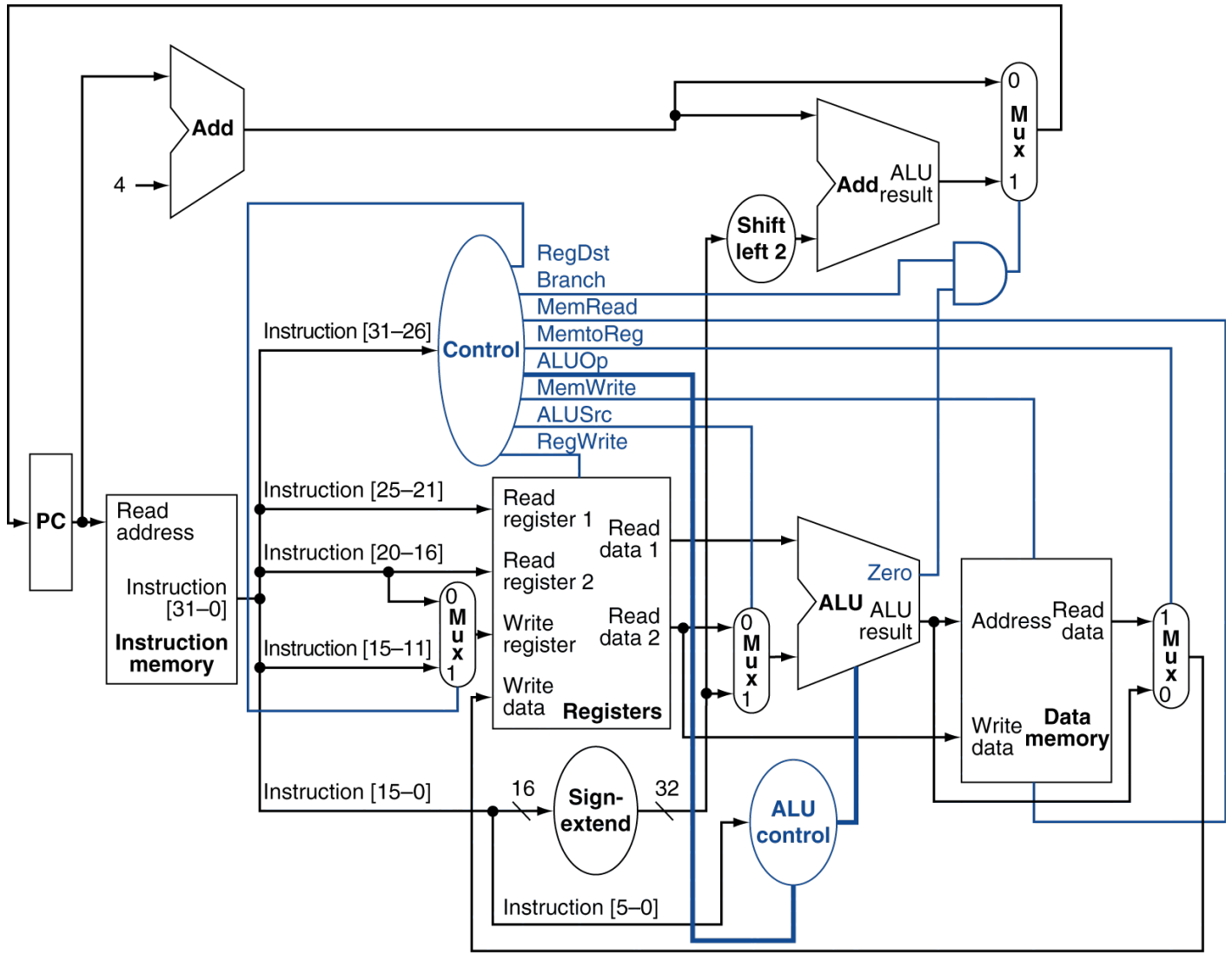
Inf2C - Computer Systems
Lecture 10
Processor Design – Multi-Cycle

Boris Grot

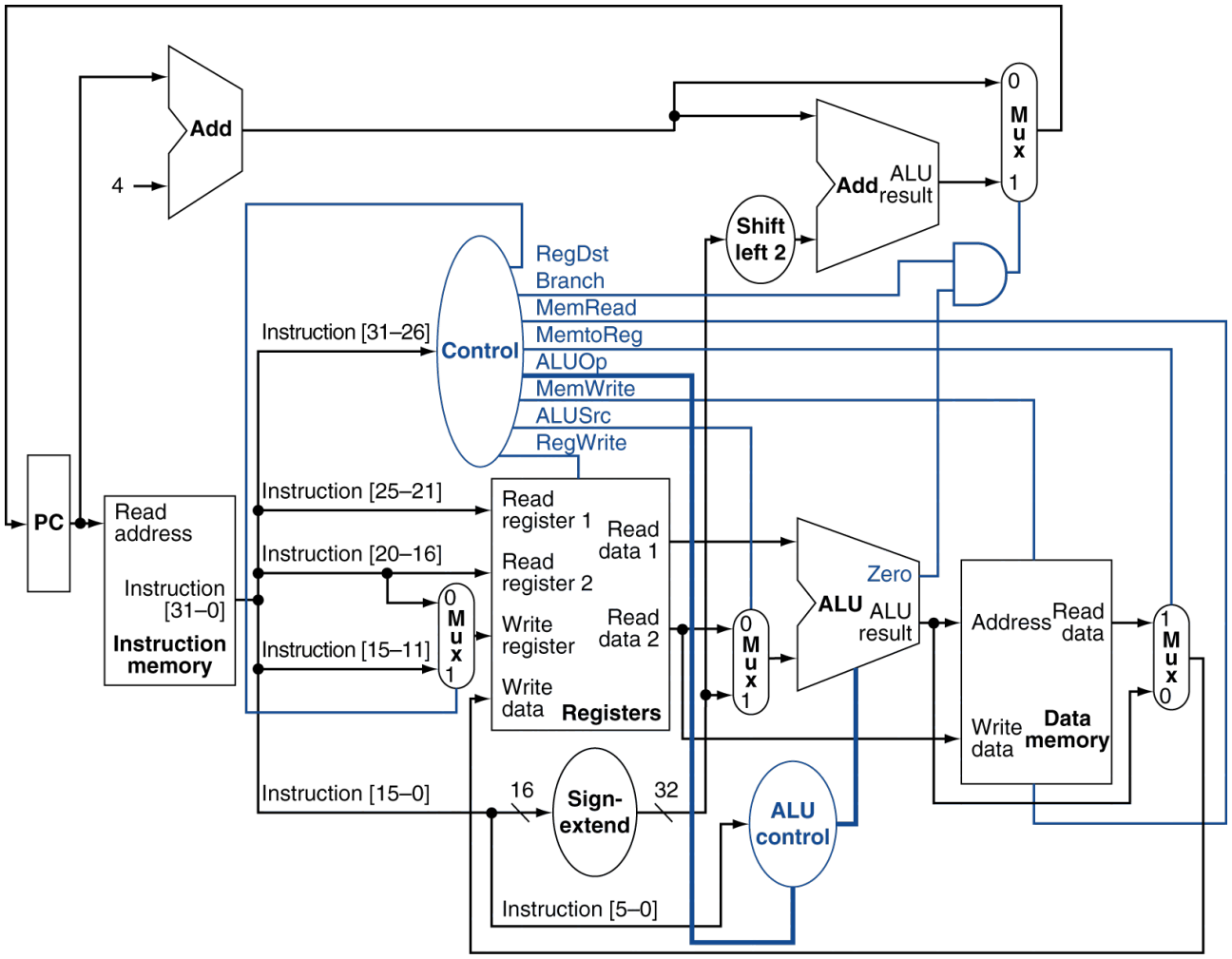
School of Informatics

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

Multi-cycle processor

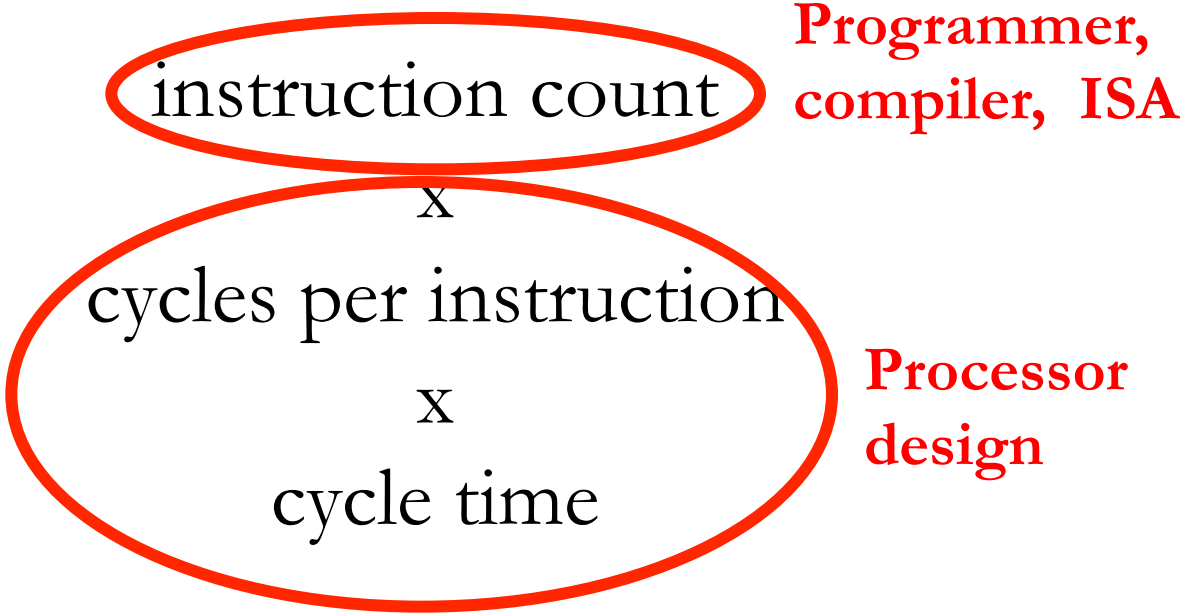
Basic idea:

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

End result: no instruction takes more time or uses more functional units than required

Measuring processor speed

Execution time is



Multi-Cycle Processor design guidelines

- Cycle time determined by the delay through the slowest functional unit
- At the 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
- Reuse functional units as much as possible
  - Multiplexors added to select the different inputs

As before

New!

Determine the components

Processor task

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

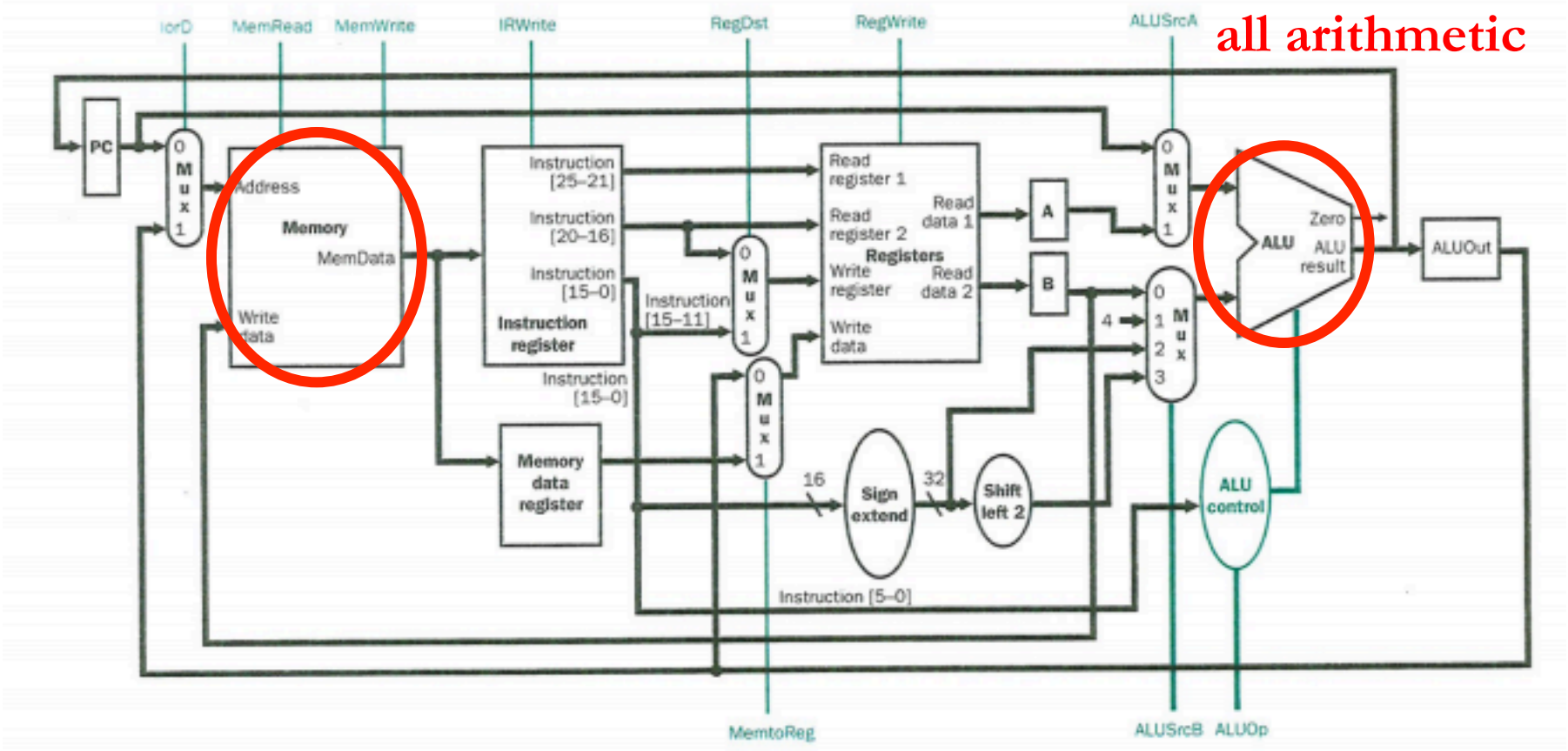
Component list

- PC register
- ~~Memory (instructions)~~
- ~~Adder: PC+4~~
- Register file
- ALU
- Memory (data)
- ~~Adder: branch target~~

