TEAM 18

Hardware Project

Multi-Cycle Mips

Parts

Mohamed Ghareeb	Mohamed Kadry	Mahmoud Khaled	Mohamed Mostafa	Mohamed nabil
Memory	ALU	Countrol unit	Connection	Test Benches
Register file	ShiftLeft	ALU control		Simulation
MDR	Sign Extend			
PC	MUXs			

Microprocessors

Microprocessors are essential in today's **technological landscape**, powering everything from smartphones to supercomputers. Understanding their design and functionality is crucial for innovation.

DEFINITION

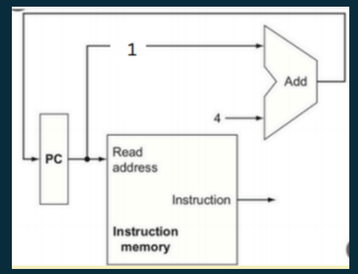
- Integrated circuit technology
- Executes instructions
- Performs calculations

IMPORTANCE

- Drives modern technology
- Enables automation
- Supports diverse applications

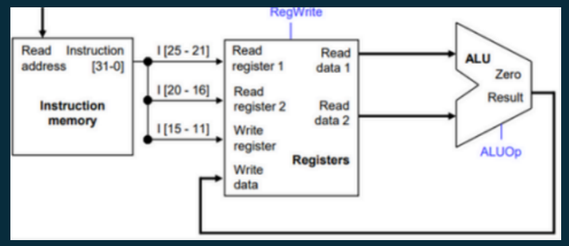
Project States

Instruction Fetch



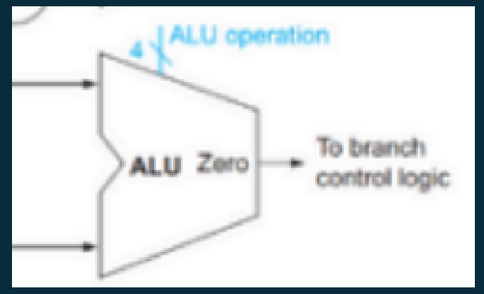
Fetch instruction, increment PC.

Instruction Decode / Register Fetch



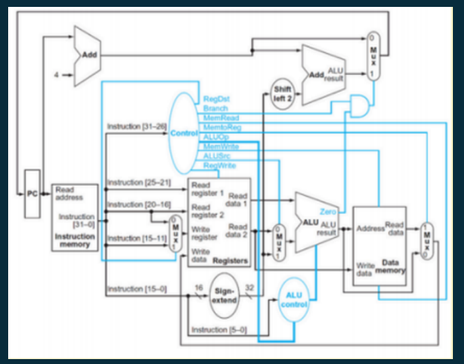
Decode, read registers, sign-extend.

Execute / Address Calculation



ALU ops, address calc, branch decision.

