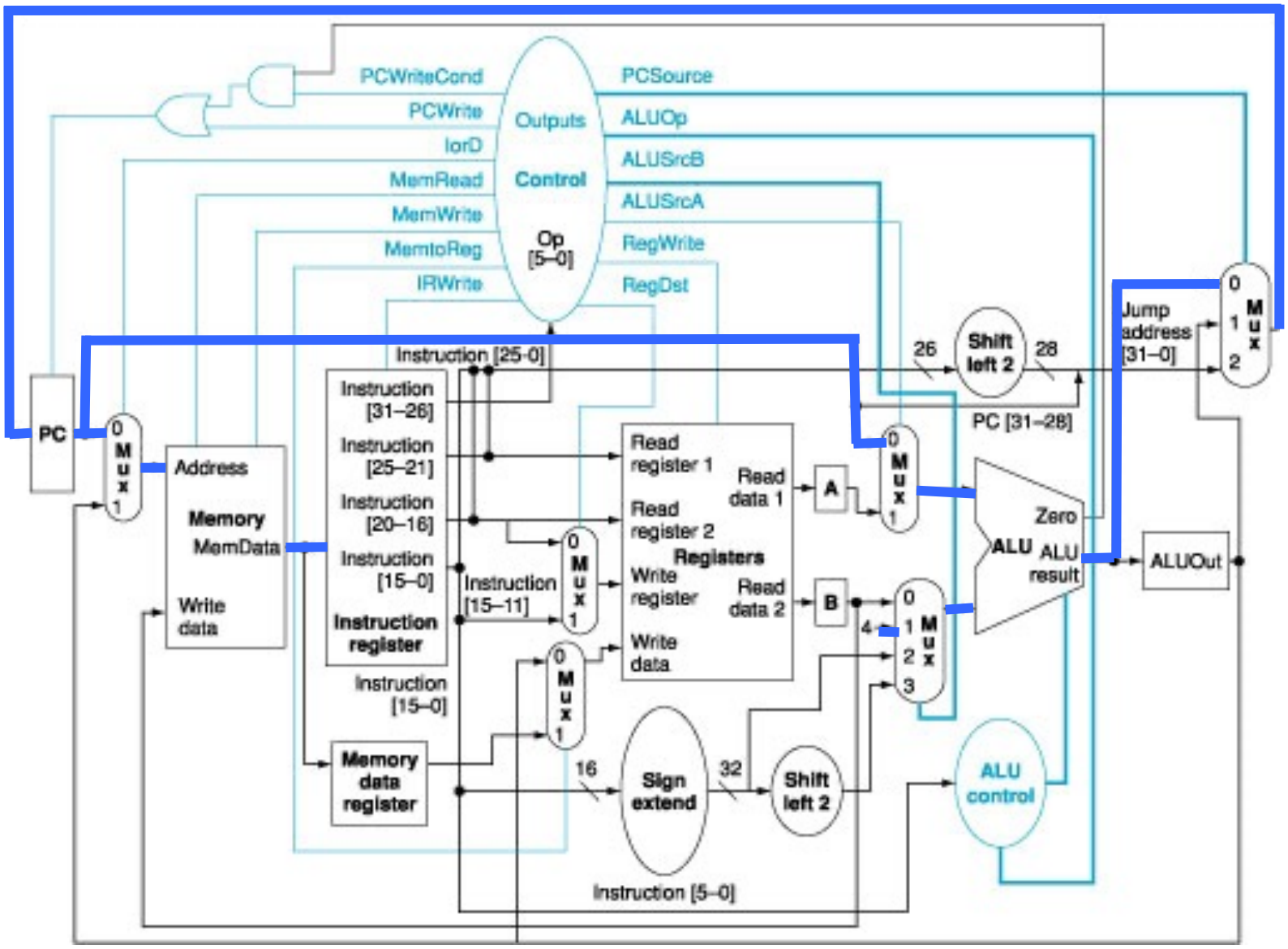
What happens in each cycle – 1 & 2

1. Instruction fetch

```
IR <= Mem[PC]
PC <= PC+4
```



Cycle 1 – instruction fetch (all)



What happens in each cycle – 1 & 2

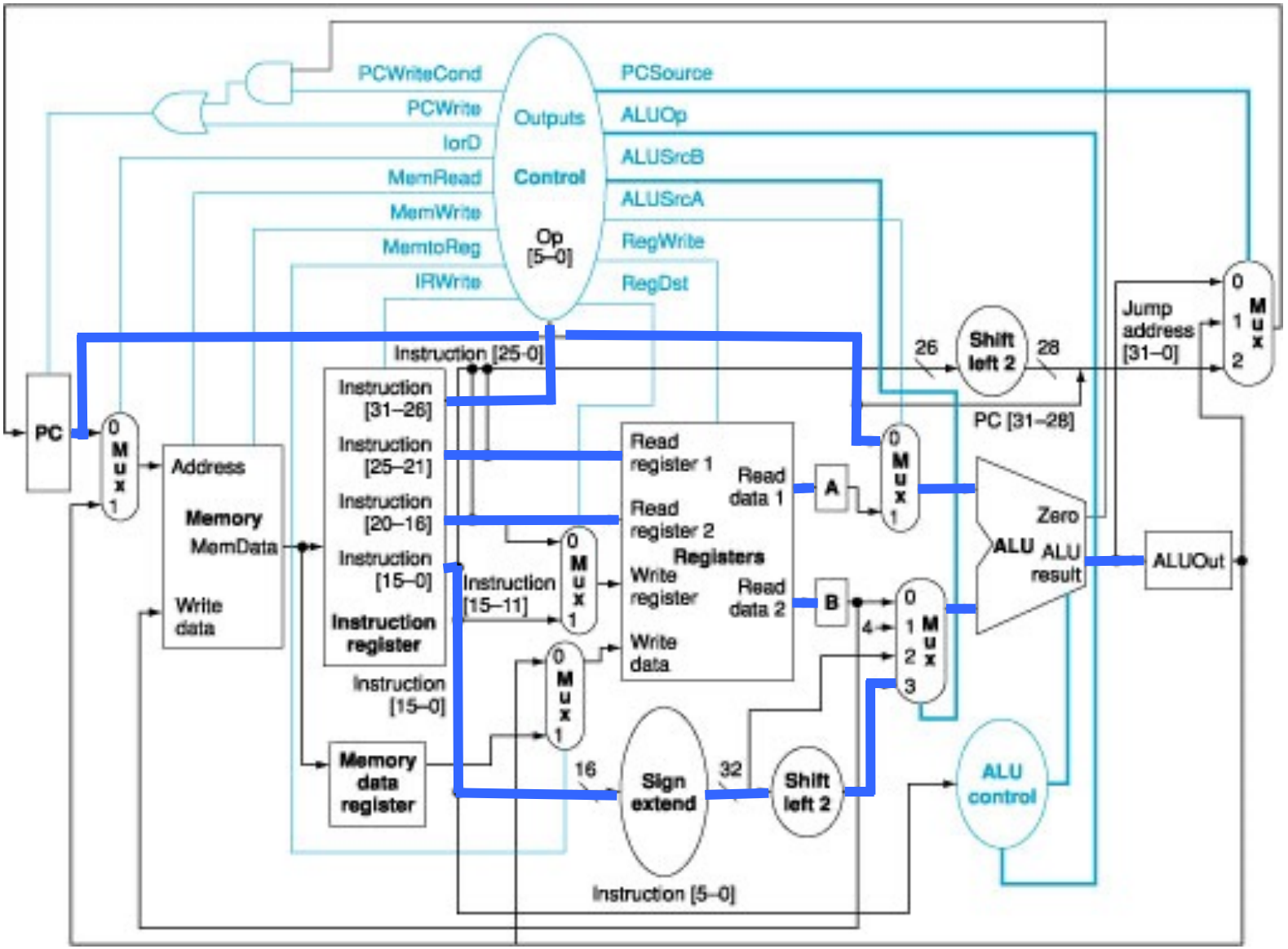
1. Instruction fetch

```
IR <= Mem[PC]
PC <= PC+4
```