Multi-cycle datapath (overview)

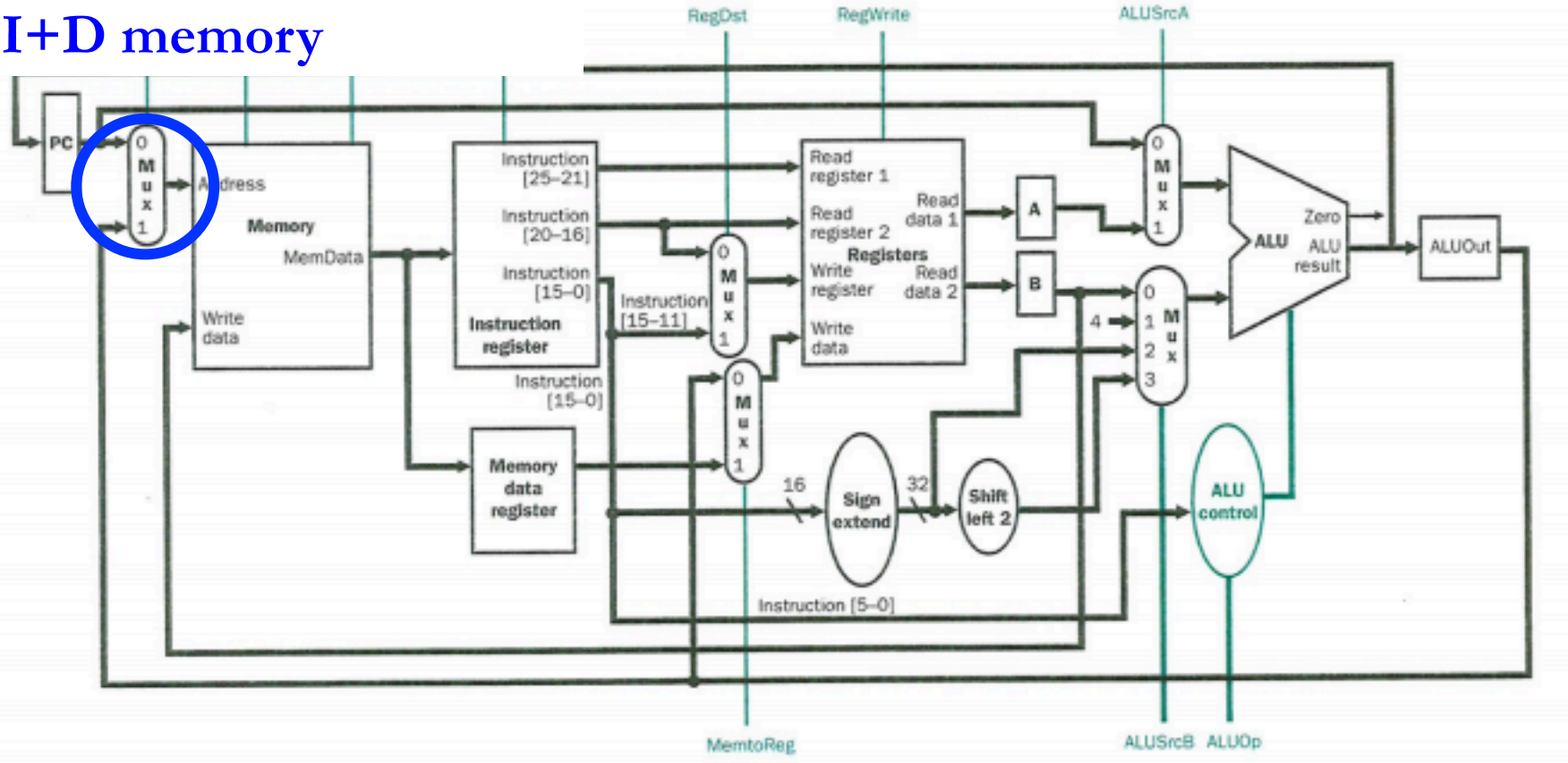
One memory for both Instruction & Data

One ALU for all arithmetic

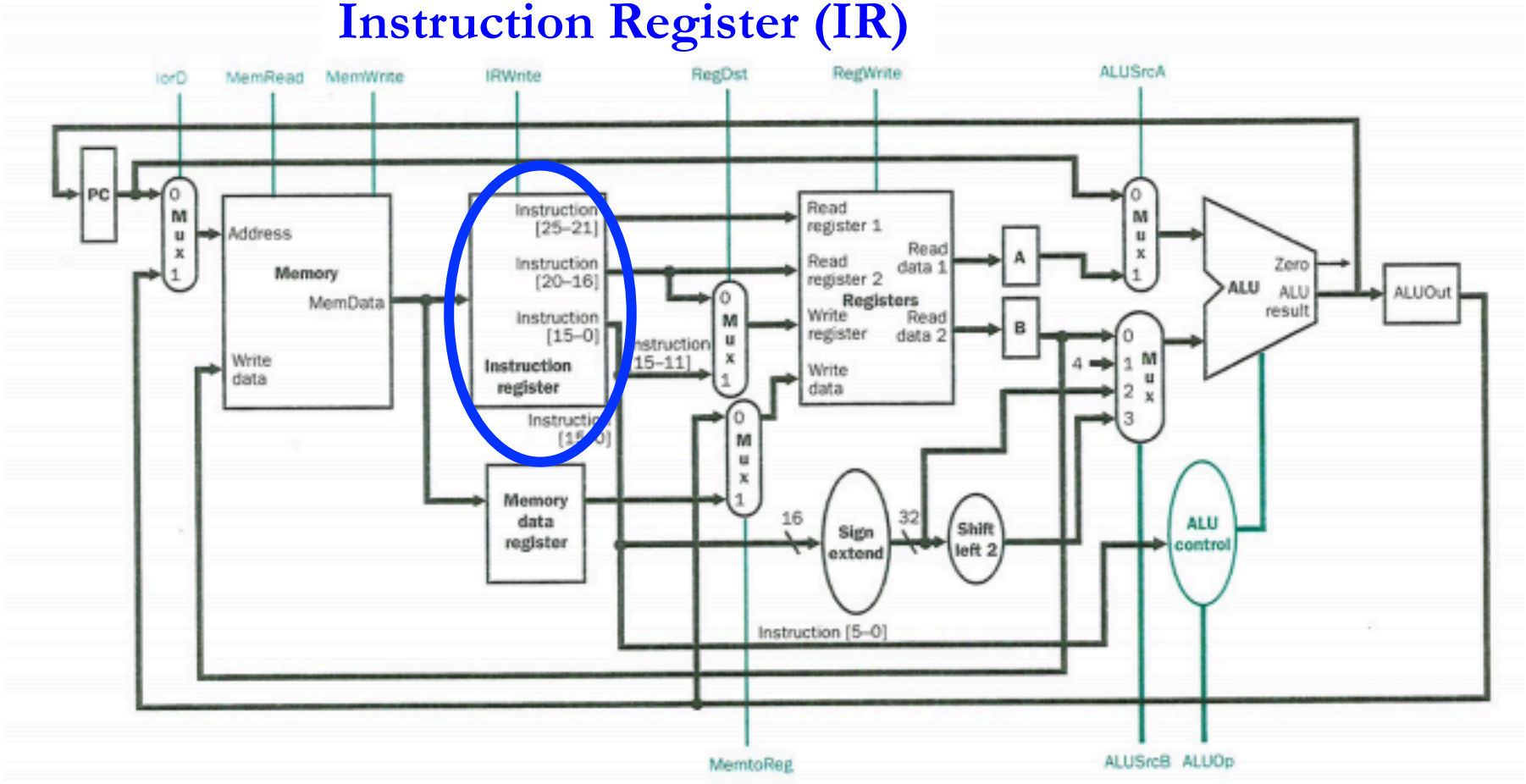


Multi-cycle datapath (overview)

