

Lecture 9: Processor design – multi cycle

- 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 cycle

- Multiple & varied cycles per instruction means that no instruction takes more time or uses more func. units than required



Lecture outline

- Brief processor performance evaluation
- Determine the components
- Build the datapath
- Build the control



Measuring processor speed

Execution time is

$$\begin{aligned} & \text{instruction count} \\ & \quad \times \\ & \text{cycles per instruction} \\ & \quad \times \\ & \text{cycle time} \end{aligned}$$



Determine the components

Processor task

- Instruction fetch from memory
- Read registers
- Execution
 - Data processing instructions
 - Data transfer instructions
 - Branch instructions

Component list

- PC register
- Memory (~~instructions~~)
- ~~Adder: PC+4~~
- Register file
 - 2 read, 1 write
- ALU

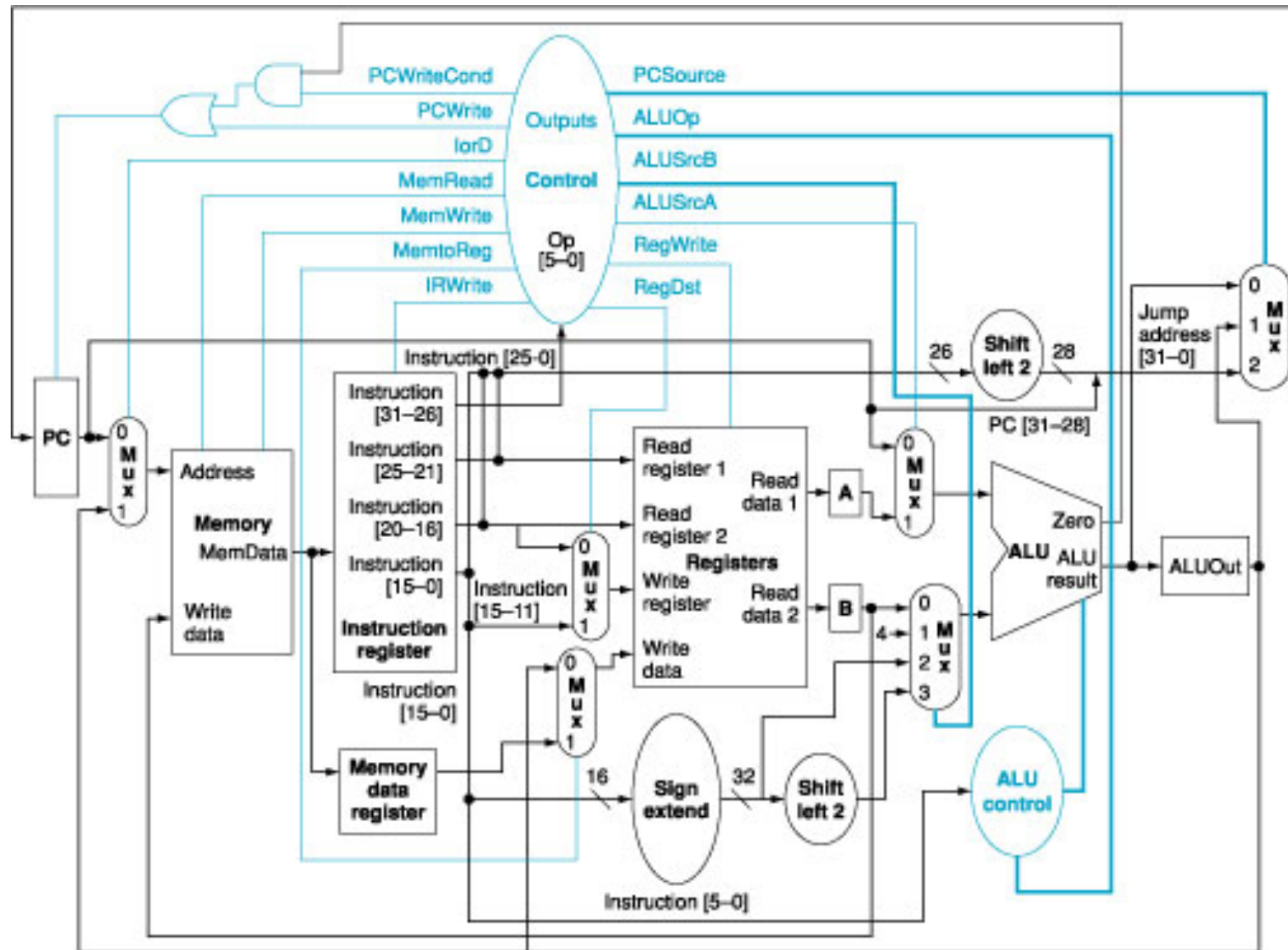


Design guidelines

- Cycle time determined by the delay through the slowest functional unit
- Reuse functional units as much as possible
 - Multiplexors added to select the different inputs
- At end of each cycle, data required in subsequent cycles must be stored somewhere
 - Data for other instructions are kept in the memory, register file, or the PC
 - Data for same instruction are kept in new registers not visible to the programmer



Multi-cycle datapath



How to design the control part

- The control unit of a multicycle processor is an FSM
- Determine exactly what happens in each cycle
- and what is the next step
- Be careful with register load-enable control signals