2. Instruction decode and register read

```
A <= Reg[IR[25:21]]
B <= Reg[IR[20:16]]
ALUOut <= PC+sgnext(IR[15:0]<<2)
```

Cycle 2 –instr decode & reg read (all)



What happens in each cycle – 3

3a. R-type arithmetic/logical instruction

```
ALUOut <= A op B
```

3b. Immediate arithmetic, including memory address generation

```
ALUOut <= A + sgnext(IR[15:0])
```

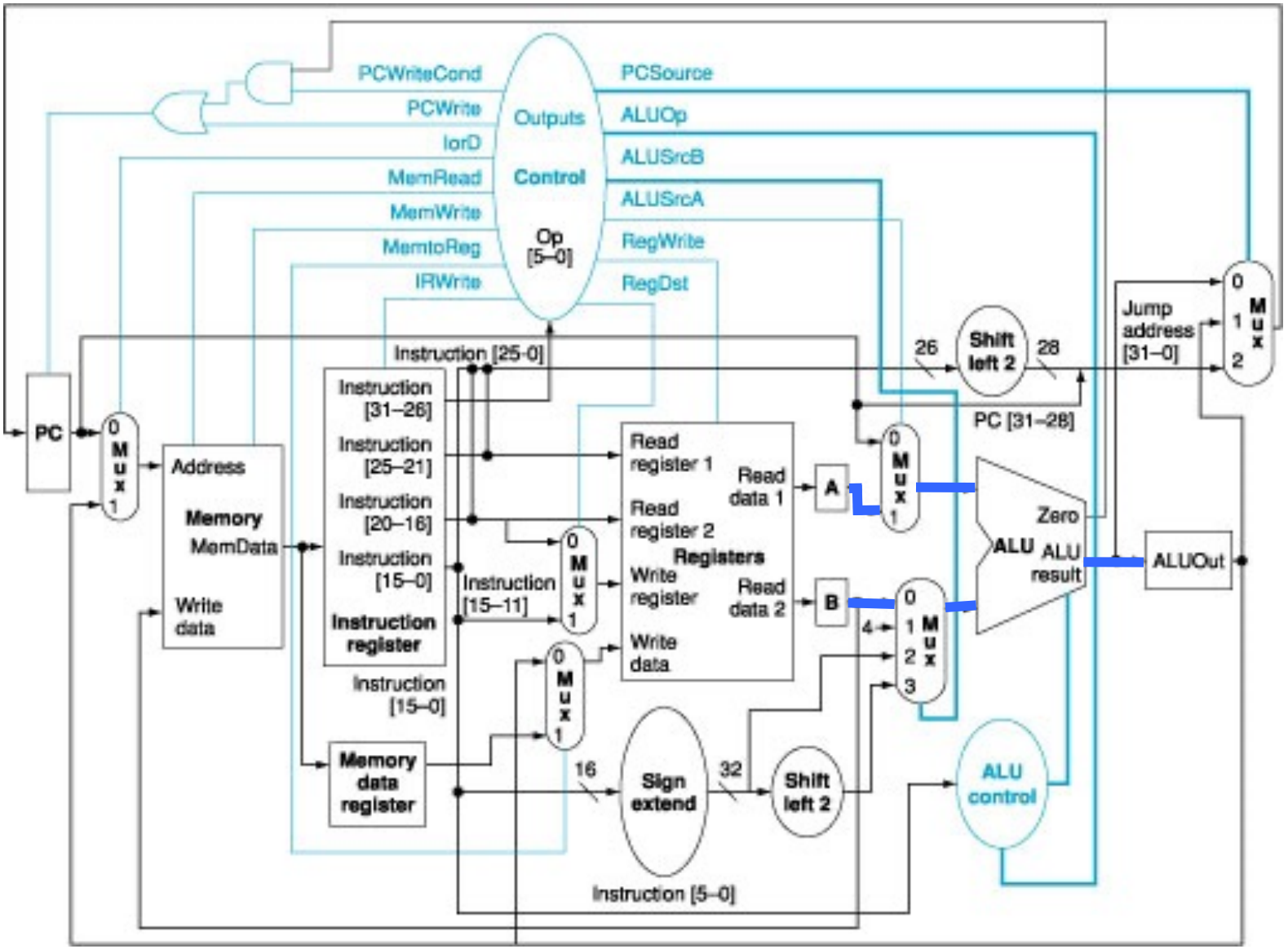
3c. Branch completion

```
if (A == B) PC <= ALUOut
```

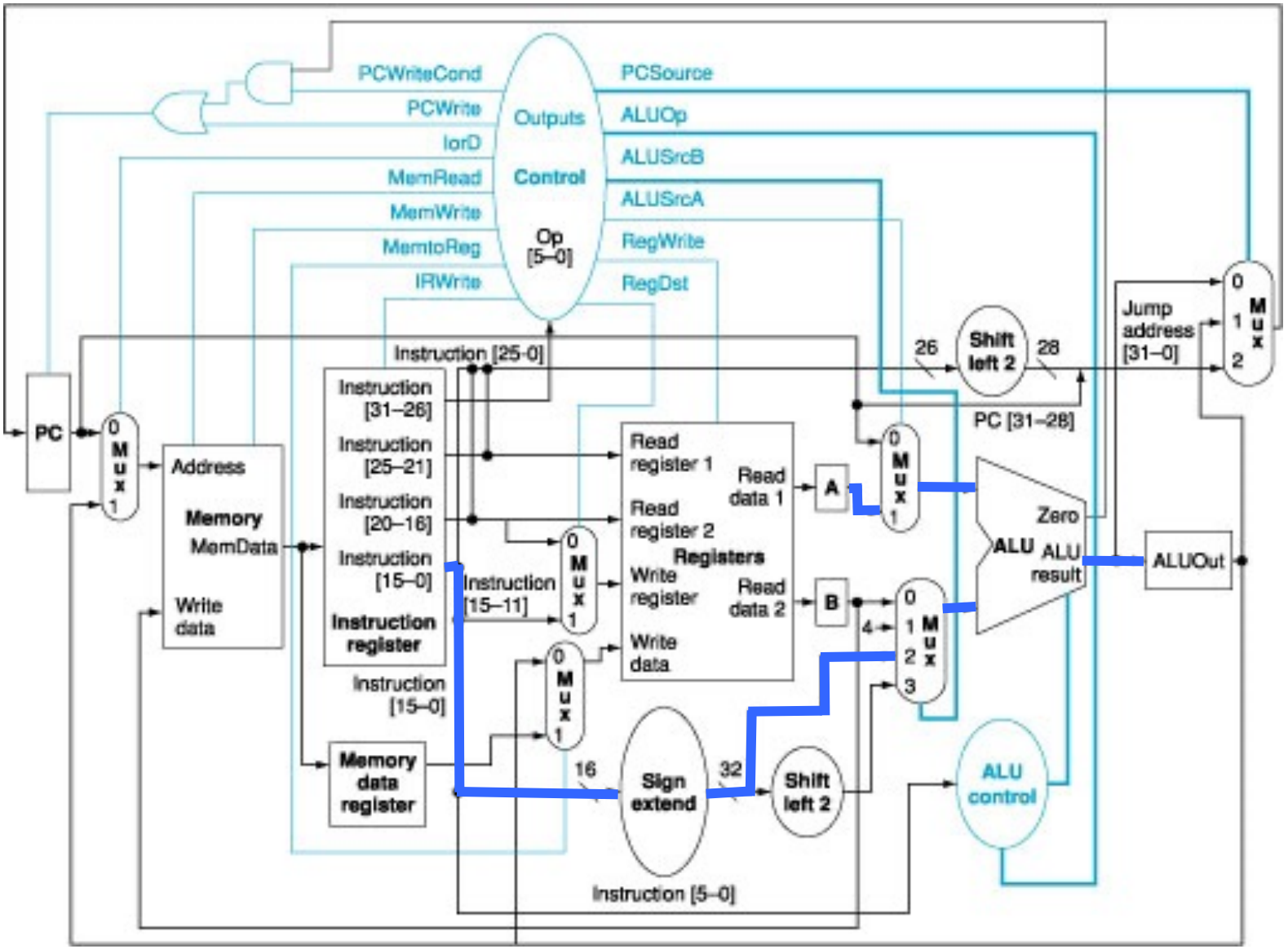
3d. Jump completion

```
PC <= {PC[31:28],IR[25:0],2'b00}
```