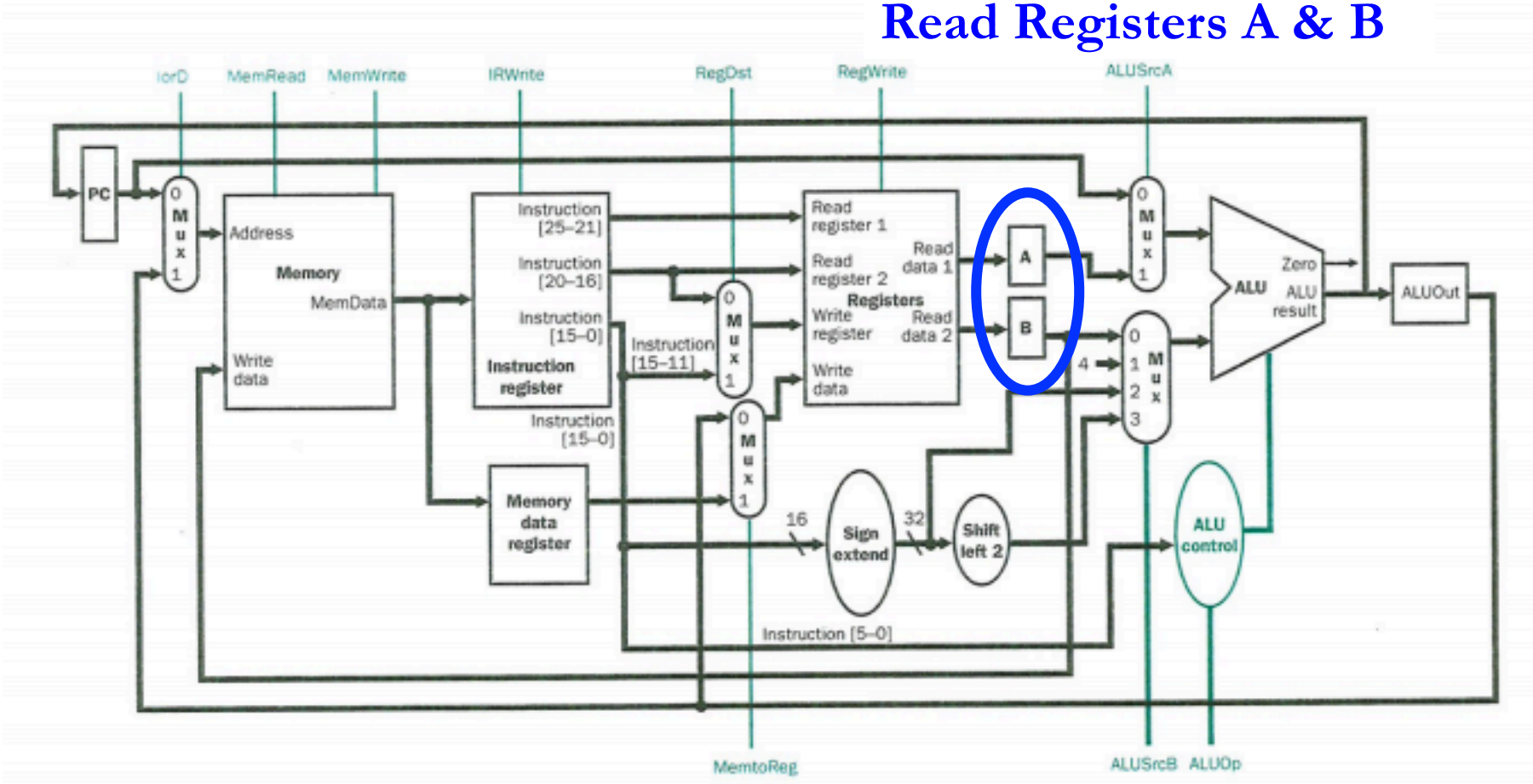
Select address input to I+D memory



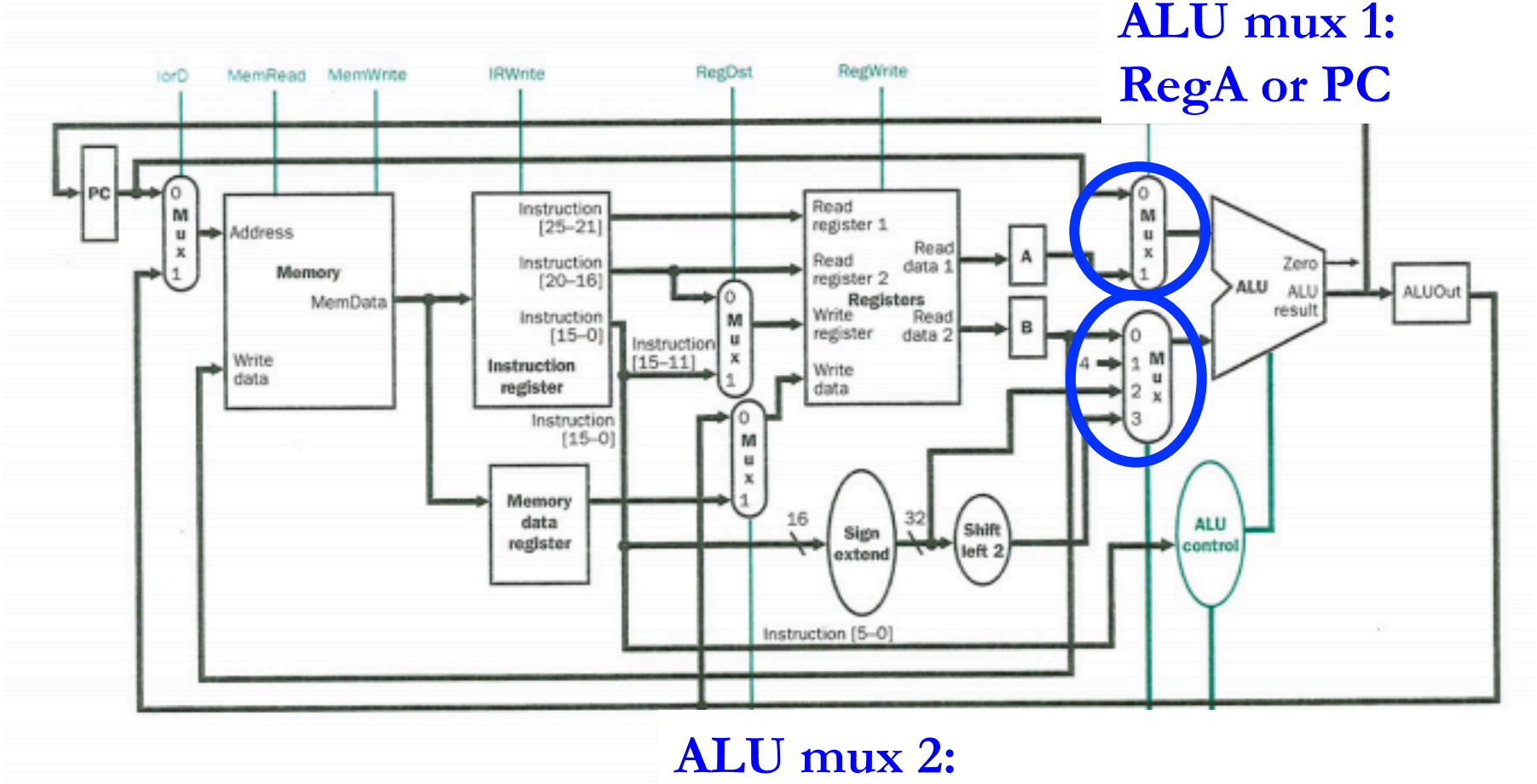
Multi-cycle datapath (overview)



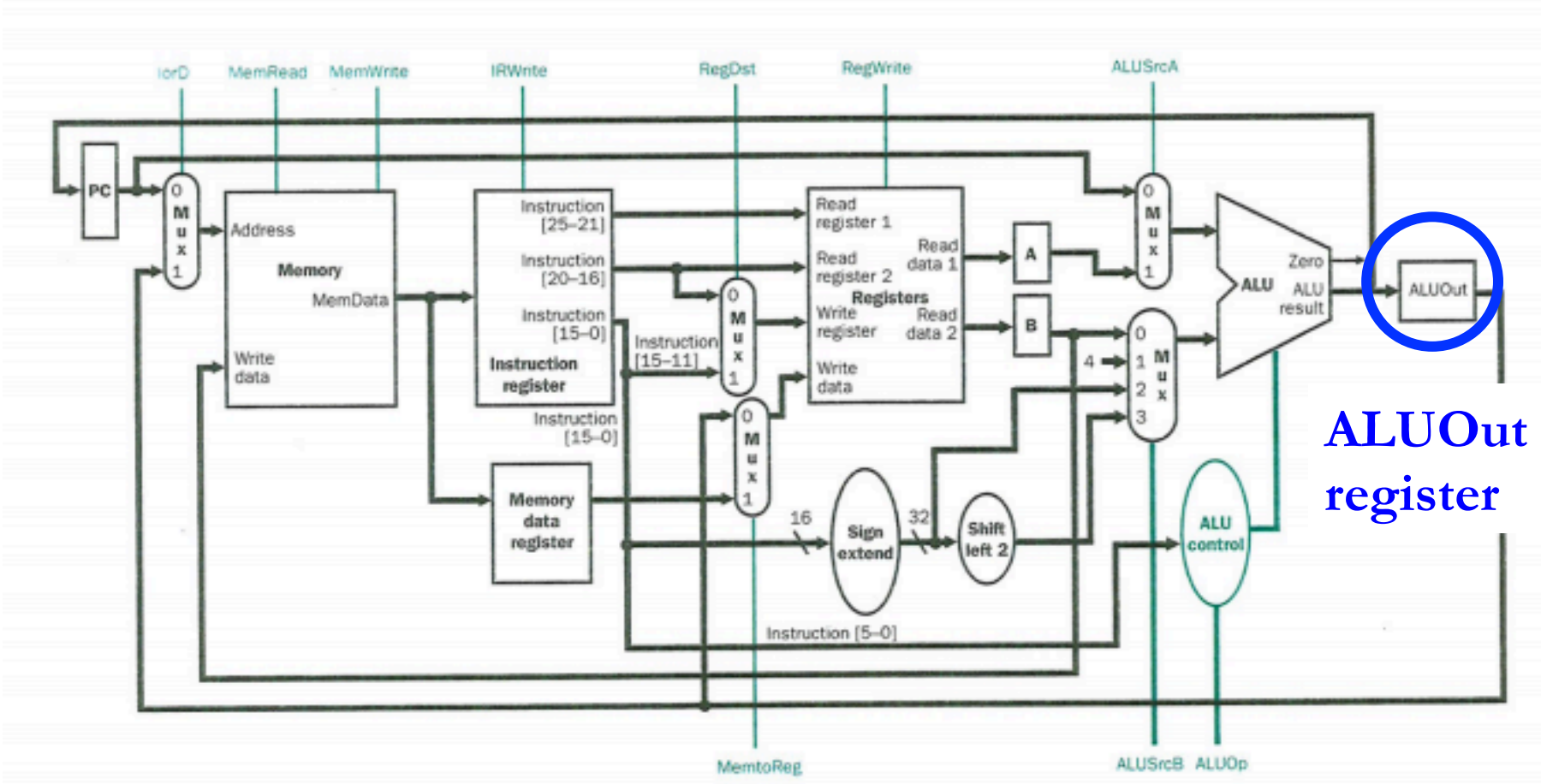
Multi-cycle datapath (overview)



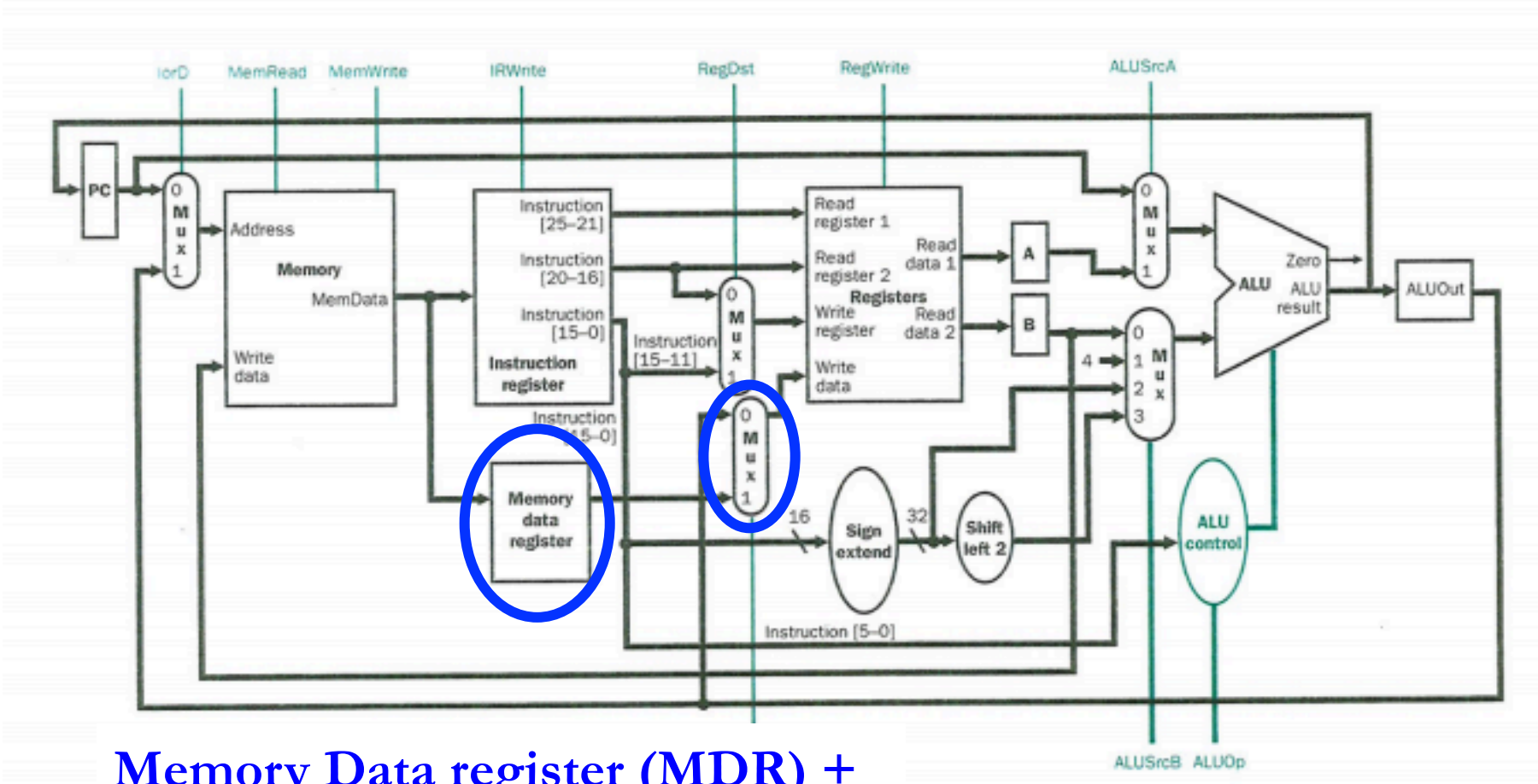
Multi-cycle datapath (overview)



Multi-cycle datapath (overview)

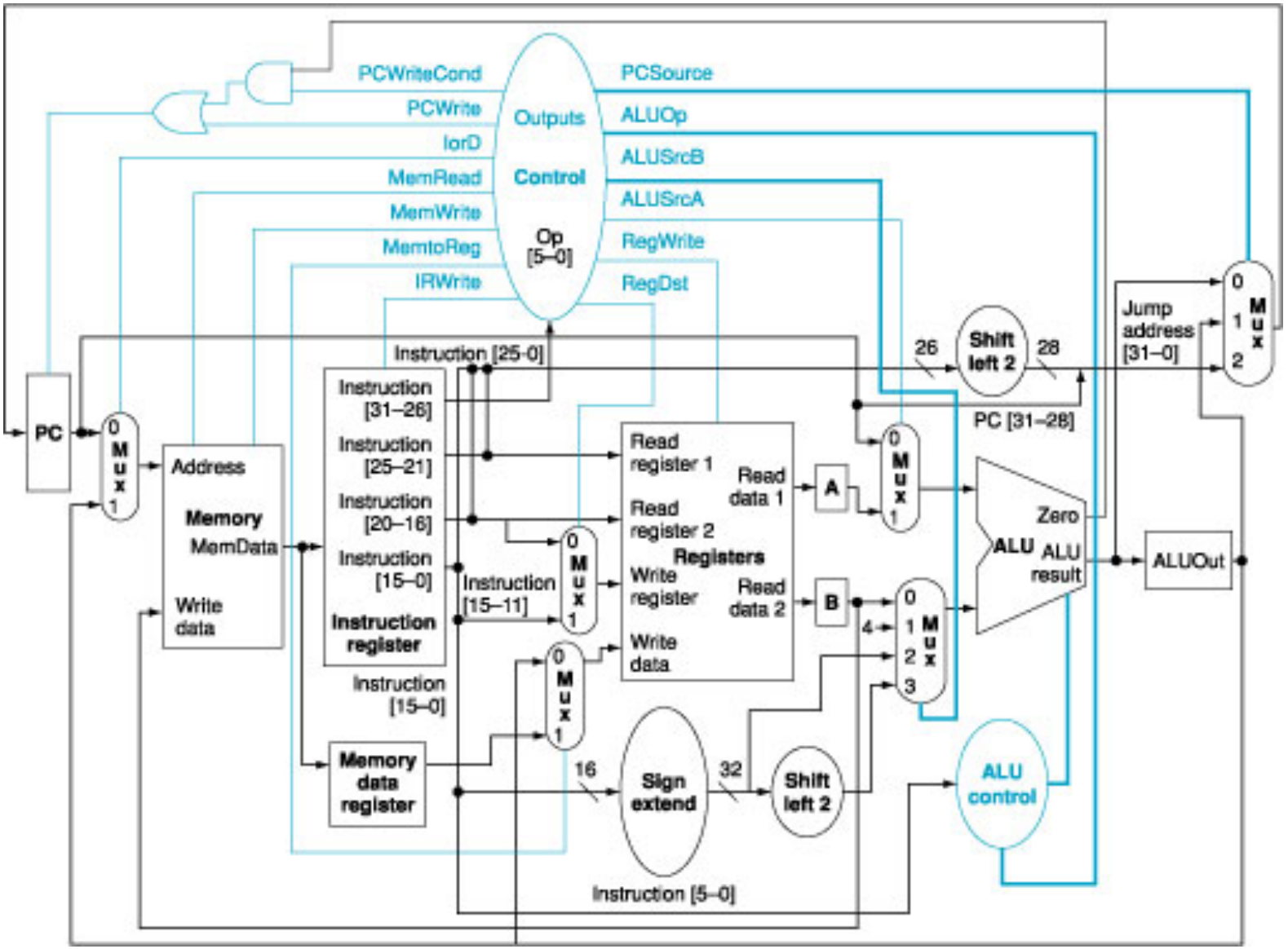


Multi-cycle datapath (overview)