What happens in each cycle – 1 & 2

1. Instruction fetch

$IR \leftarrow Mem[PC]$

$PC \leftarrow PC+4$

2. Instruction decode and register fetch

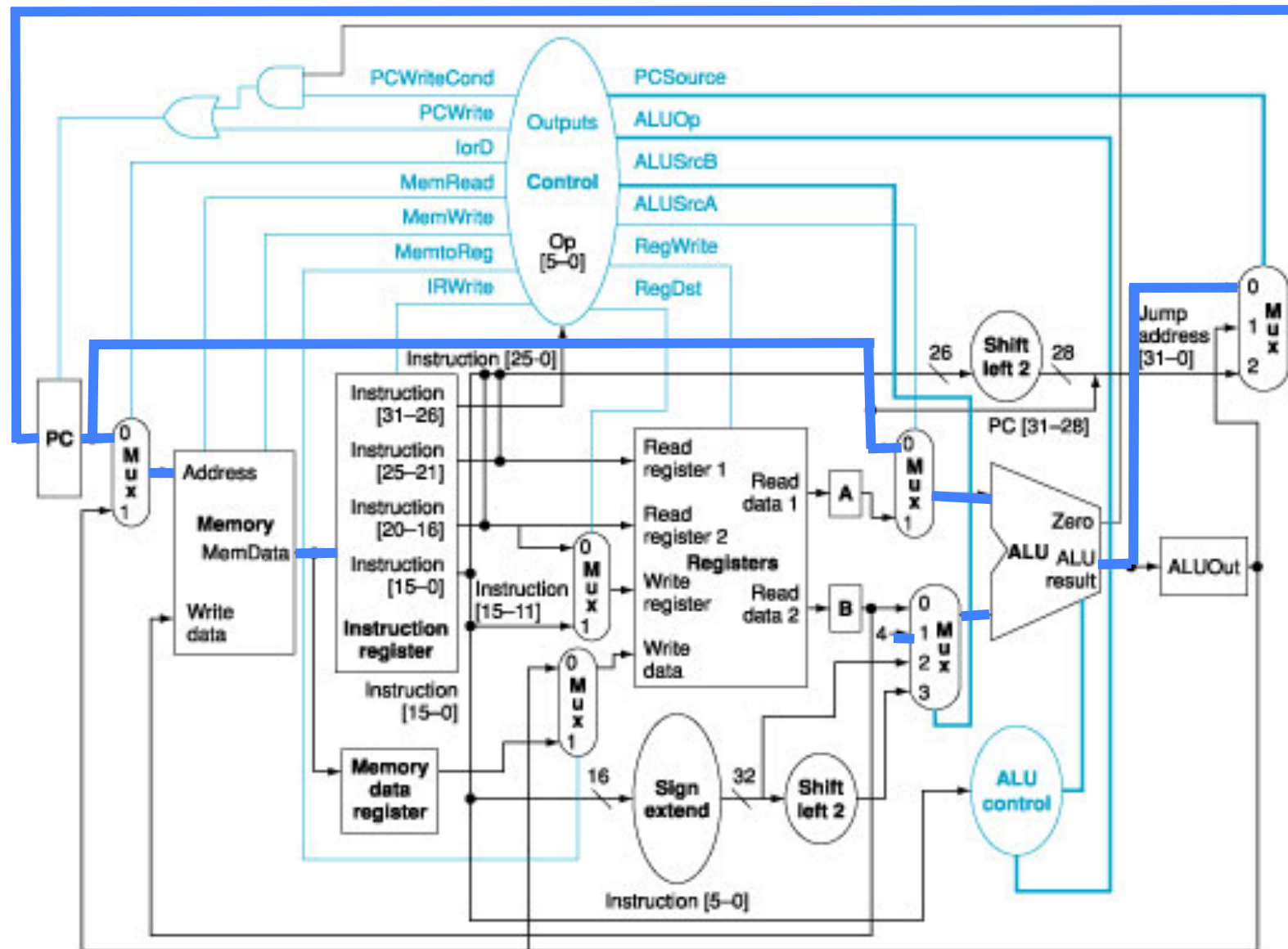
$A \leftarrow Reg[IR[25:21]]$

$B \leftarrow Reg[IR[20:16]]$

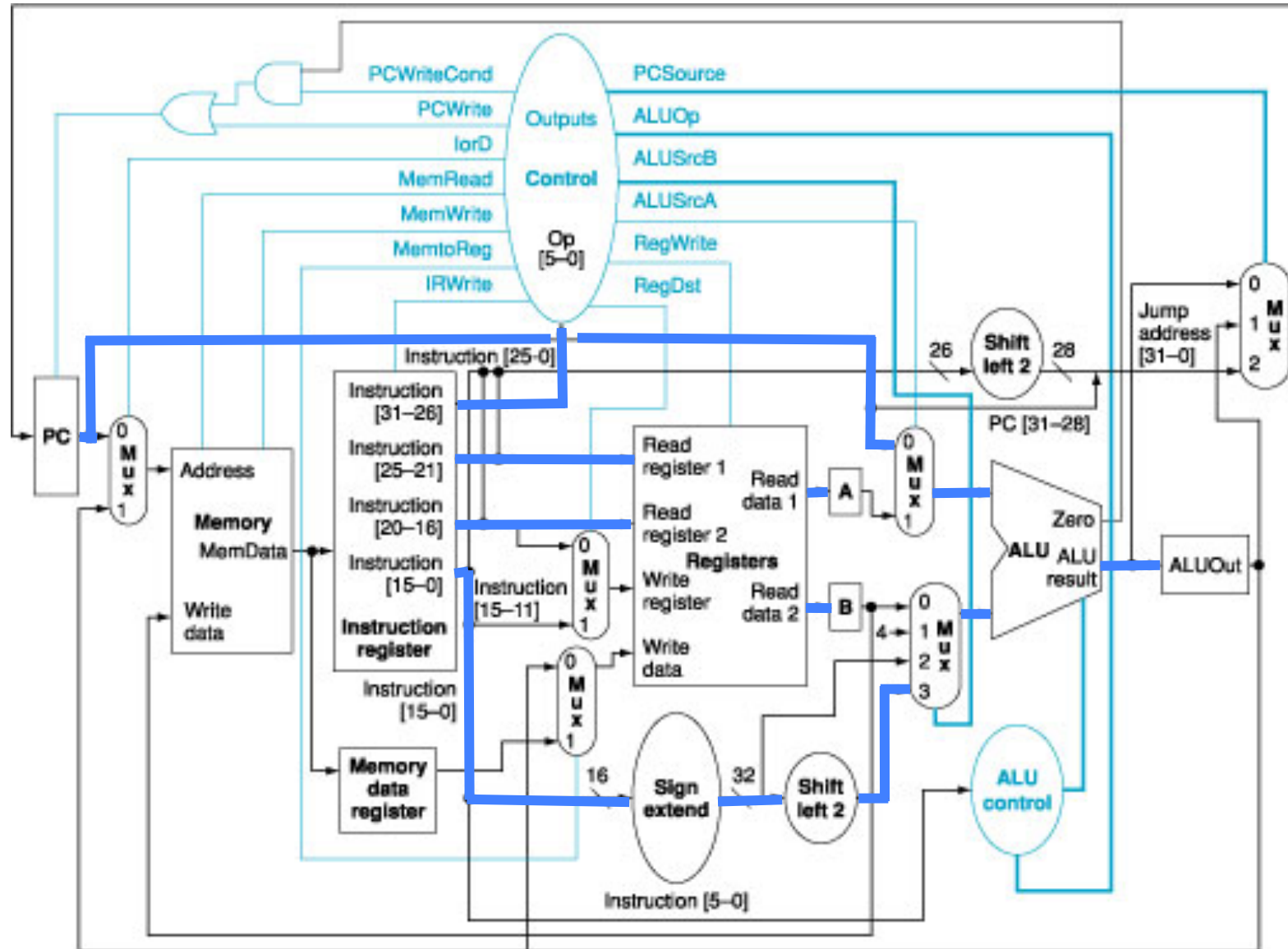
$ALUOut \leftarrow PC + sgnext(IR[15:0] \ll 2)$