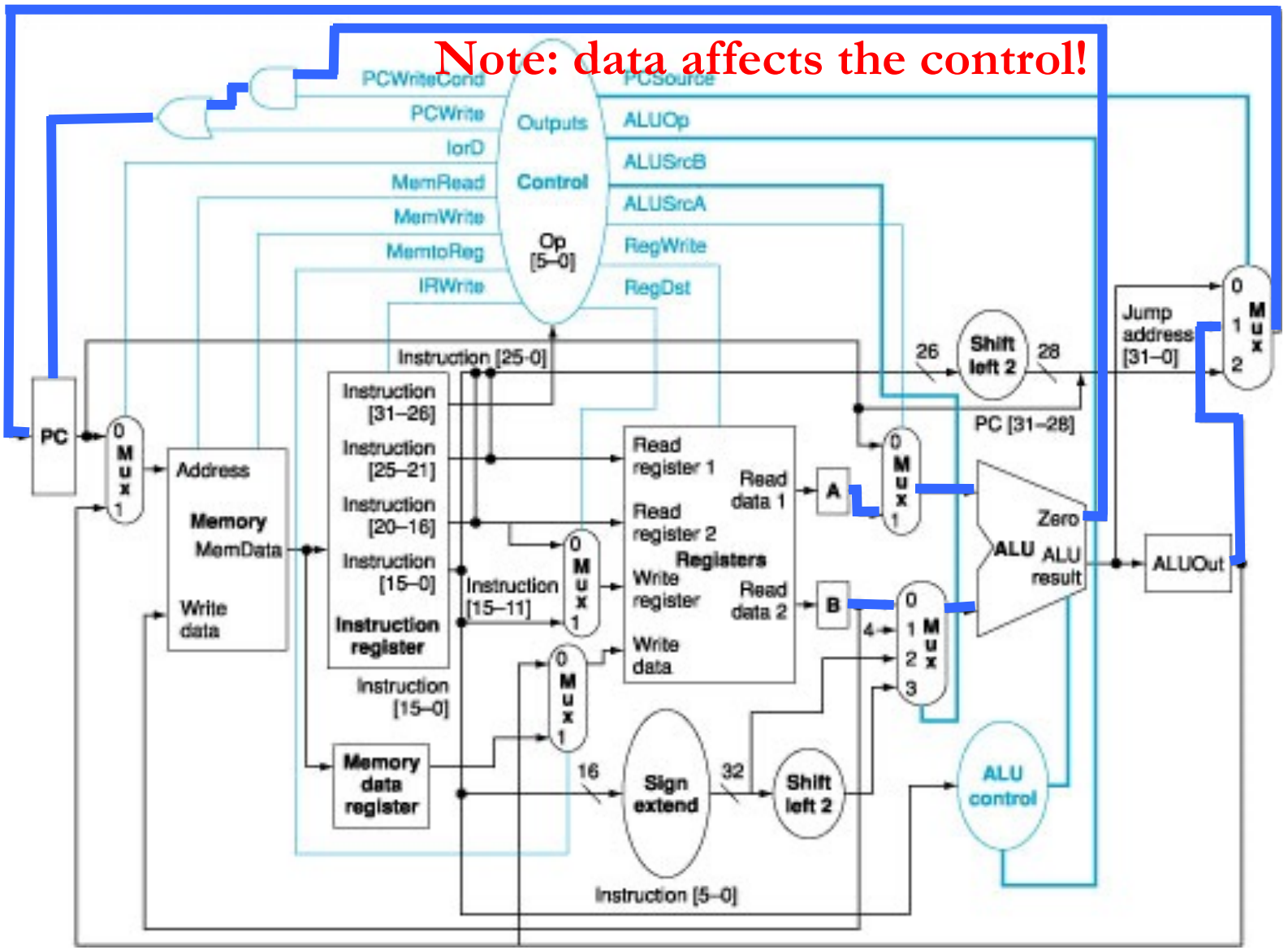
Cycle 3a: R-type ALU op (add, and)



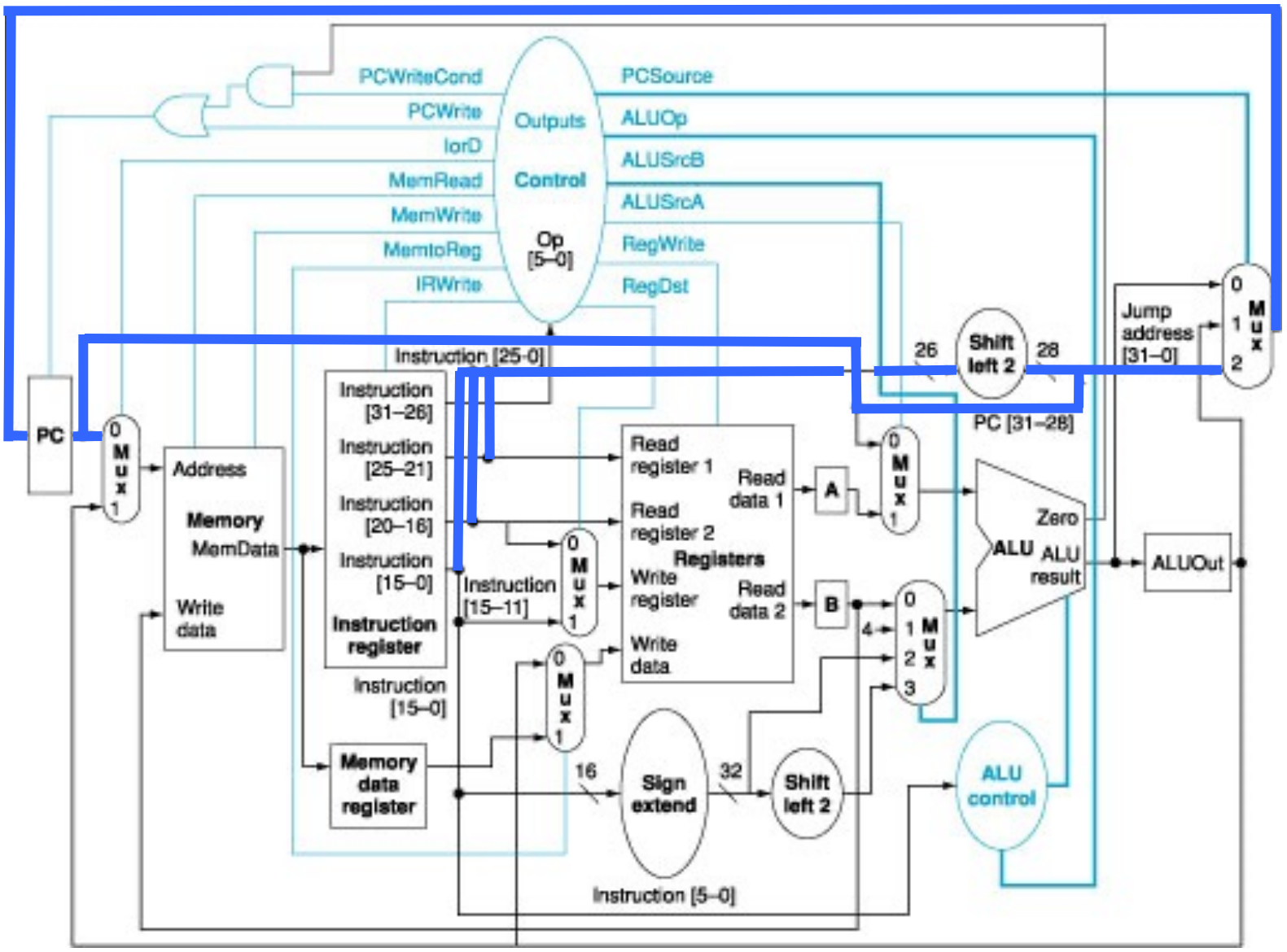
Cycle 3b: imm ALU op (addi, lw, sw)