Memory Data register (MDR) + Write Data mux

Multi-cycle datapath (with control)



Multi-cycle processor control signals (1-bit)

Signal name	Function
RegDst	Write register in the register file (*rs* 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 data to be written to the register file (*ALUout* 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

Multi-cycle processor control signals (2-bit)

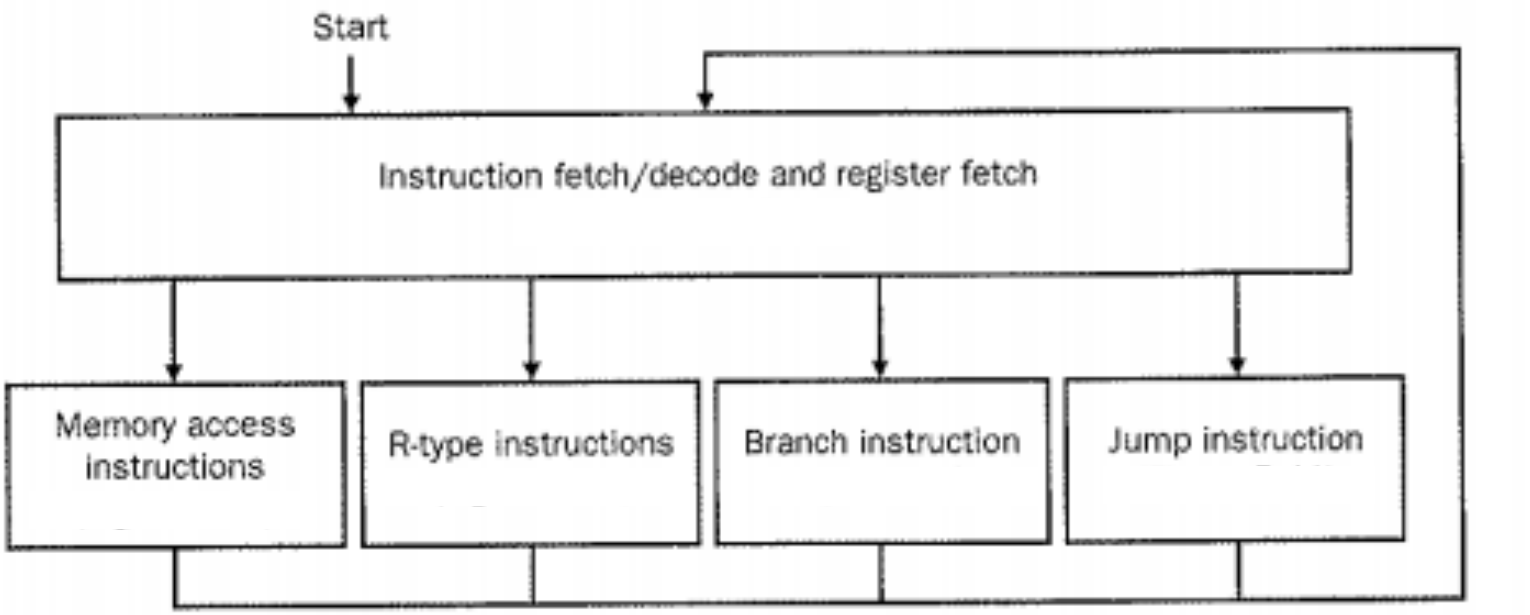
Signal name	Value	Function
ALUSrcB	00	Second ALU input comes from B register
	01	Second ALU input is a constant 4
	10	Second ALU input is sign-extended lower 16 bits of IR
	11	Second ALU input is sign-extended lower 16-bits of IR shifted left by 2
ALUOp	00	ALU performs an Add
	01	ALU performs a Subtract
	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

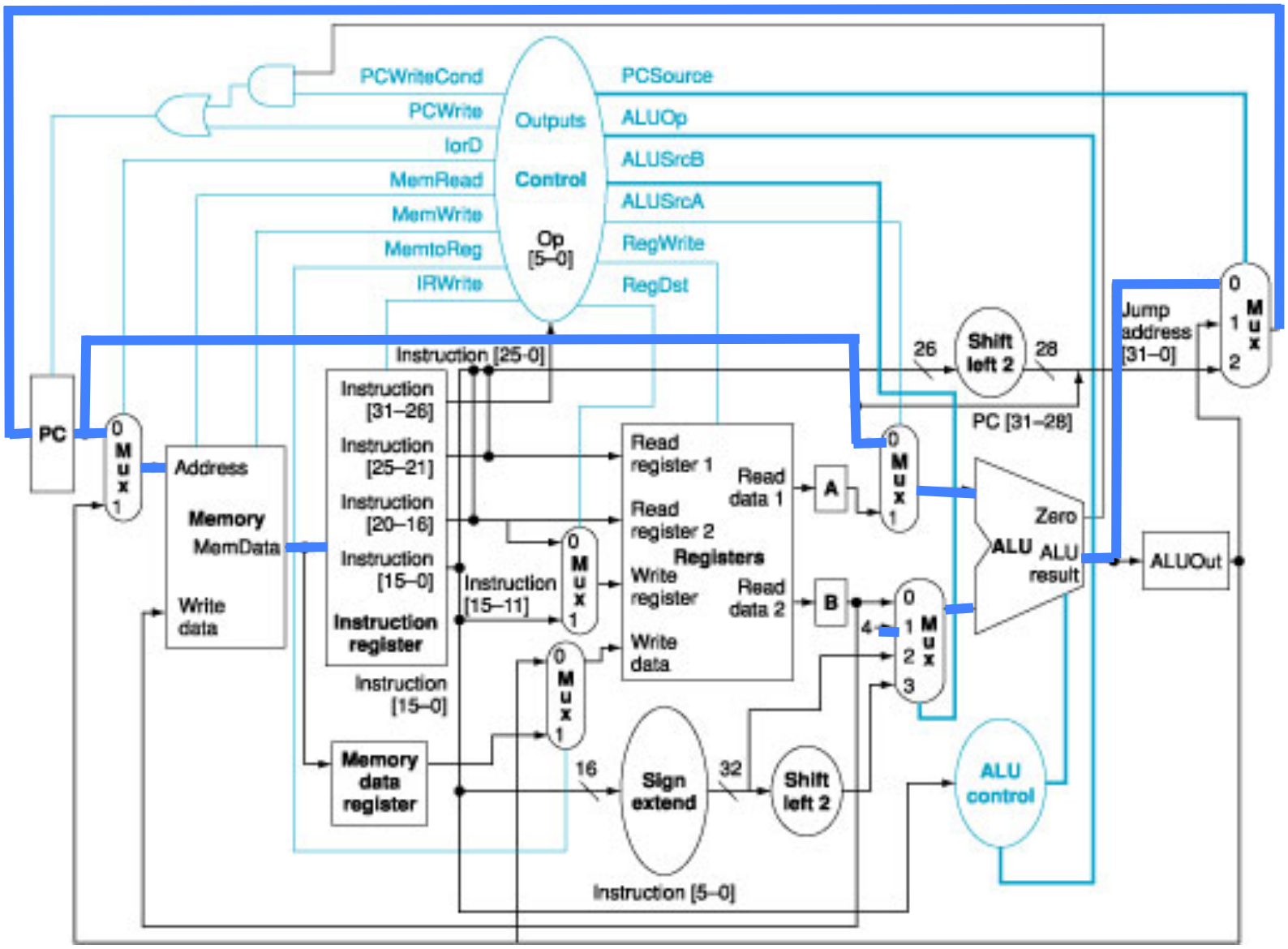
1. Instruction fetch

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

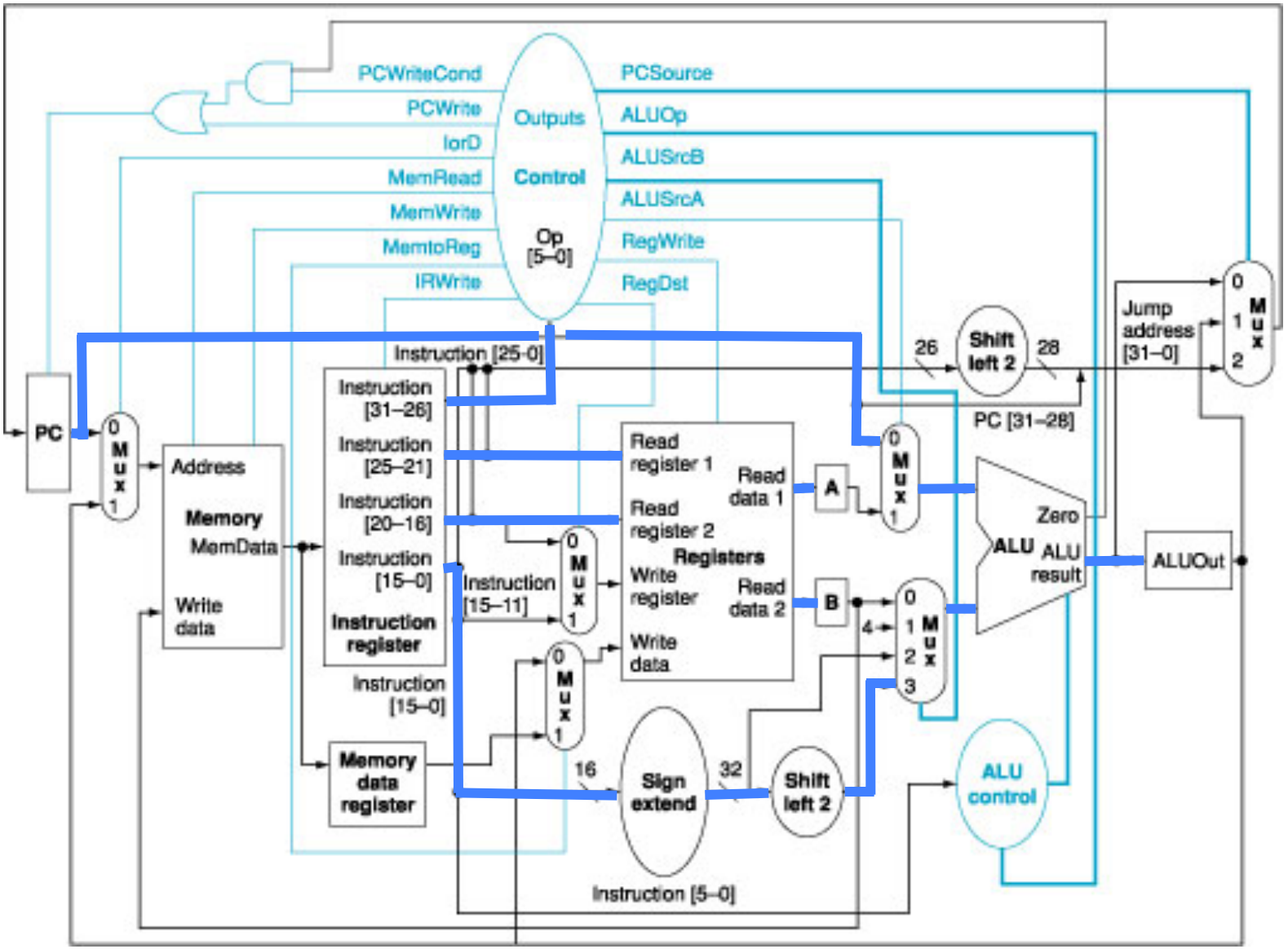
2. Instruction decode and register fetch