Cycle 1 – instruction fetch (all)



Cycle 2 –instr decode & reg read (all)



What happens in each cycle – 3

3a. Memory address generation

$ALUOut \leq A + \text{sgnnext}(IR[15:0])$

3b. R-type arithmetic-logical instruction

$ALUOut \leq A \text{ op } B$

3c. Branch completion

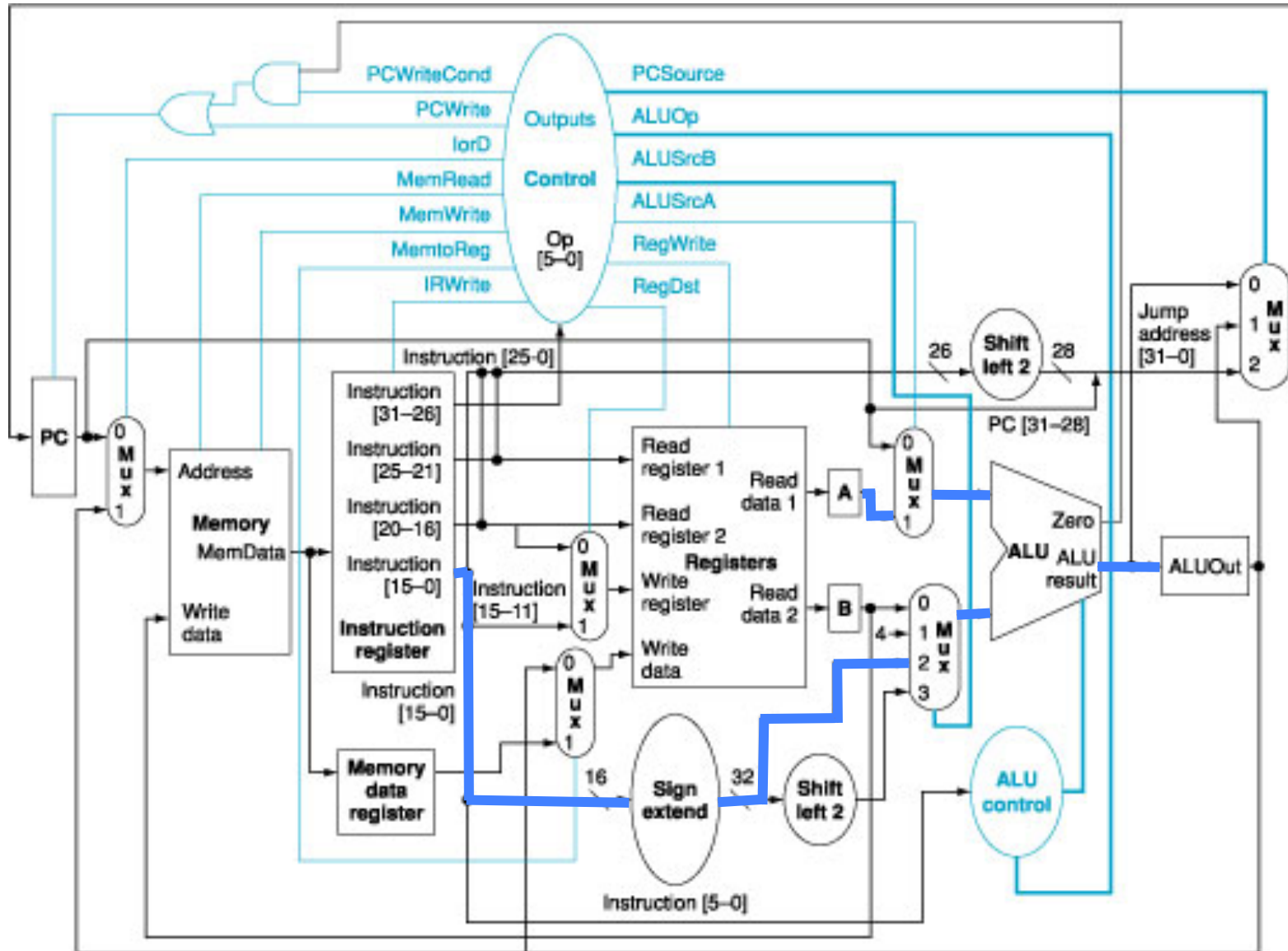
$\text{if } (A == B) \text{ PC} \leq ALUOut$