Cycle 3c: Branch taken (beq, bne)



Cycle 3d: jump (j)



What happens in each cycle – 4

4a. R-type arith-logical instruction completion

```
Reg[IR[15:11]] <= ALUOut
```

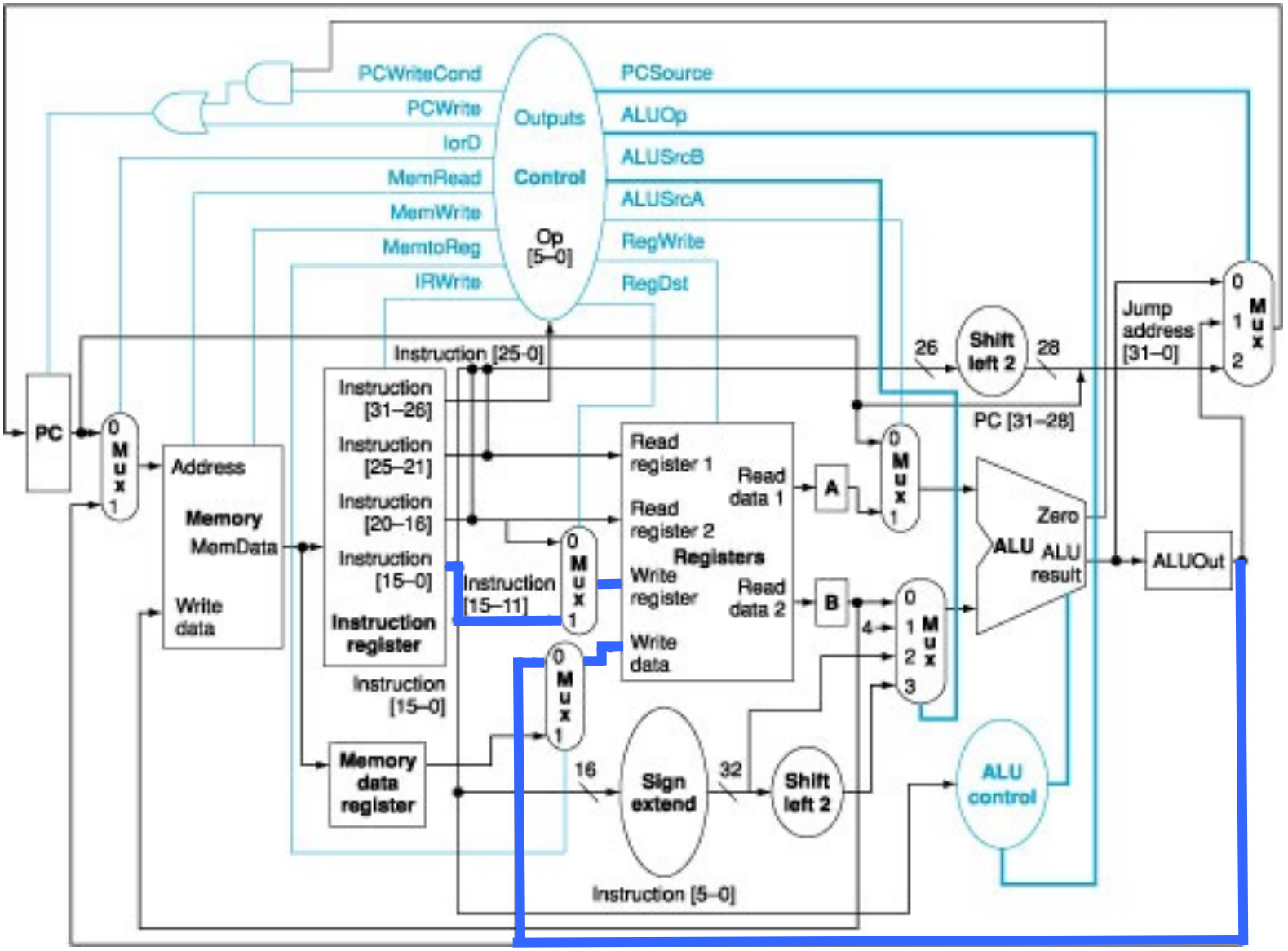
4b. Memory access (load)

```
MDR <= Mem[ALUOut]
```

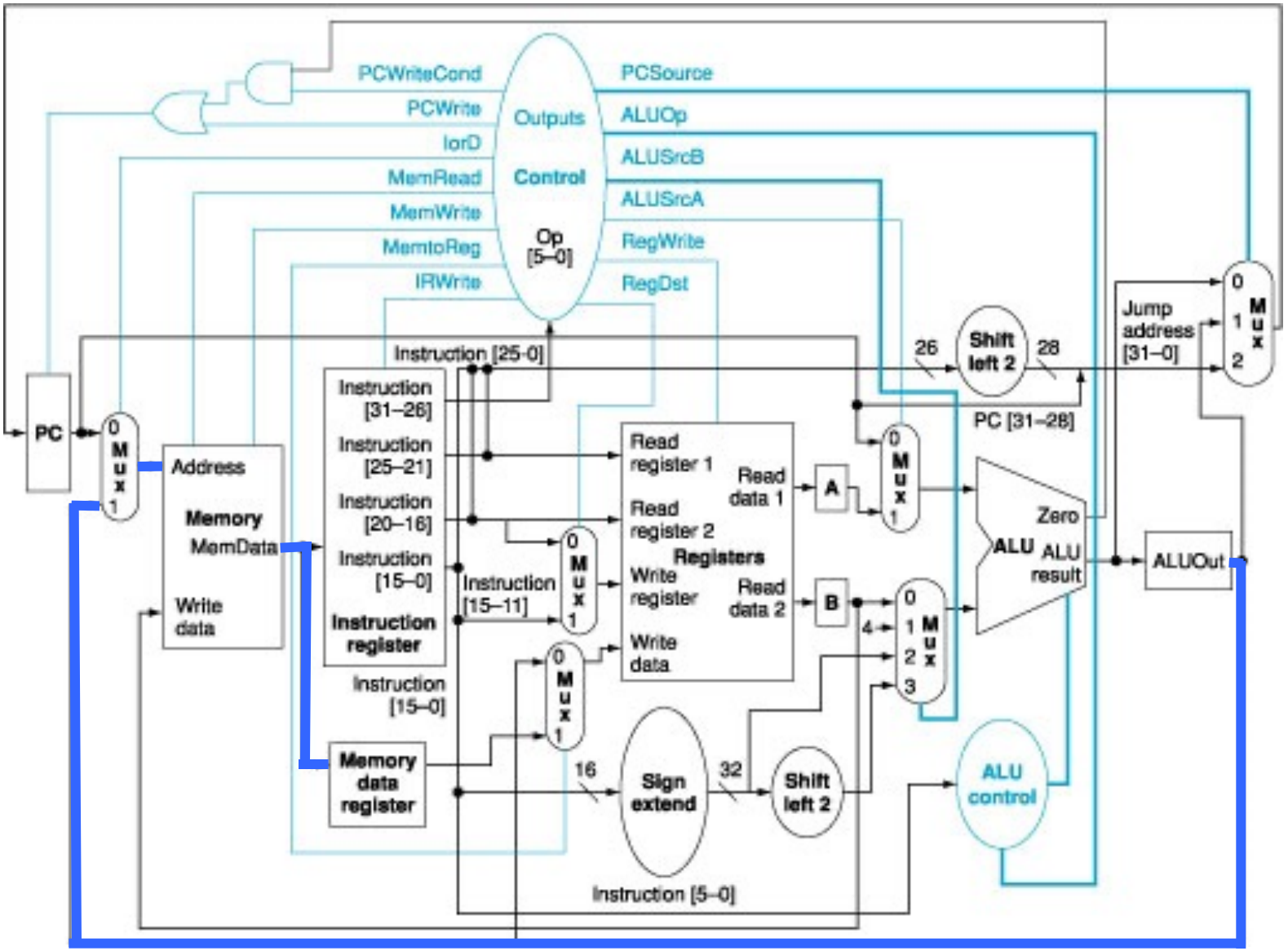
4c. Memory access (store) & completion

```
Mem[ALUOut] <= B
```