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

Cycle 1 – instruction fetch (all)



Cycle 2 –instr decode & reg read (all)



What happens in each cycle – 3

3a. Memory address generation

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

3b. R-type arithmetic-logical instruction

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

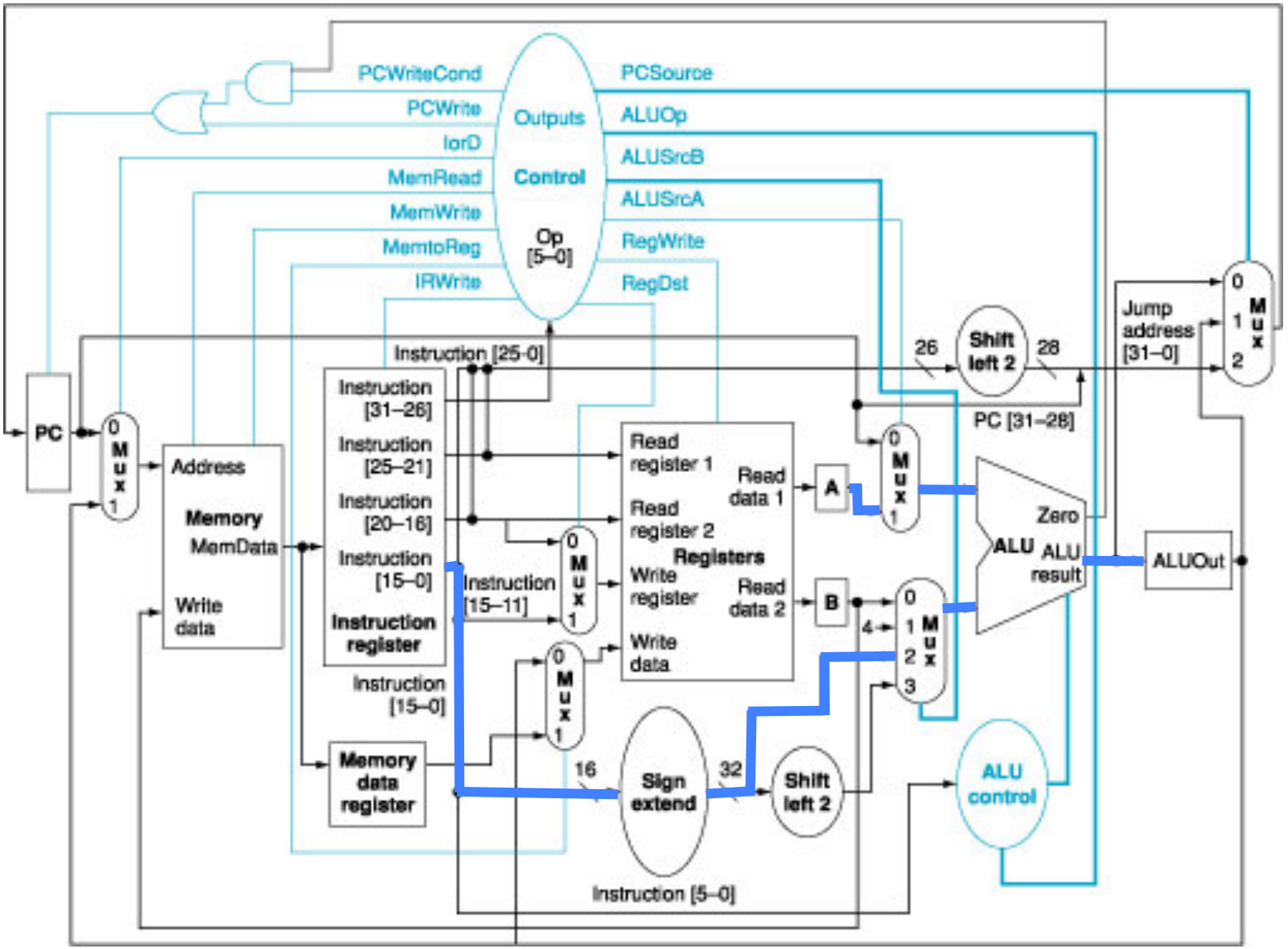
3c. Branch completion

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

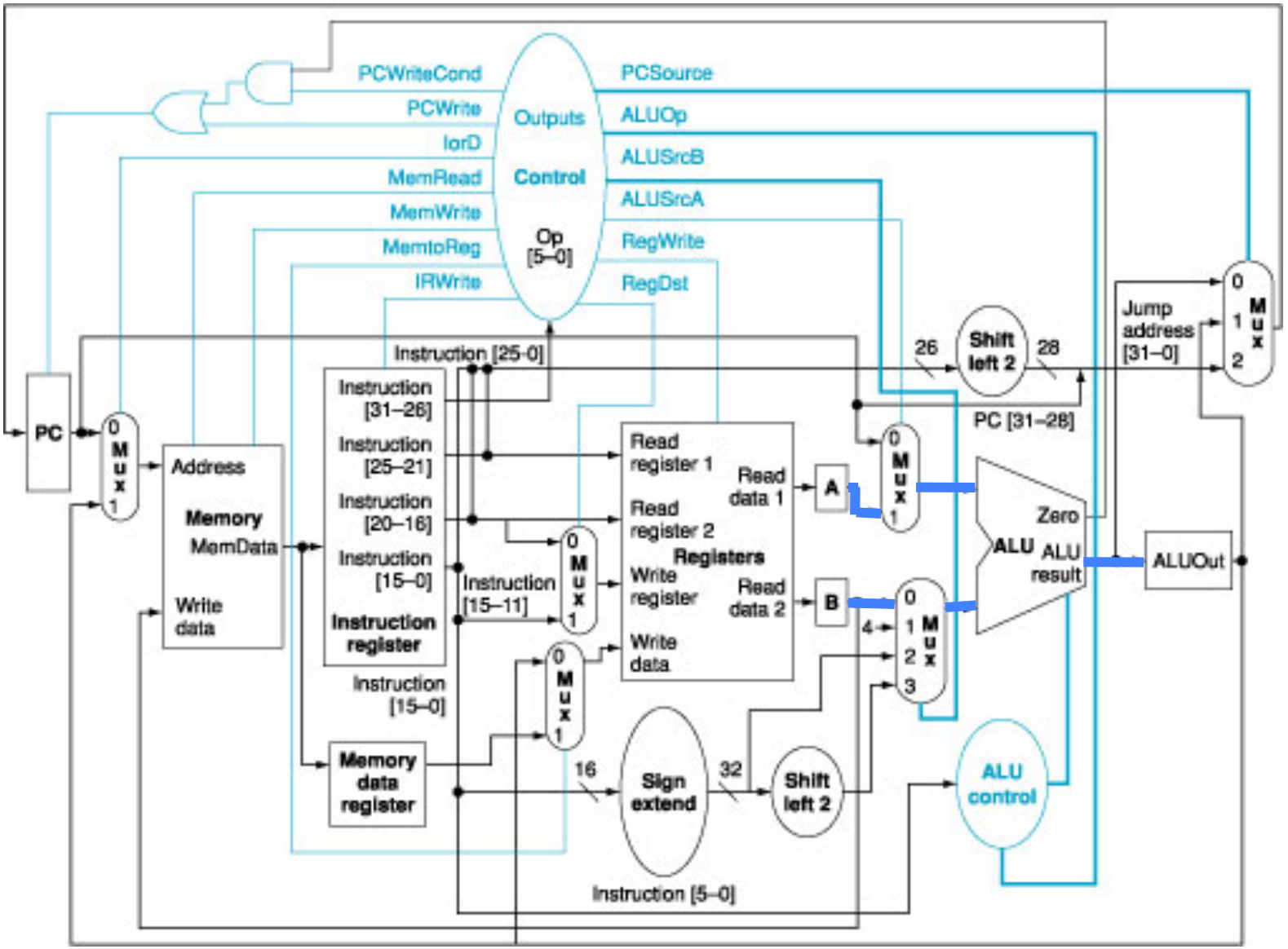
3d. Jump completion

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

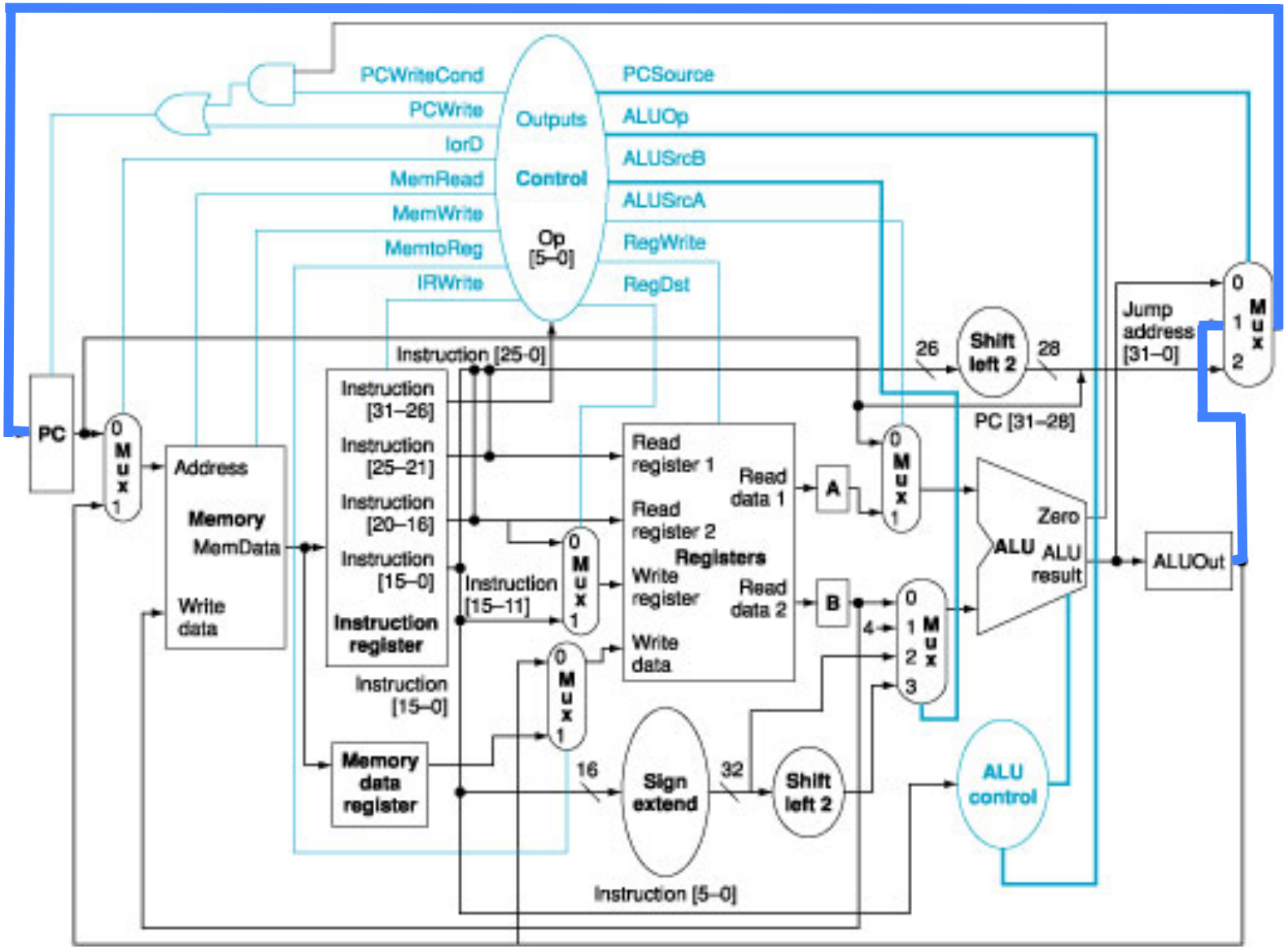
Cycle 3a: add imm arg (addi, lw, sw)