3d. Jump completion

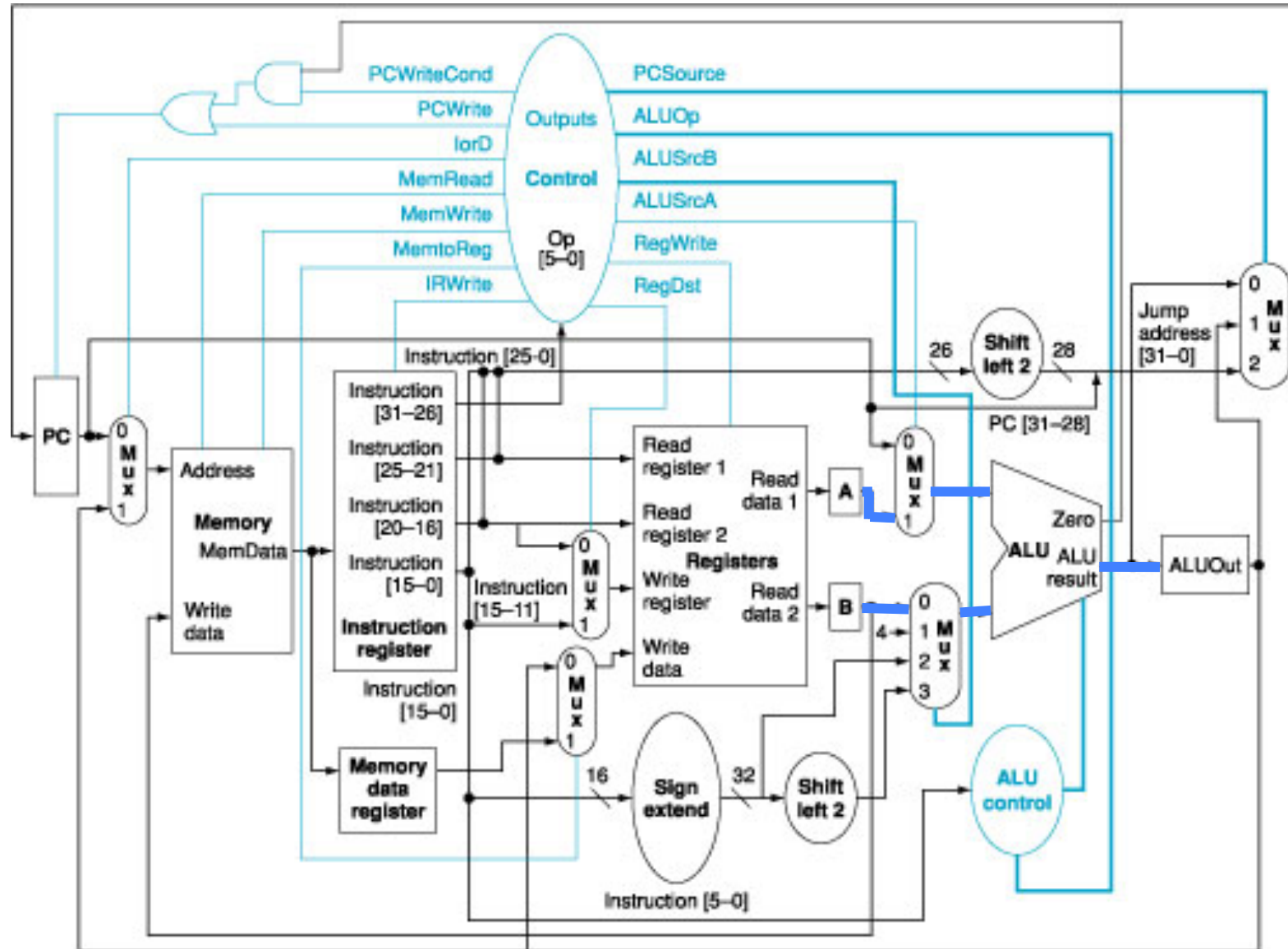
$PC \leq \{PC[31:28], IR[25:0], 2'b00\}$