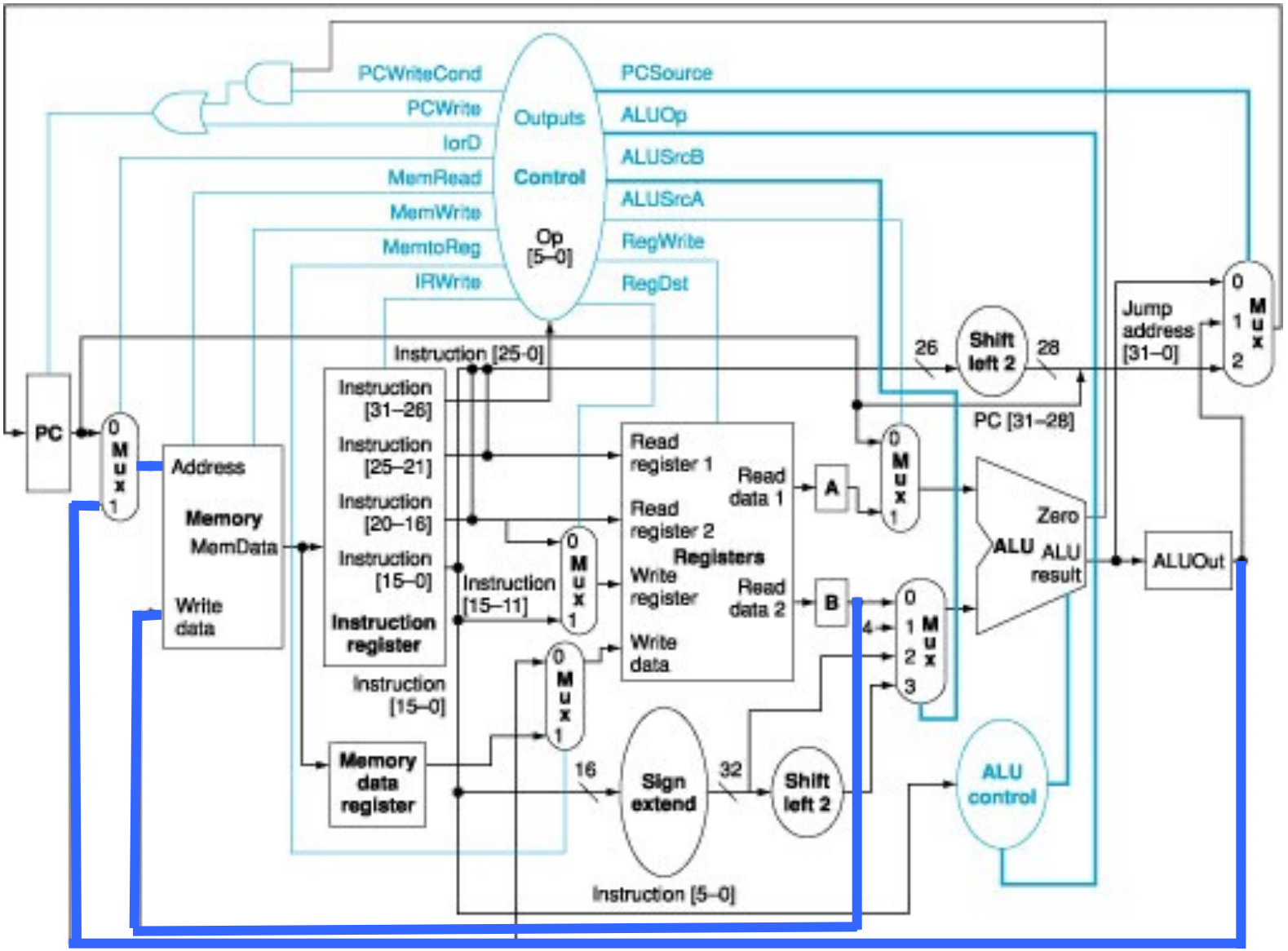
Cycle 4a: R-type result write (add, and)



Cycle 4b: Load from mem (lw)



Cycle 4c: Store to mem (sw)