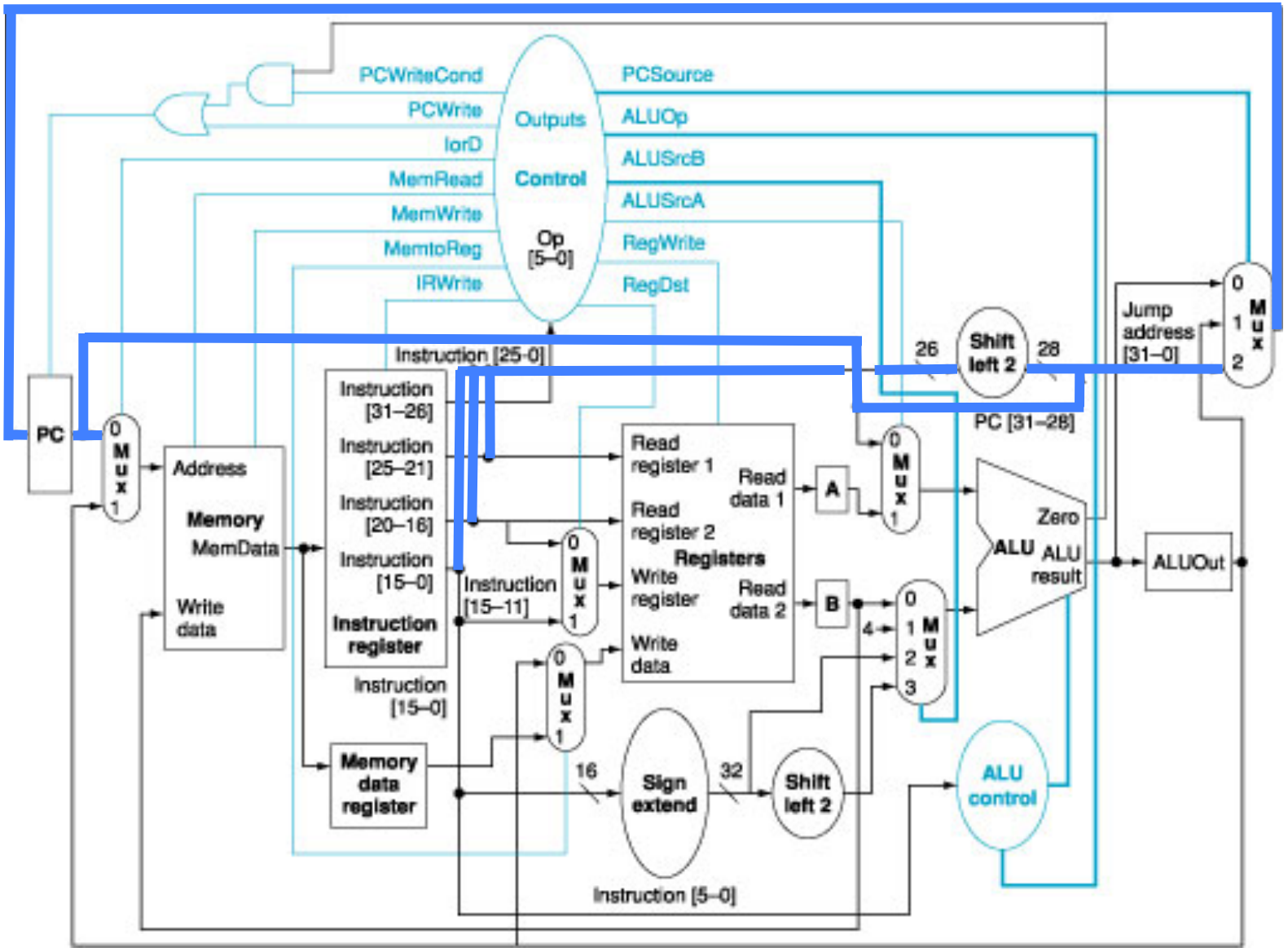
Cycle 3b: R-type ALU op (add, and)



Cycle 3c: Branch taken (beq, bne)



Cycle 3d: jump (j)



What happens in each cycle – 4

4a. Memory access (load)

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

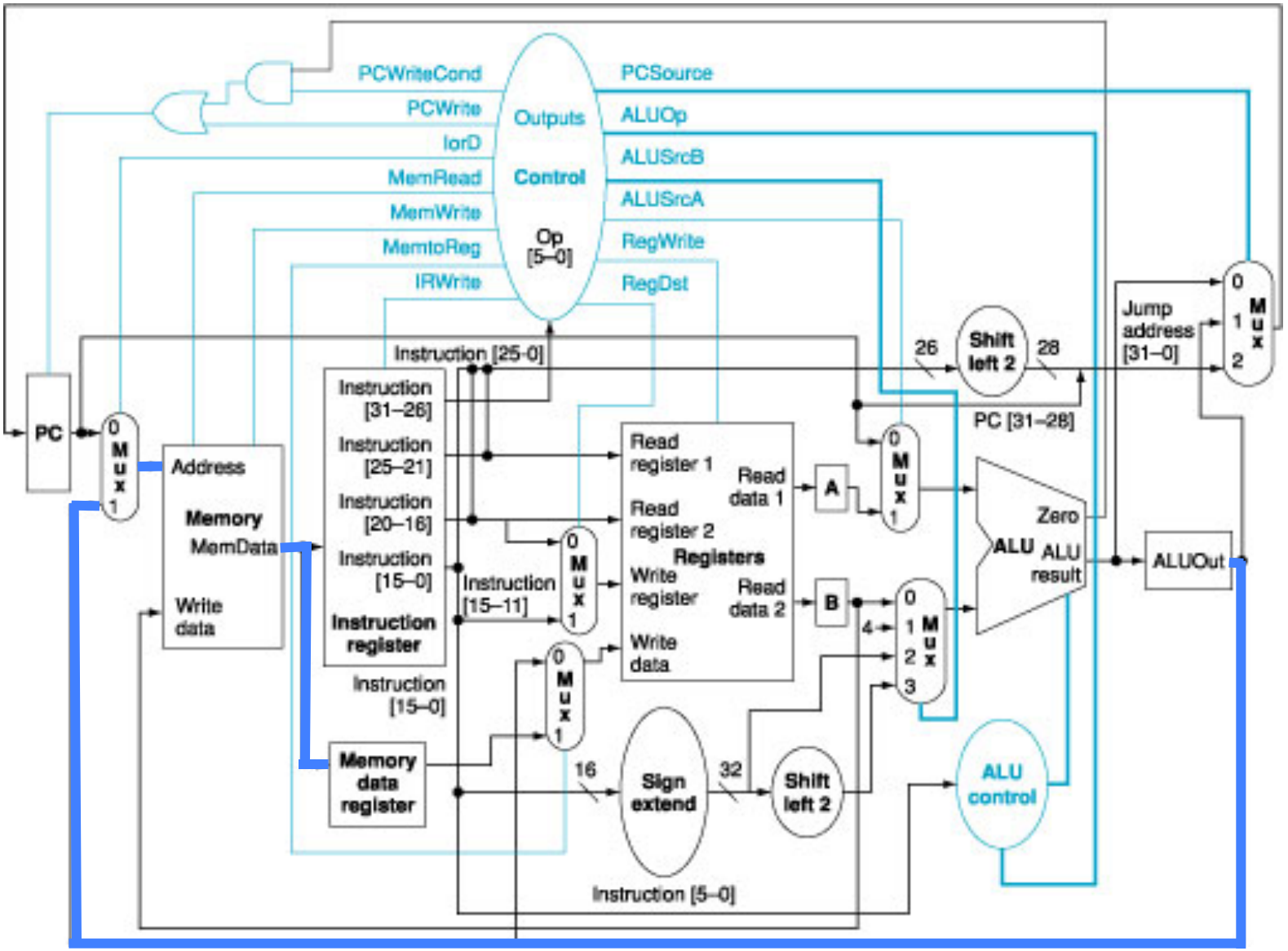
4b. Memory access (store) & completion

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

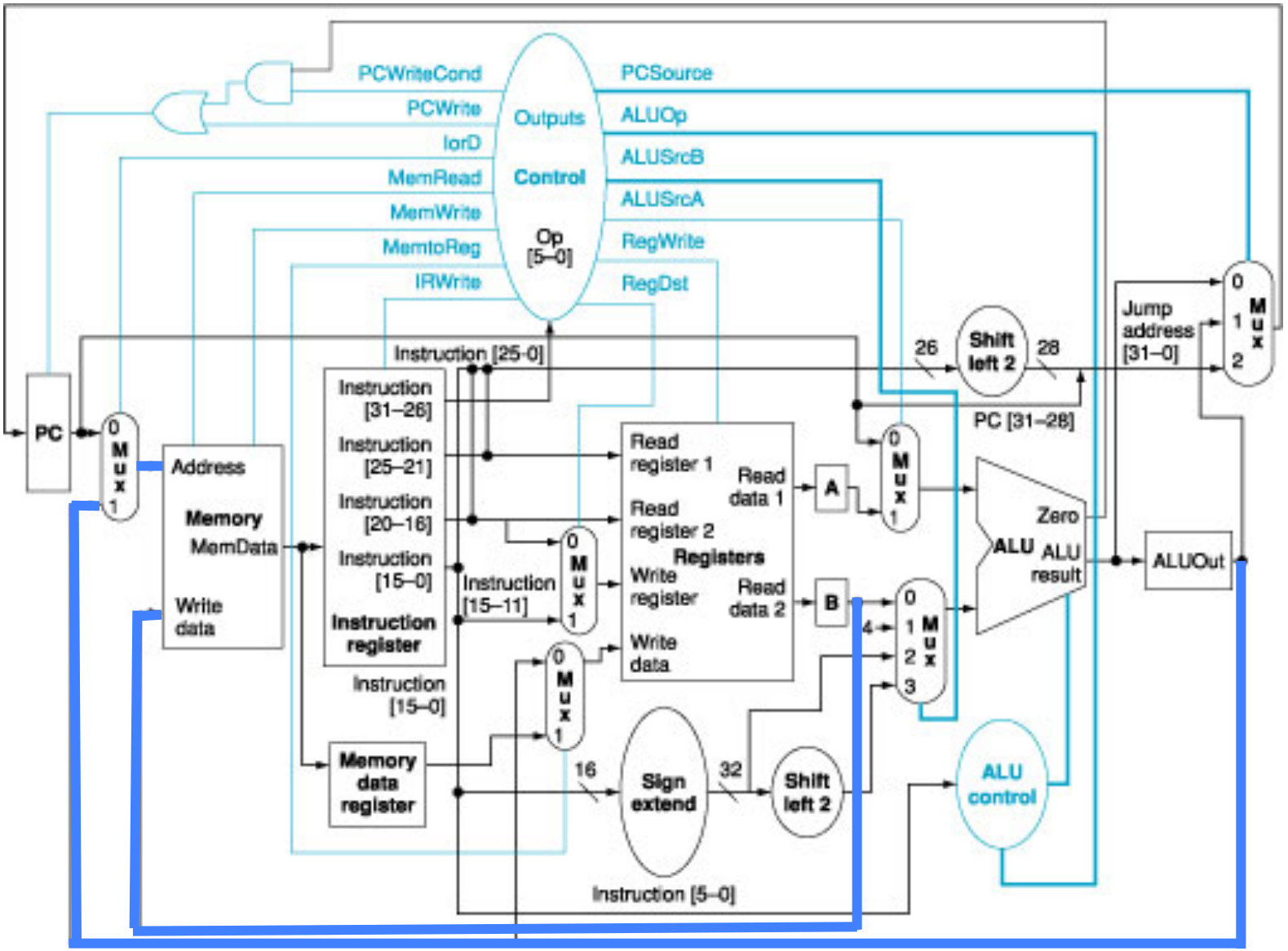
4c. R-type arith-logical instruction completion

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

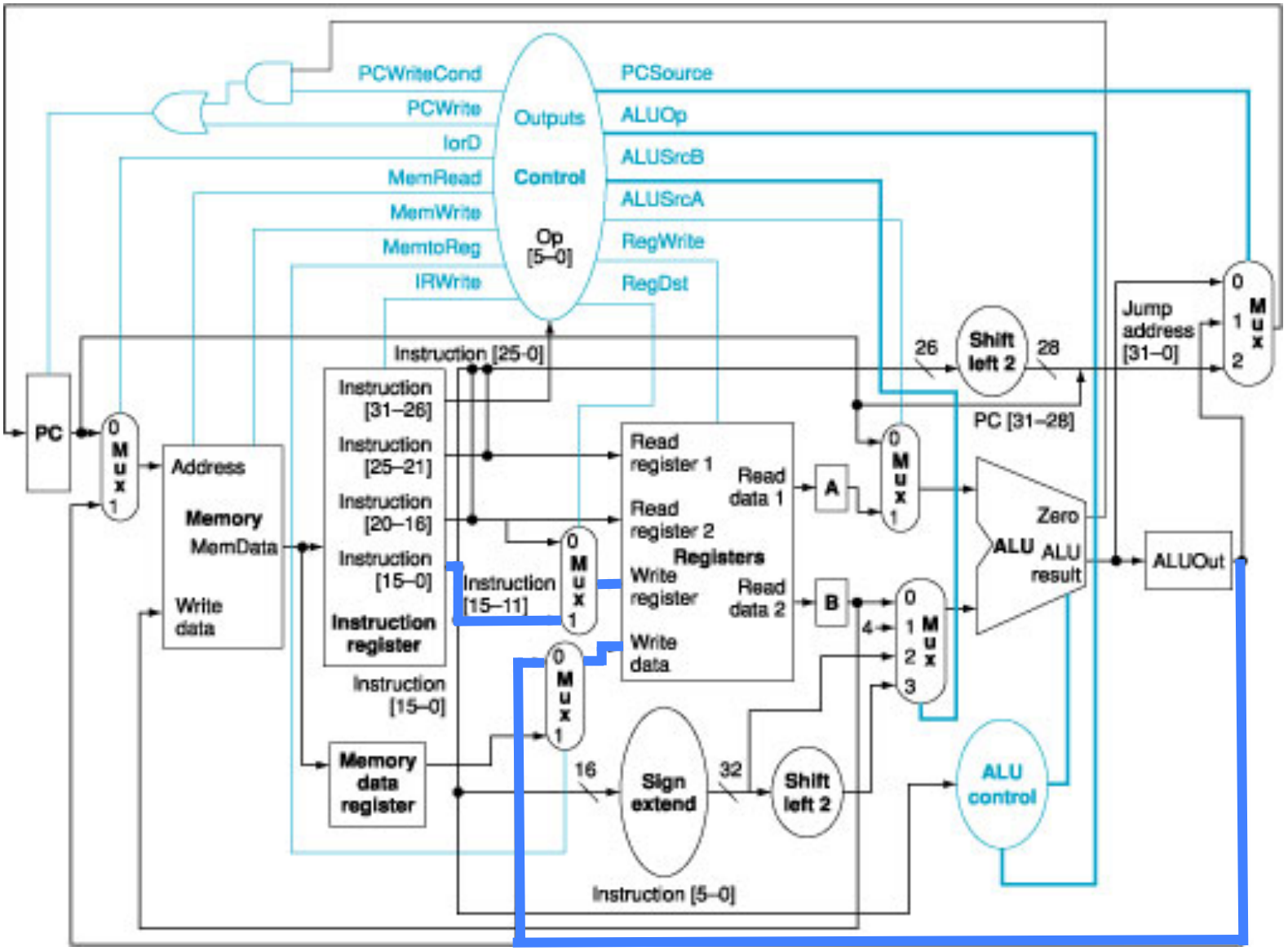
Cycle 4a: Load from mem (lw)