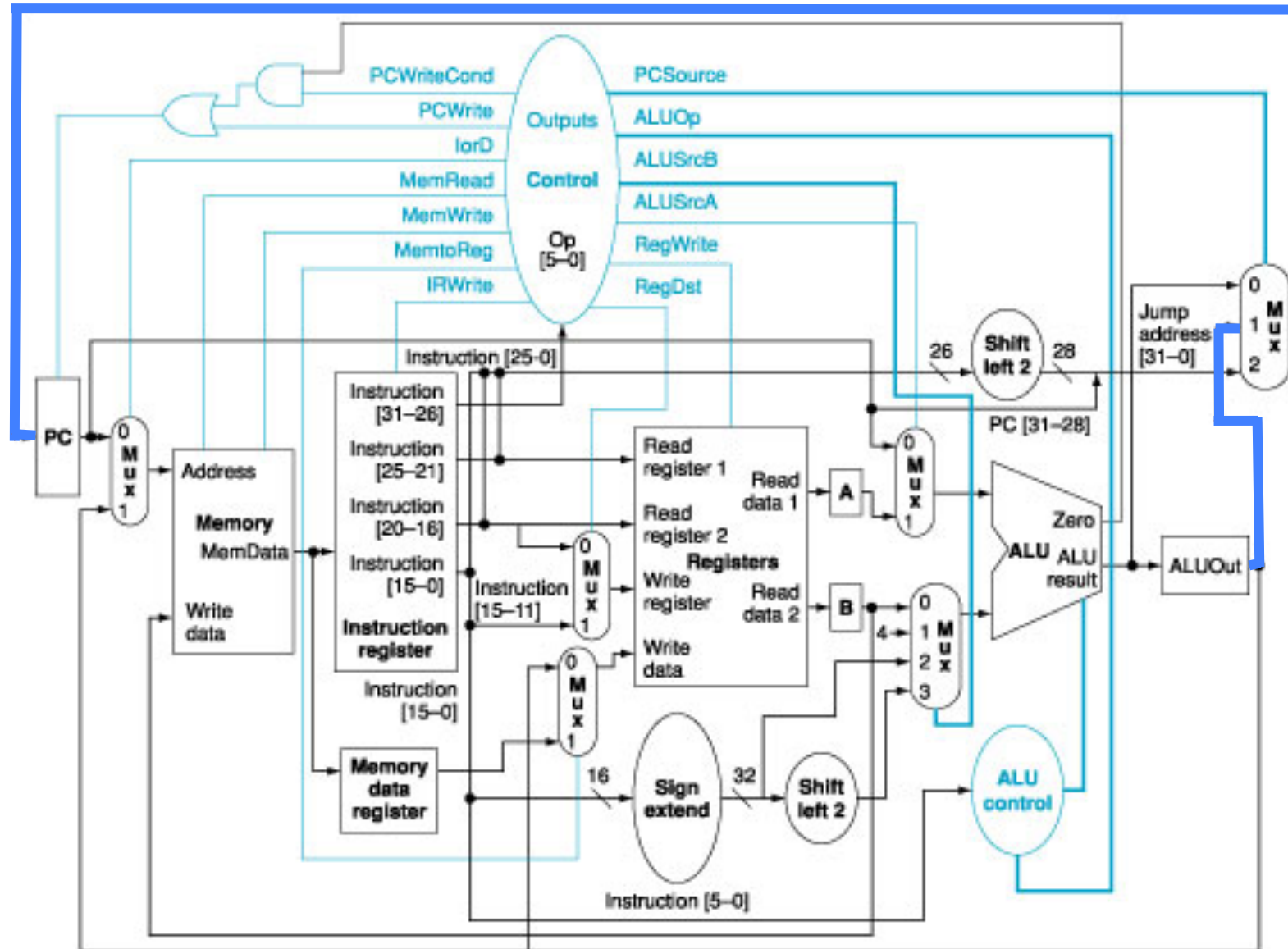
Cycle 3a: add imm arg (addi, lw, sw)