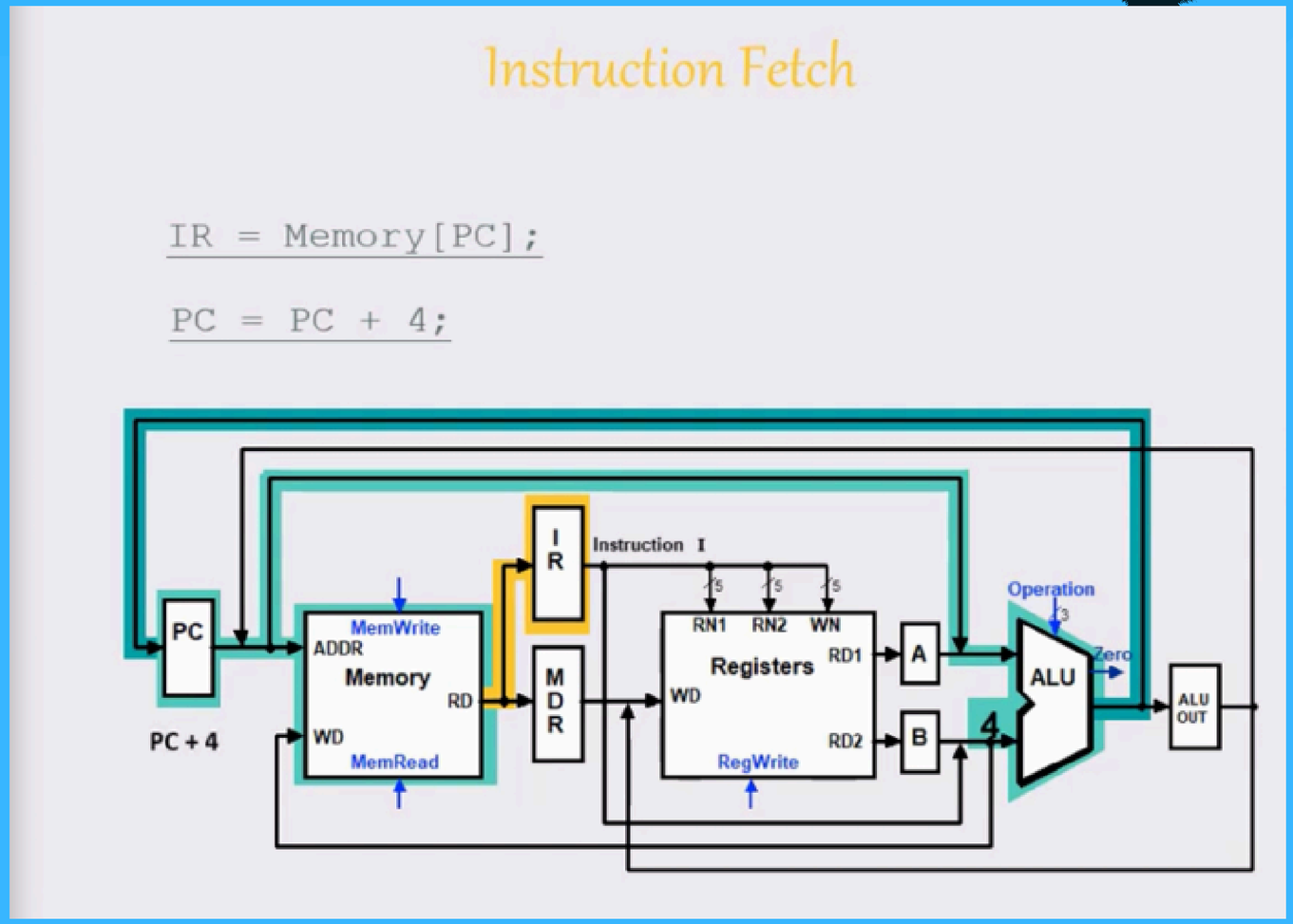
Memory Access



Read/write data memory.