Cycle 4b: Store to mem (sw)



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

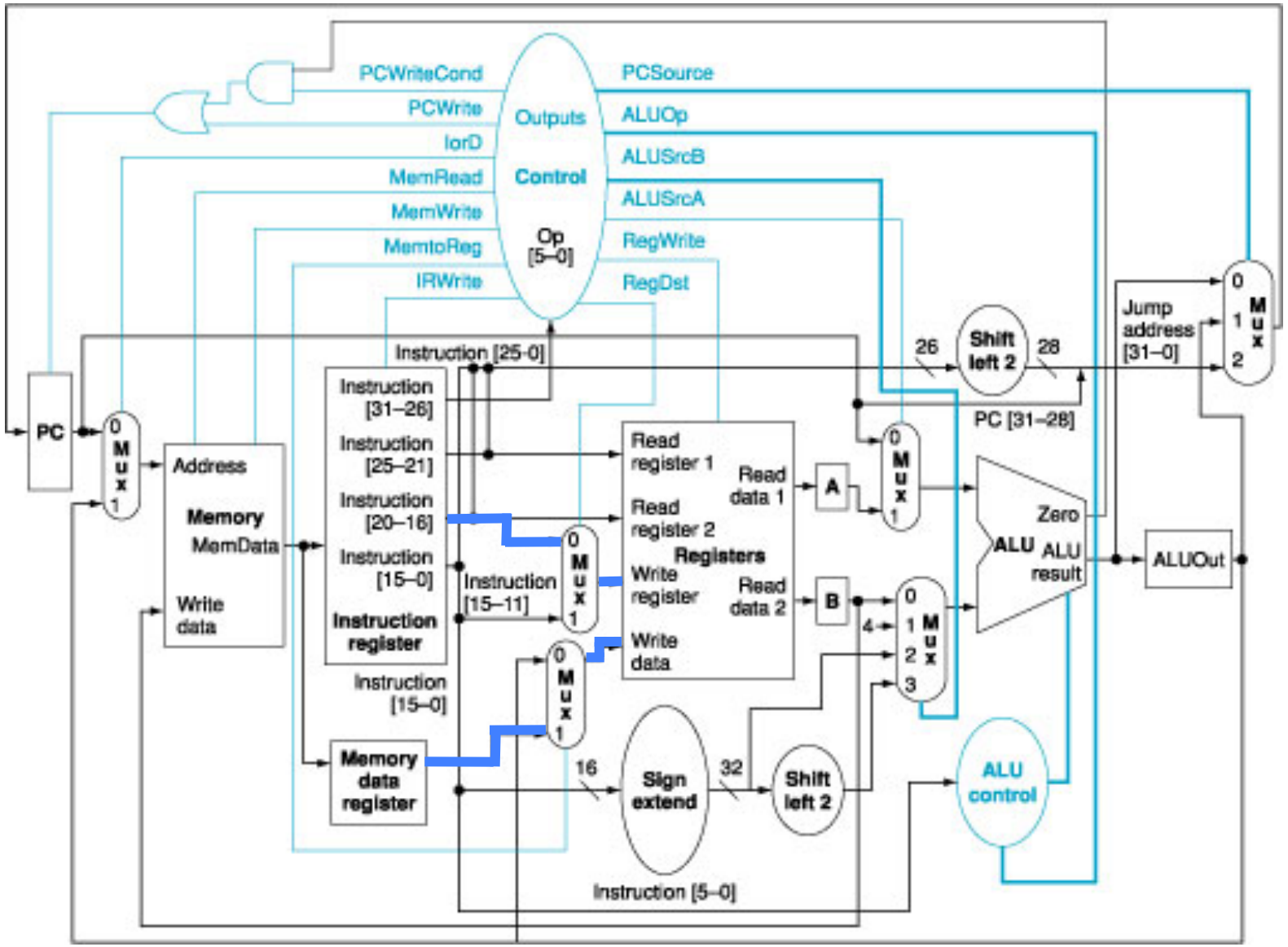


What happens in each cycle – 5

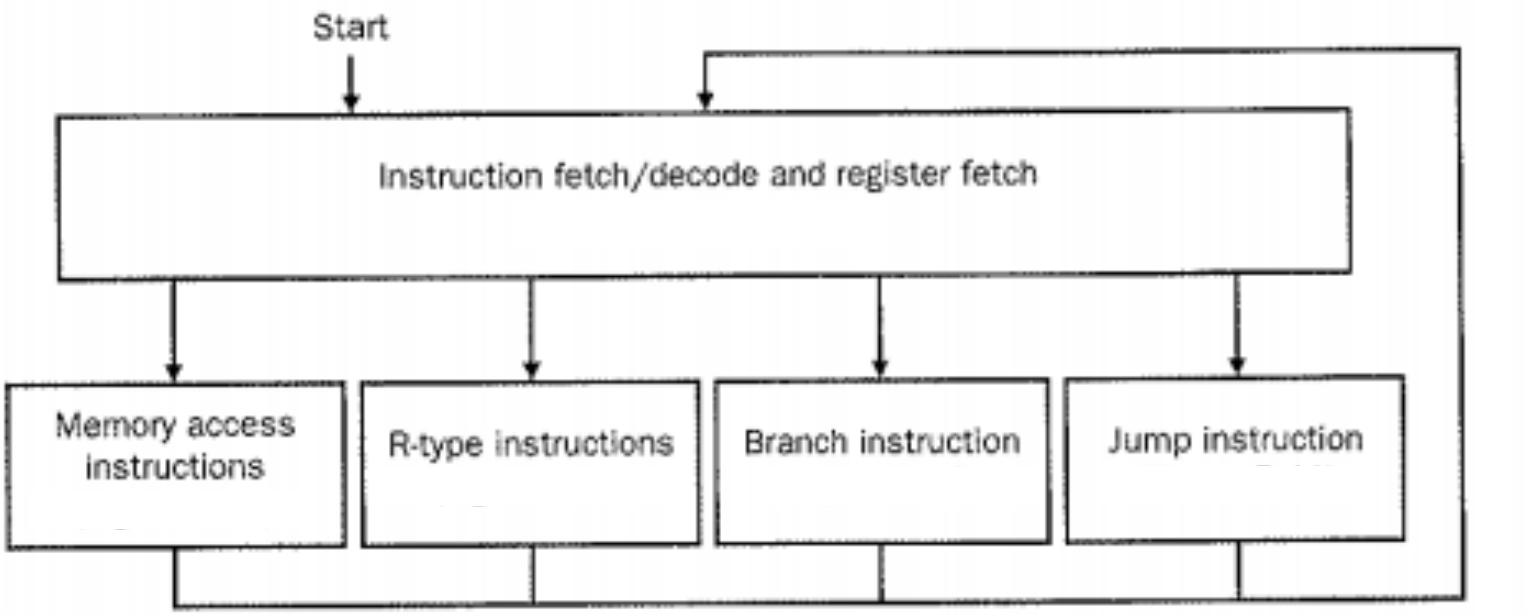
5. Load instruction completion

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

Cycle 5: save of loaded value (lw)