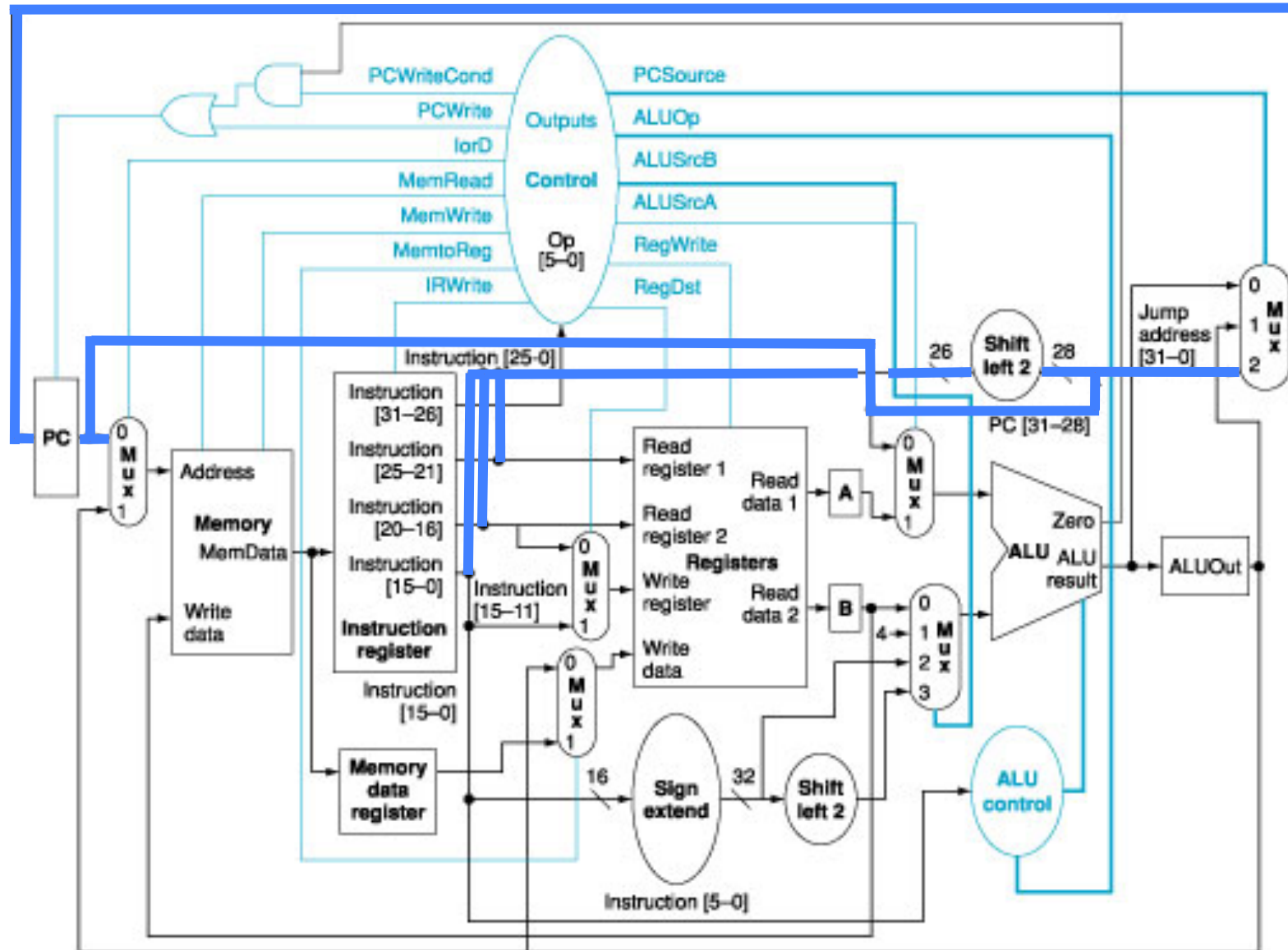
Cycle 3b: R-type ALU op (add, and)



Cycle 3c: Branch taken (beq, bne)



Cycle 3d: jump (j)



What happens in each cycle – 4

4a. Memory access (load)