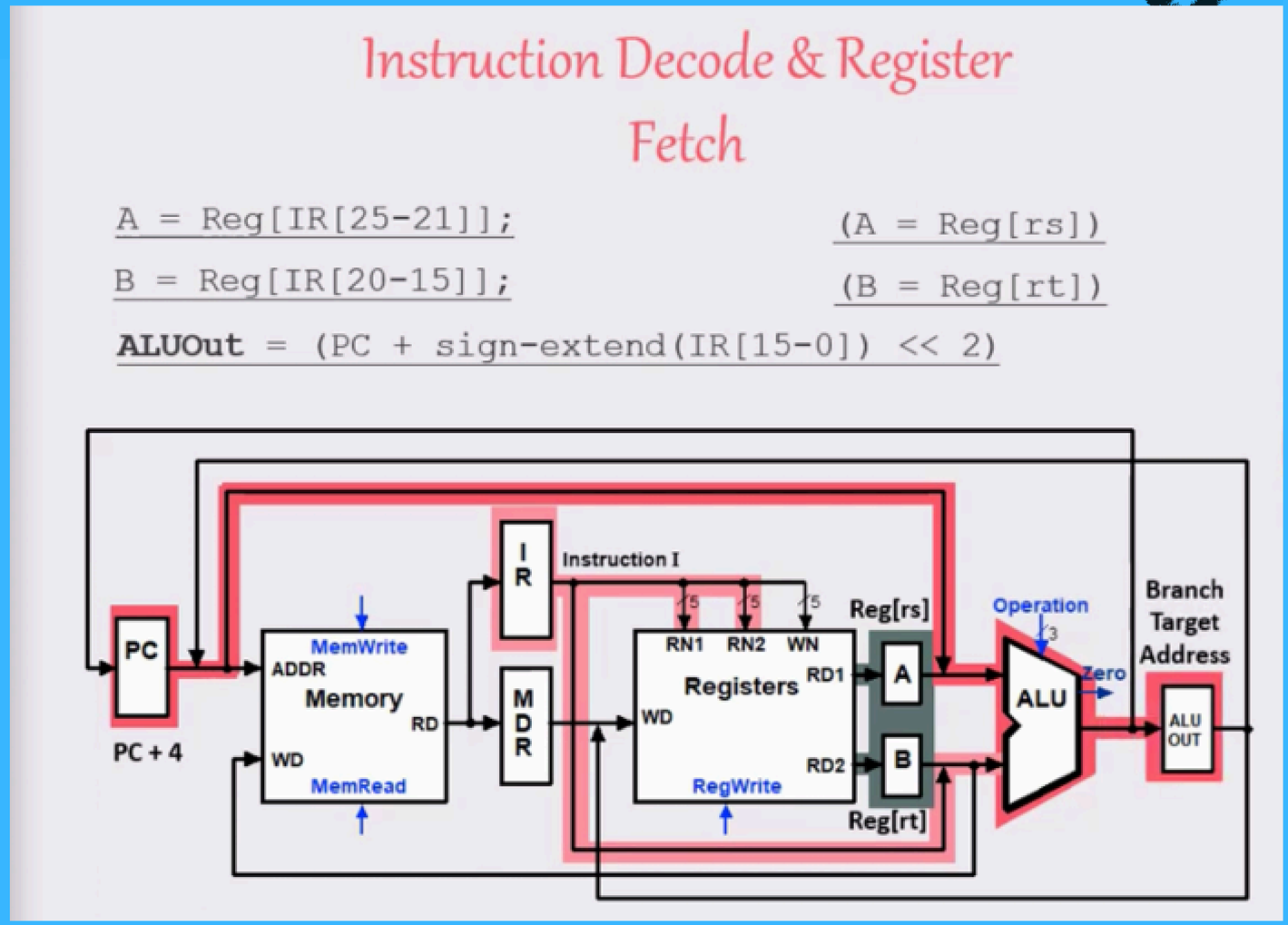
Instruction Fetch

- Fetch the instruction from memory using the Program Counter (PC).
- Increment the PC by 4 byte (assuming 32-bit instructions).