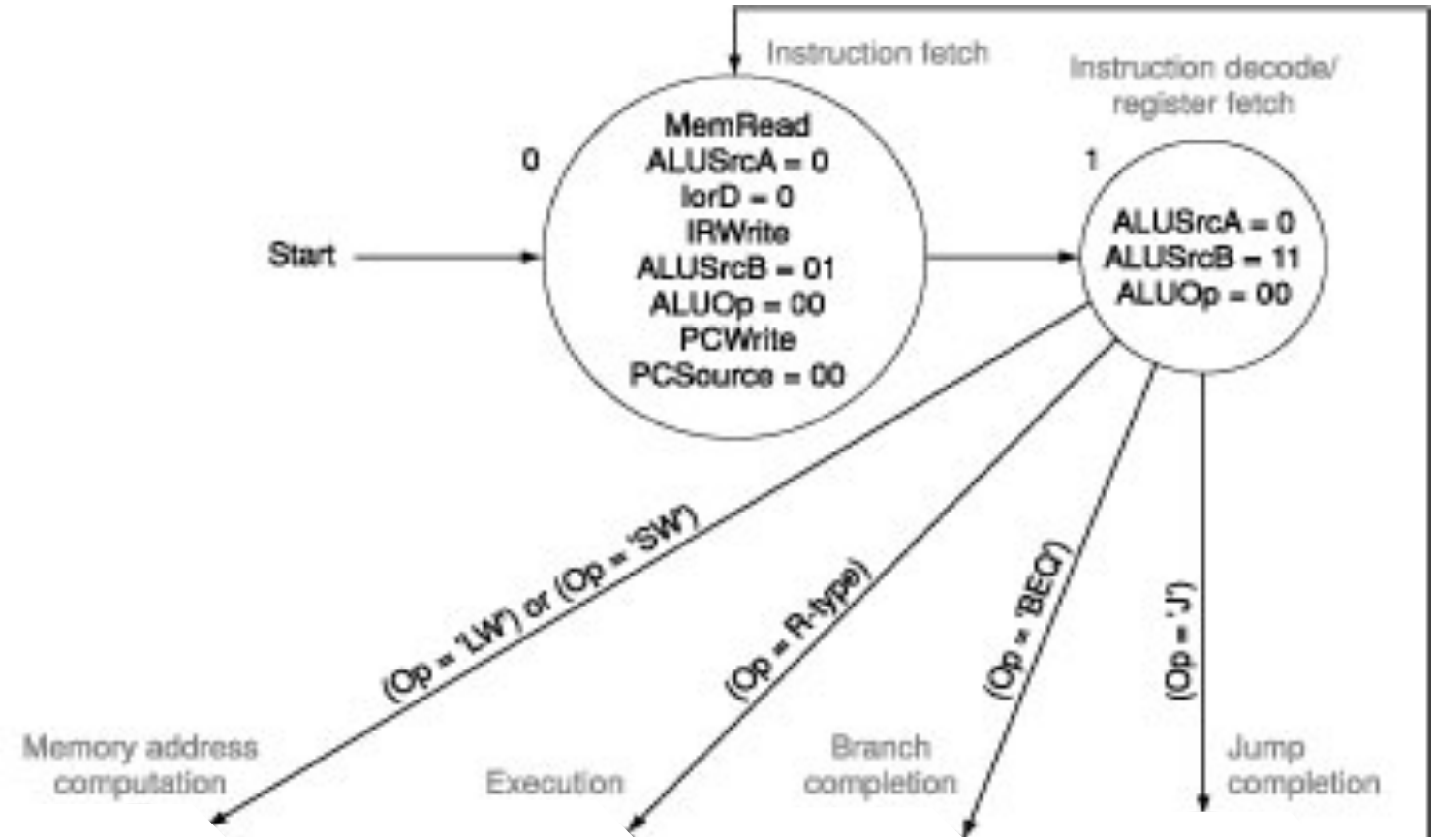


Designing the Control Unit

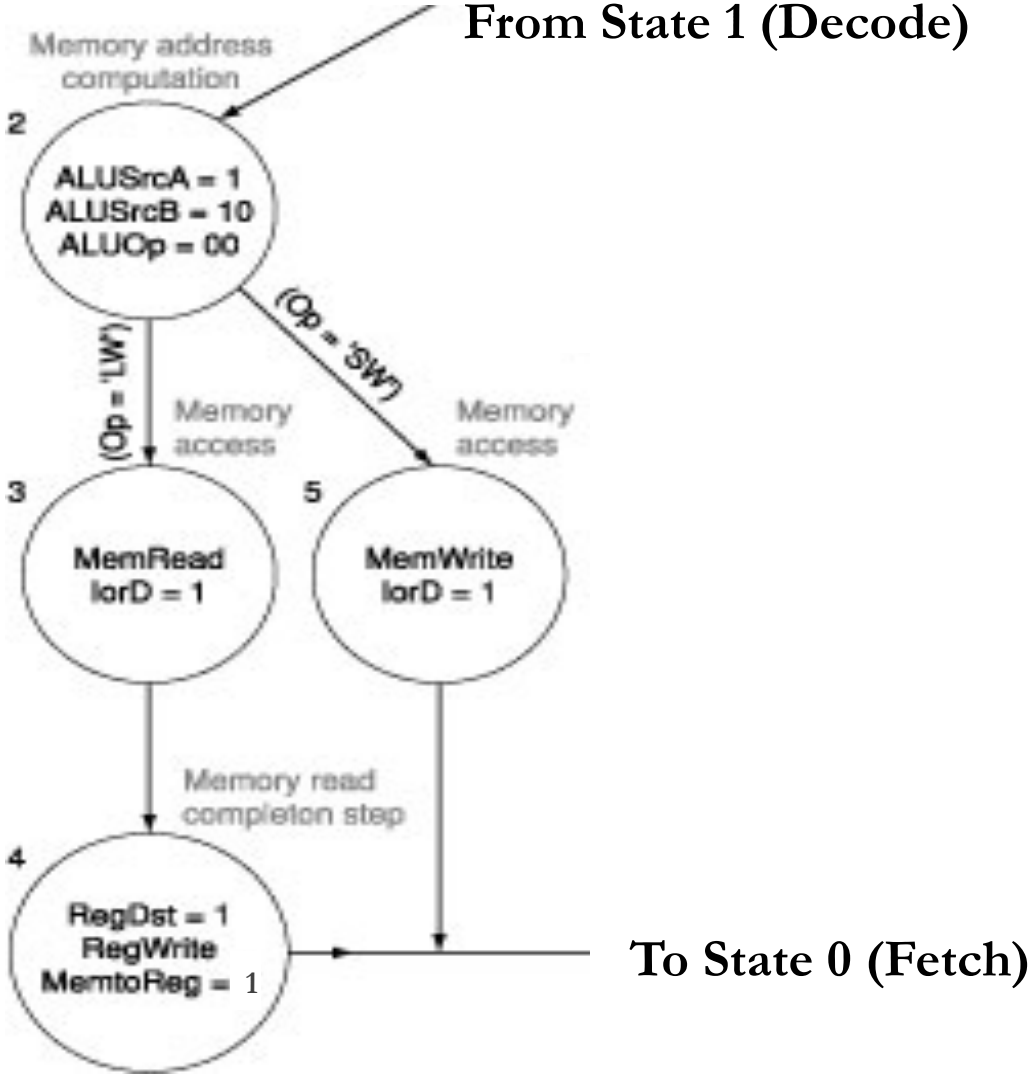


- Fetch & Decode common for all insts
  - 2 cycles total (Fetch, Decode)
- Rest: depends on the inst type
  - 1 to 3 additional cycles

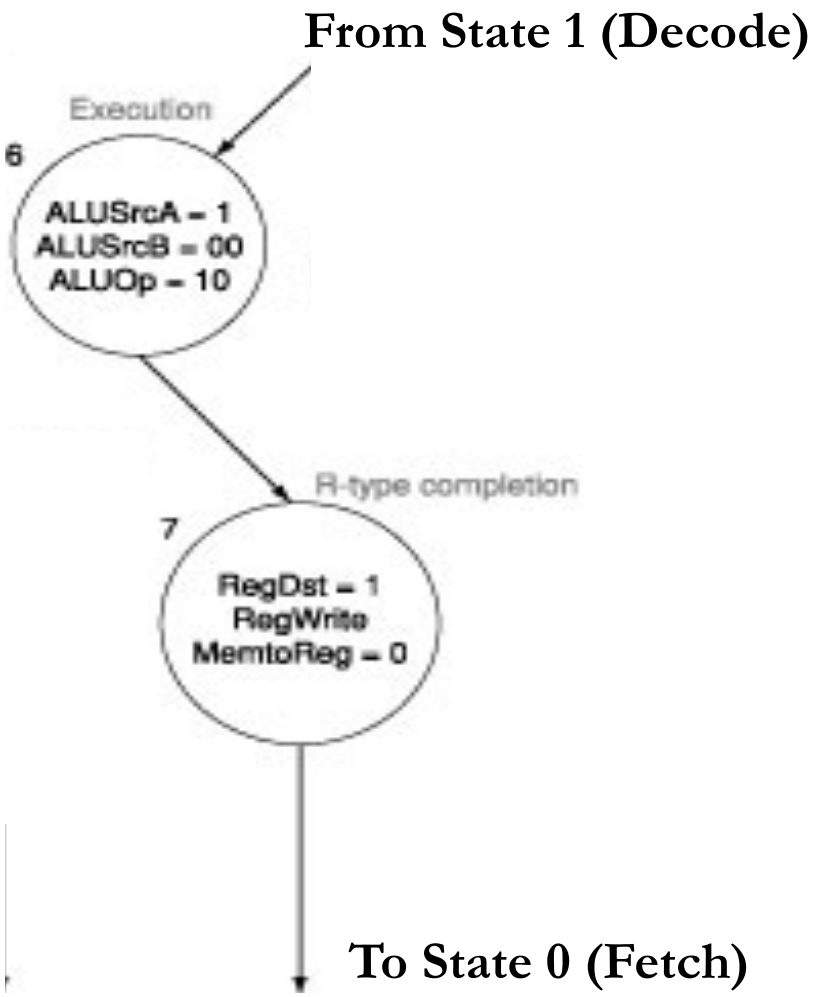
Fetch and Decode States



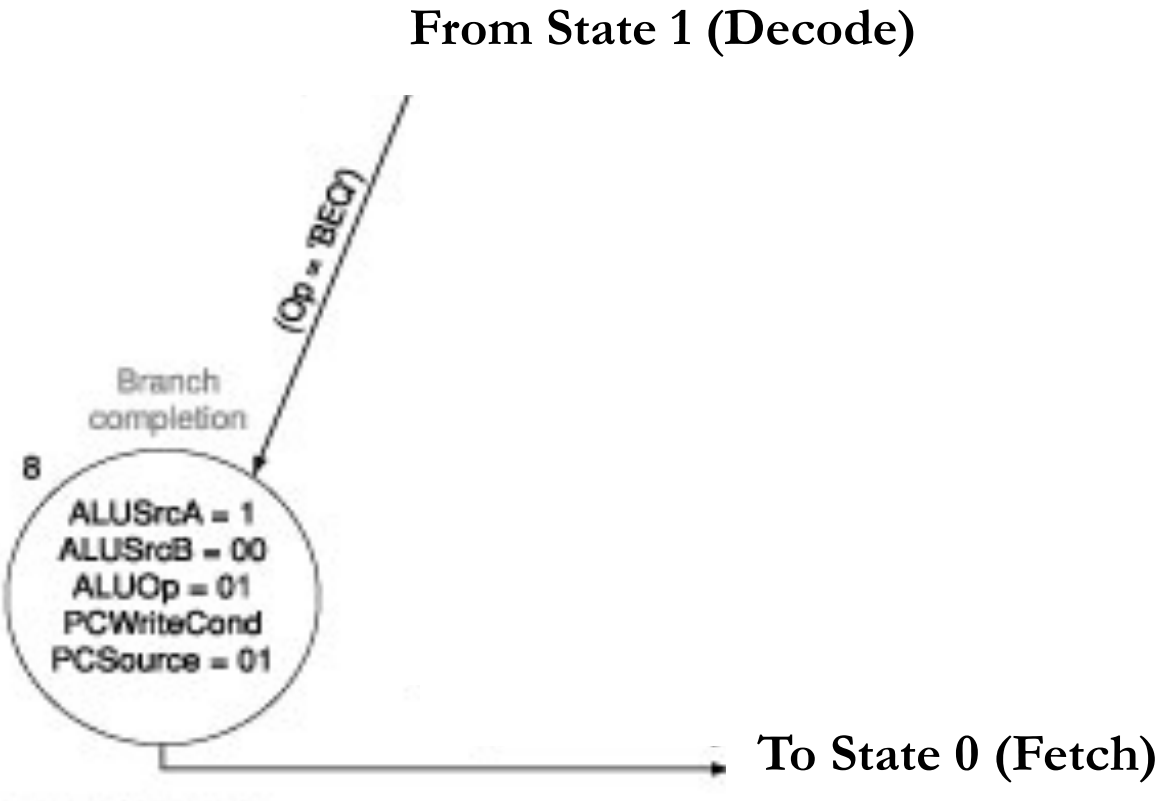
Memory Access States



R-Type Instruction States

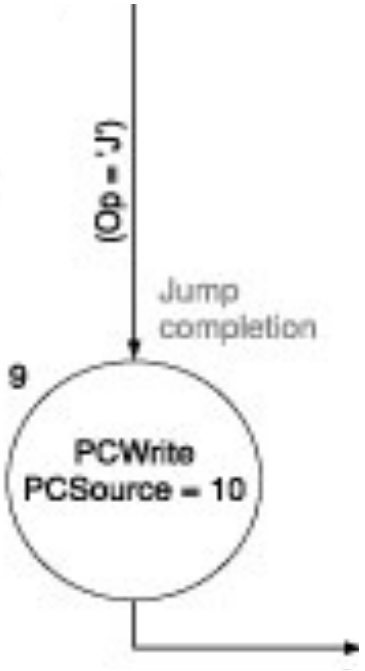


Branch Resolution States

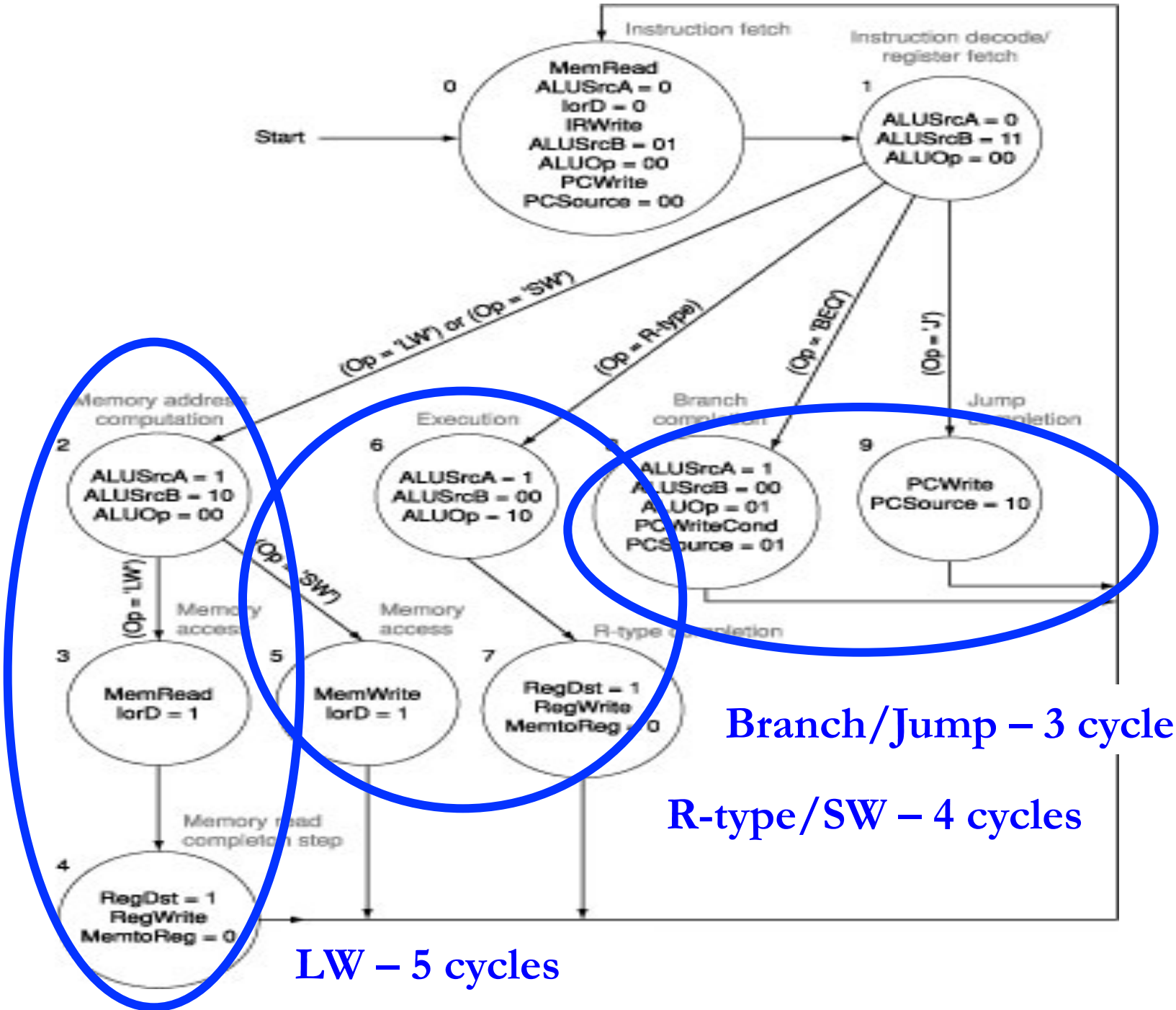


Jump Resolution States

From State 1 (Decode)



To State 0 (Fetch)



Branch/Jump – 3 cycles

R-type/SW – 4 cycles

LW – 5 cycles