$\text{MDR} \leftarrow \text{Mem}[\text{ALUOut}]$

4b. Memory access (store) & completion

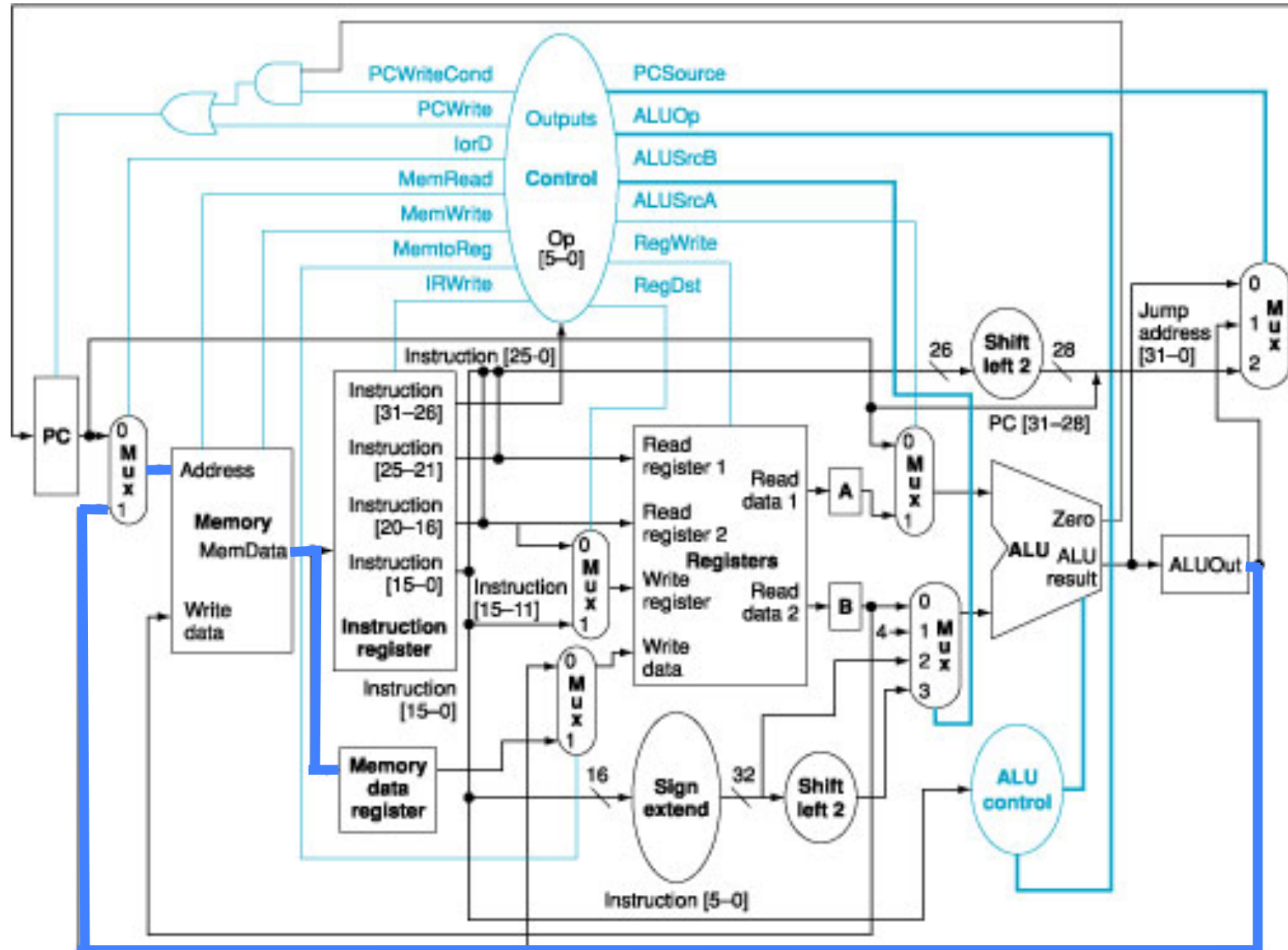
$\text{Mem}[\text{ALUOut}] \leftarrow \text{B}$

4c. R-type arith-logical instruction completion

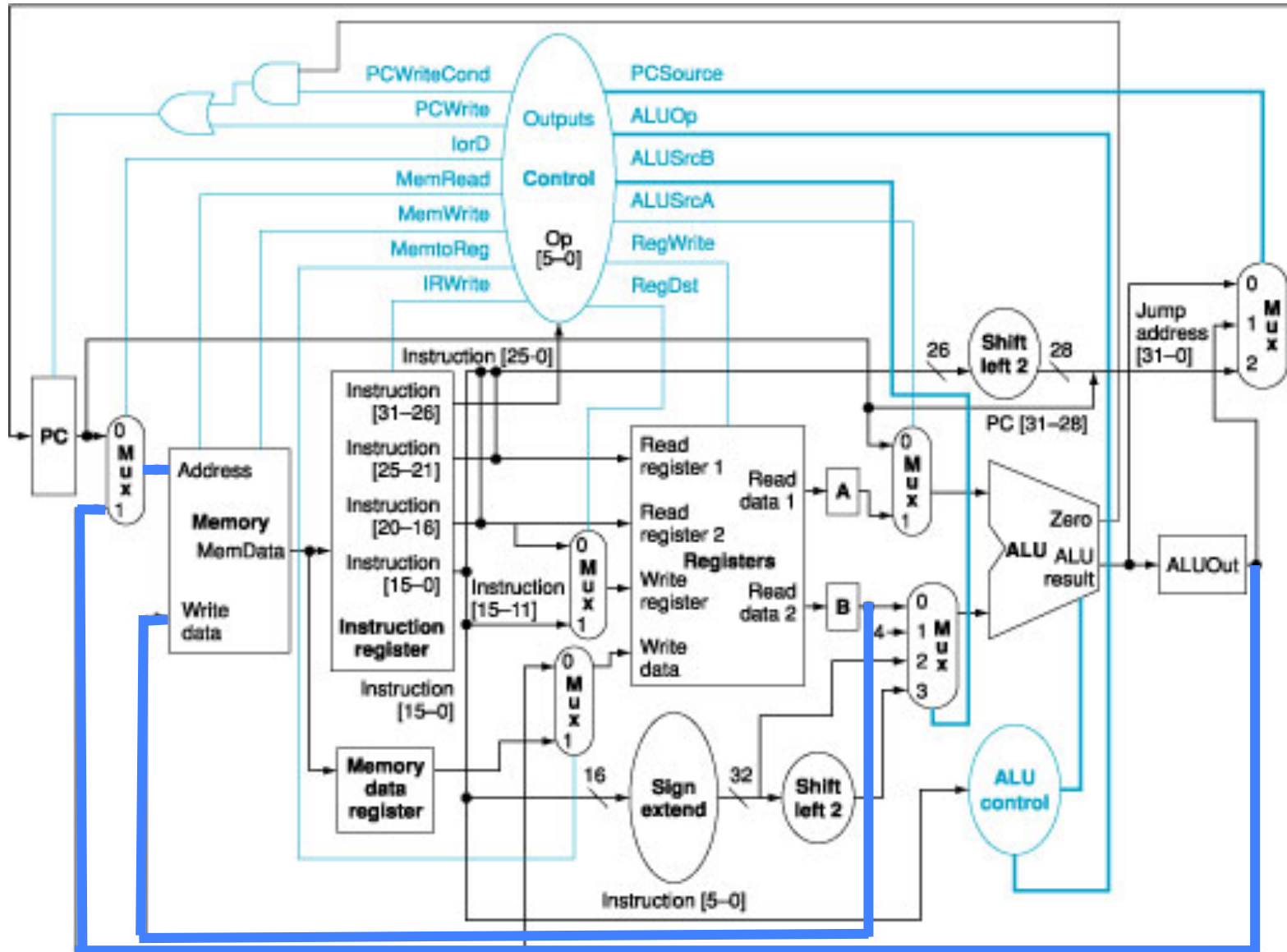
$\text{Reg}[\text{IR}[15:11]] = \text{ALUOut}$