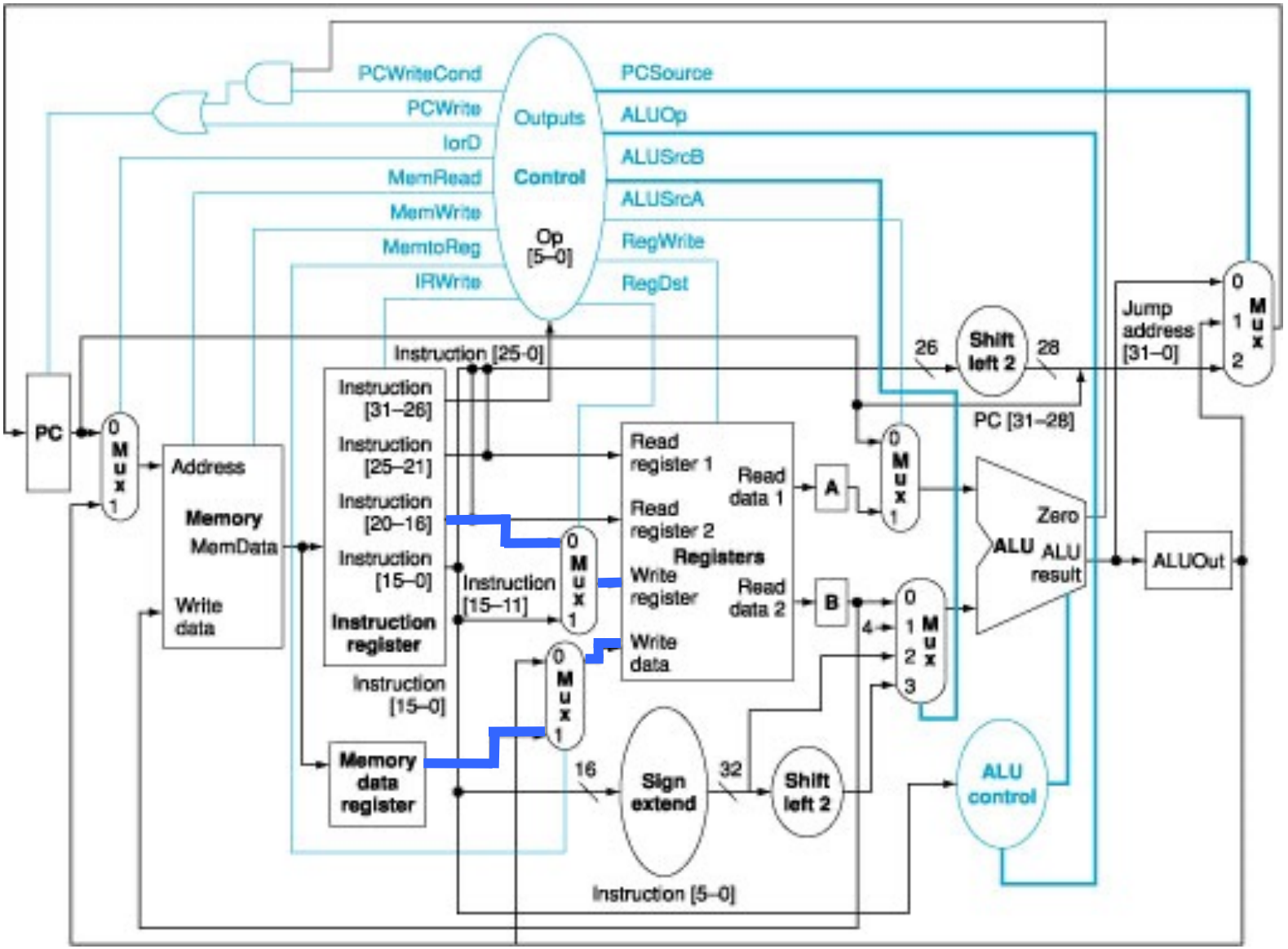


What happens in each cycle – 5

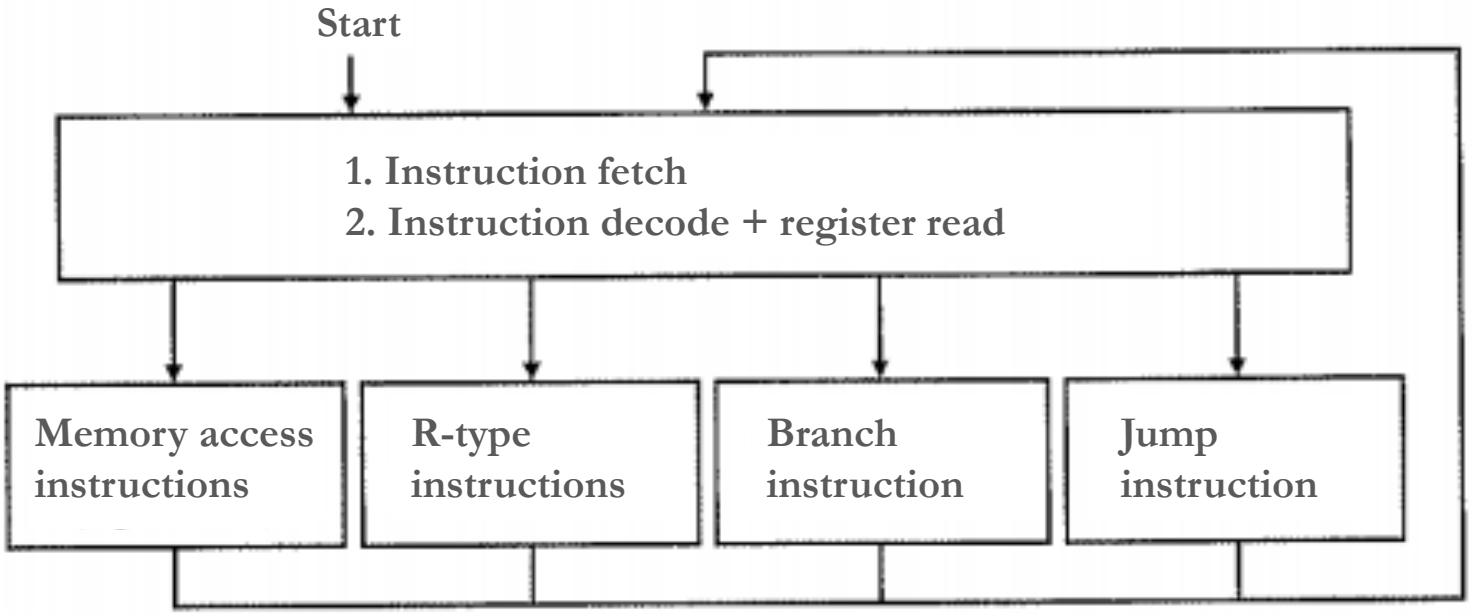
5. Load instruction completion

```
Reg[IR[20:16]] <= MDR
```

Cycle 5: save of loaded value (lw)