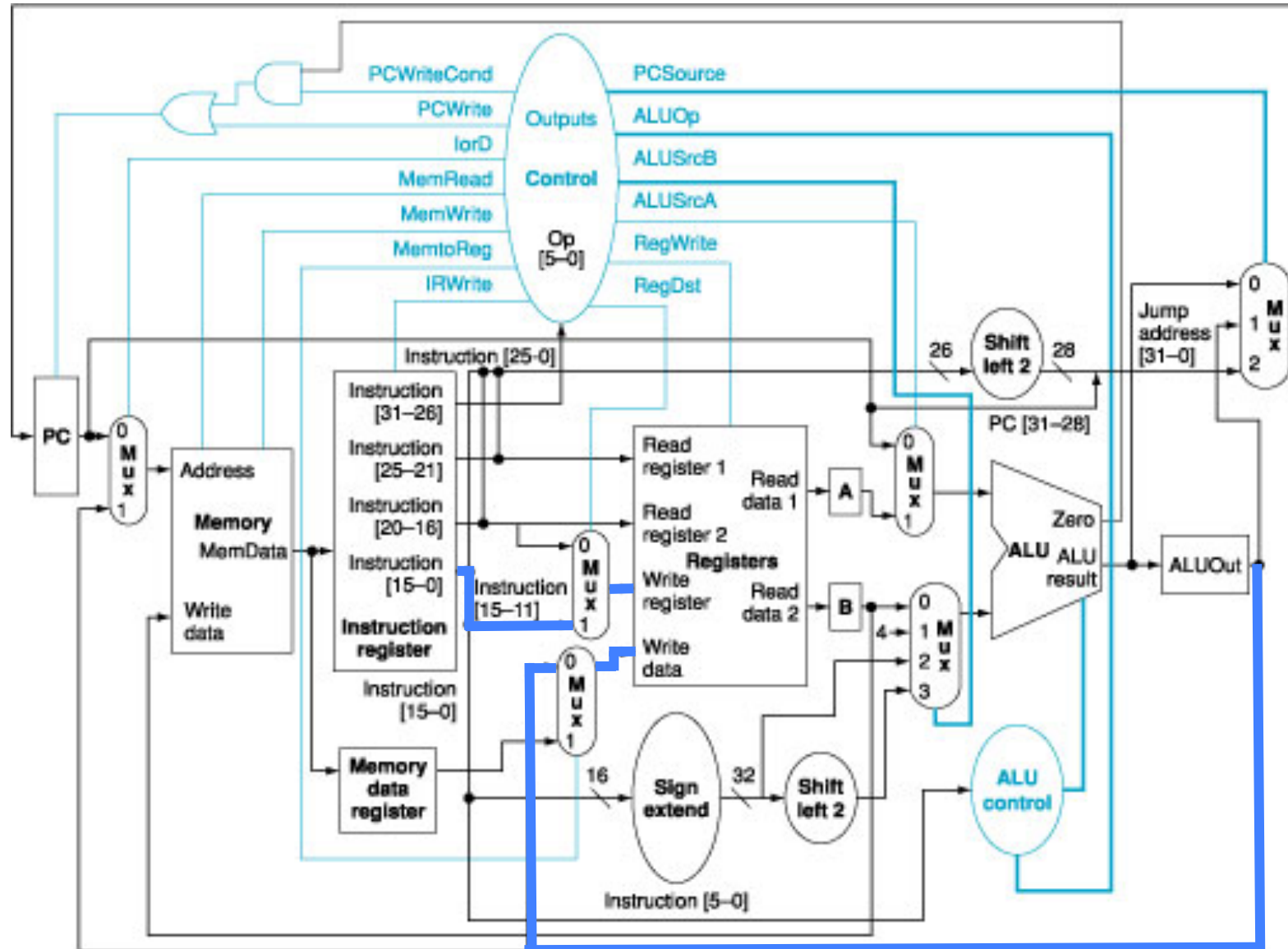
Cycle 4a: Load from mem (lw)



Cycle 4b: Store to mem (sw)



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



What happens in each cycle – 5

5. Load instruction completion

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



Cycle 5: save of loaded value (lw)