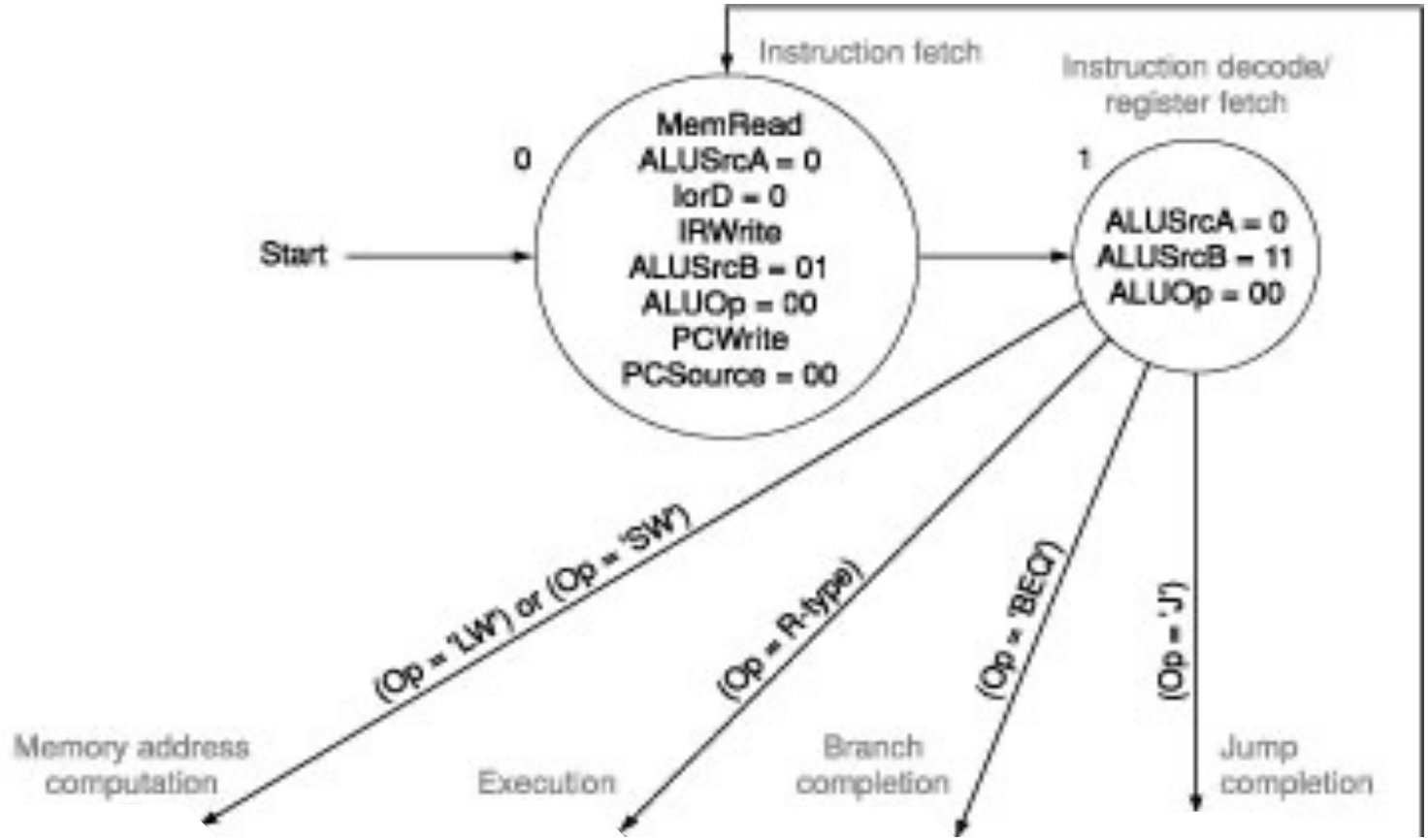


Designing the Control Unit

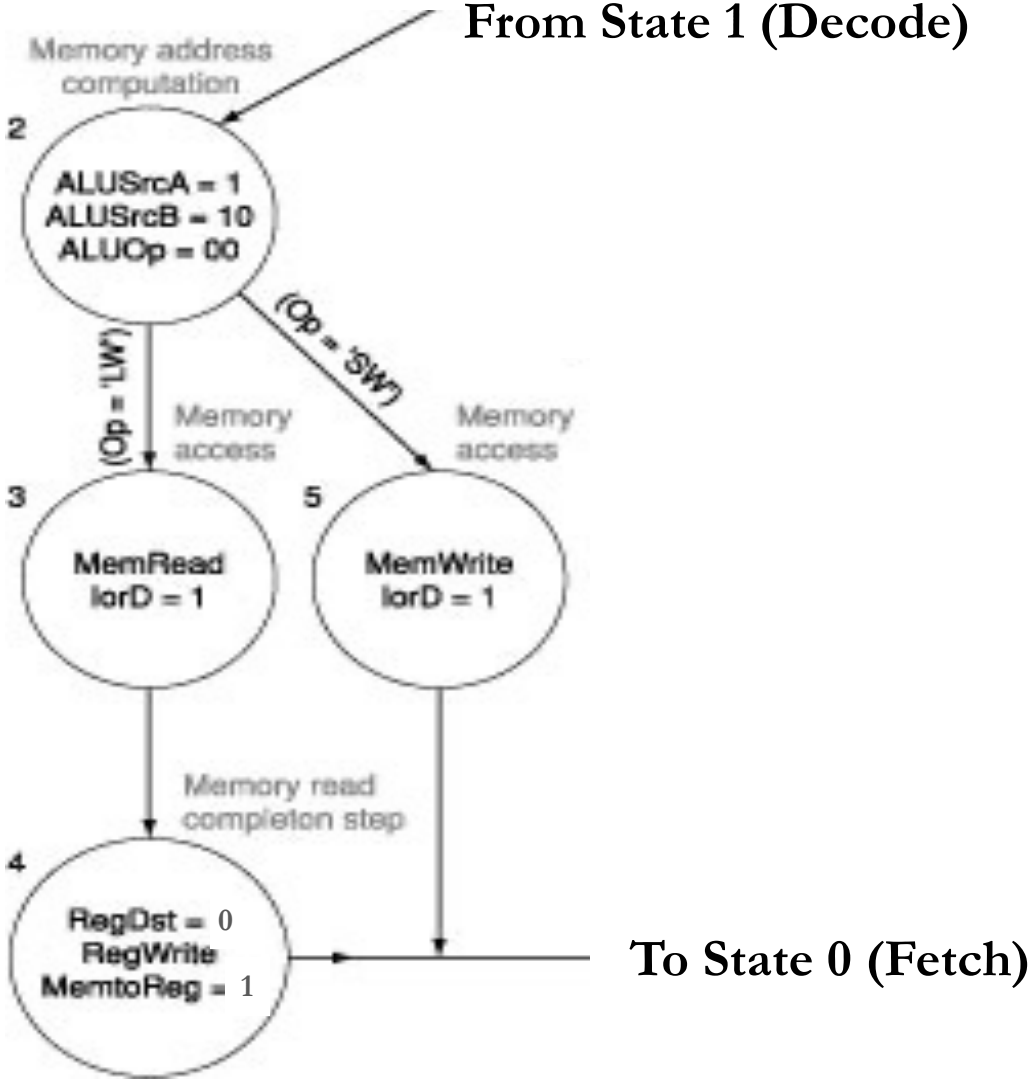


- Fetch & Decode common for all insts
  - 2 cycles total (Fetch, Decode)
- Rest: depends on the inst type
  - 1 to 3 additional cycles

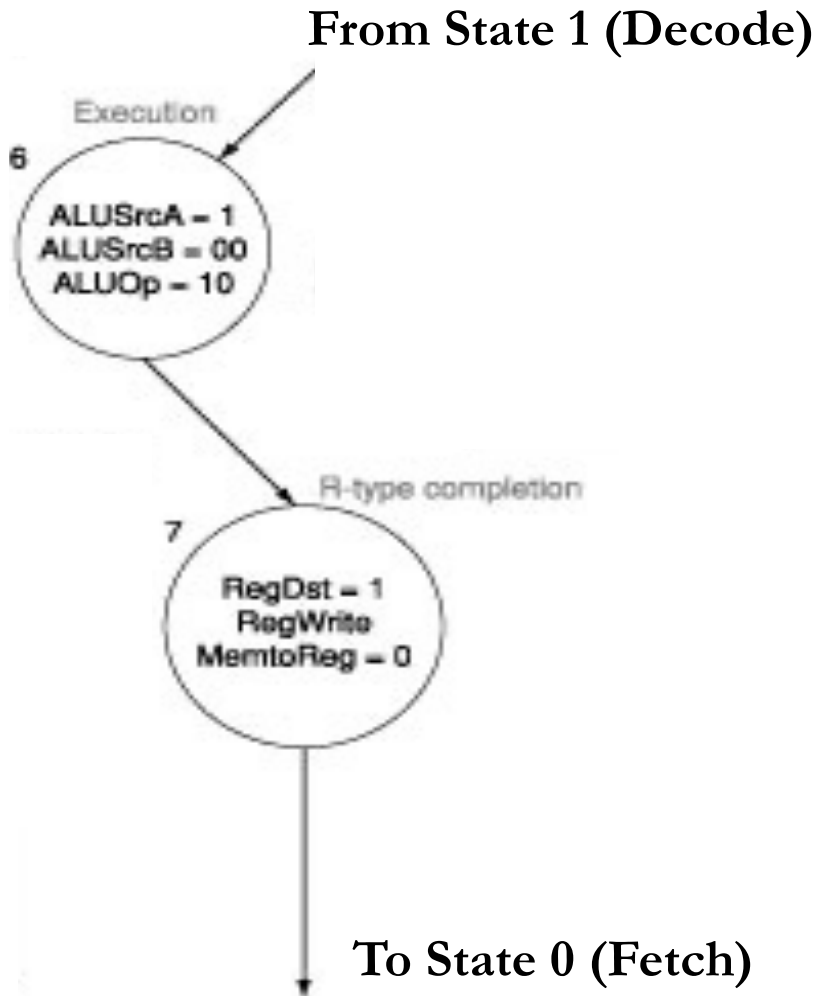
Fetch and Decode States



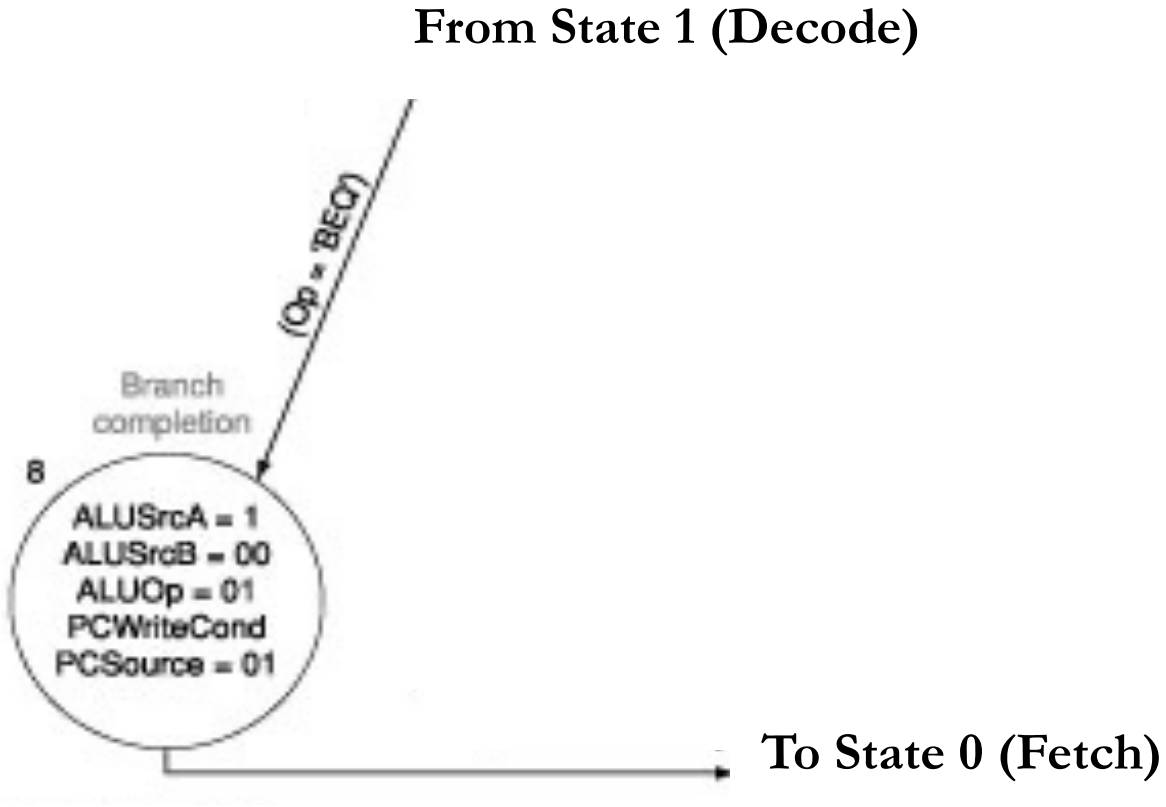
Memory Access States



R-Type Instruction States

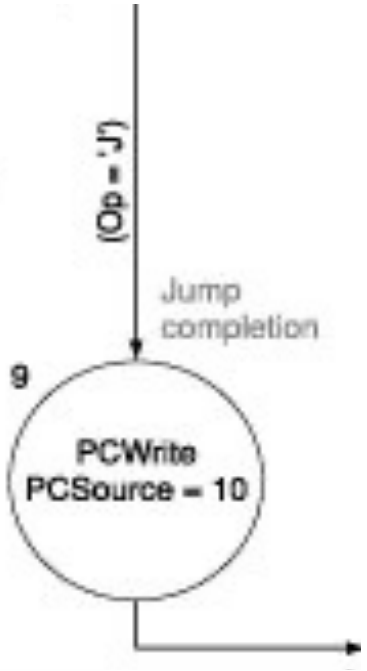


Branch Resolution States

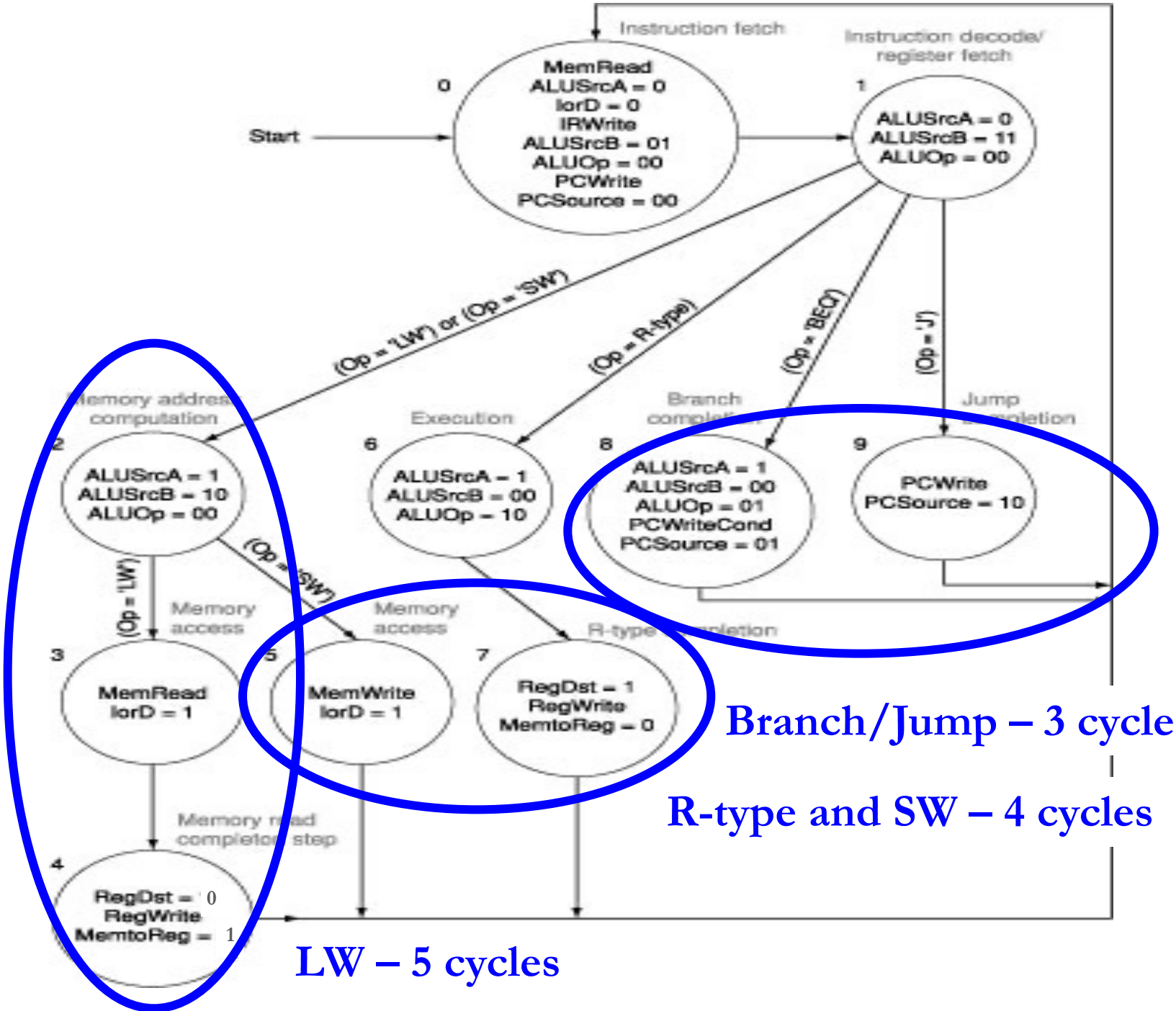


Jump Resolution States

From State 1 (Decode)



To State 0 (Fetch)



Branch/Jump – 3 cycles

R-type and SW – 4 cycles

LW – 5 cycles

Implications of Processor Design Choices

Execution time =

instruction count

x

cycles per instruction

x

cycle time

Single- vs multi-cycle processor:
which one should yield higher performance?