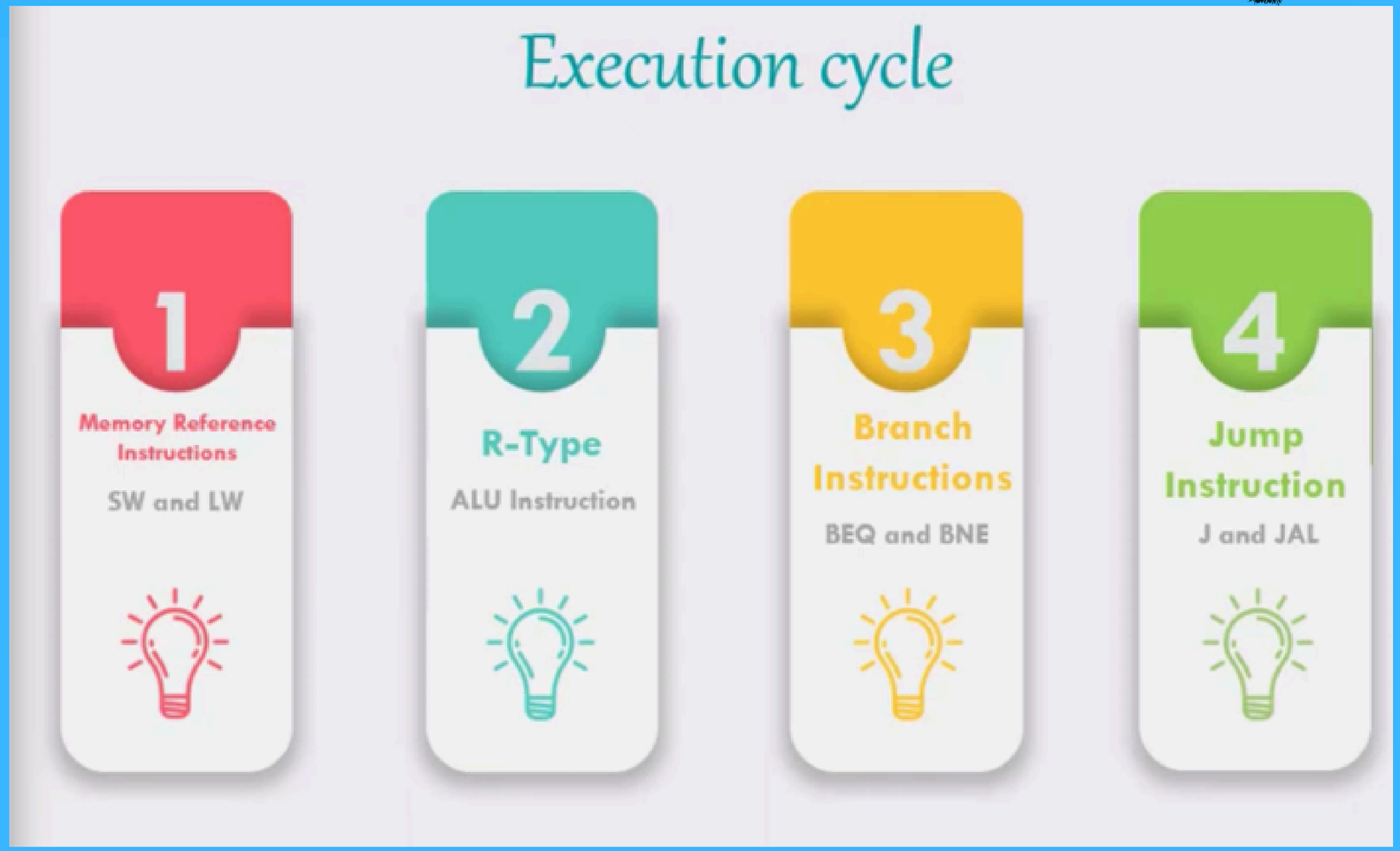
Instruction Decode

- Decode the instruction (determine opcode, rs, rt, rd, etc.).
- Read operands from the register file (registers rs and rt).
- Sign-extend the immediate field (for I-type instructions).

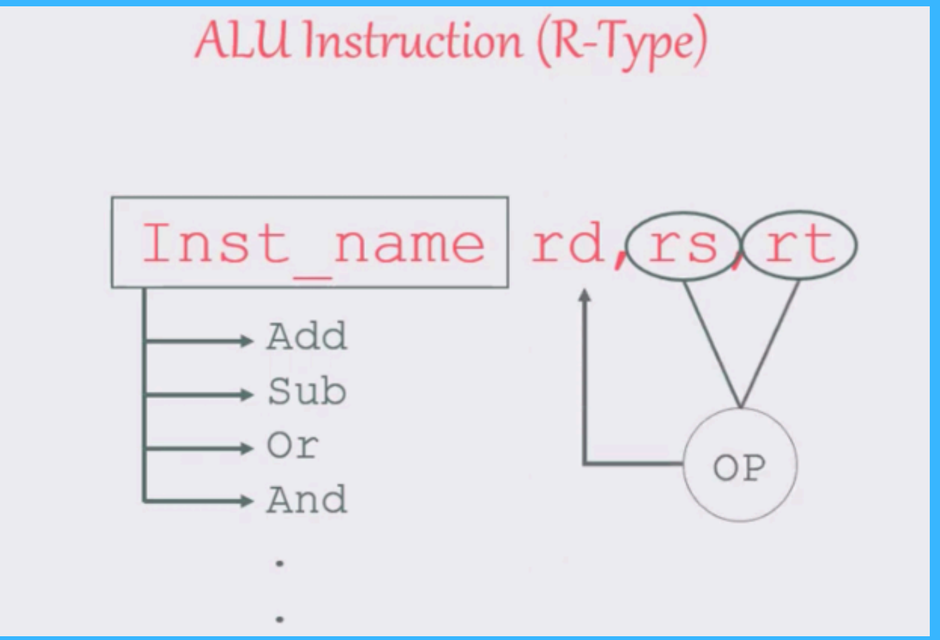


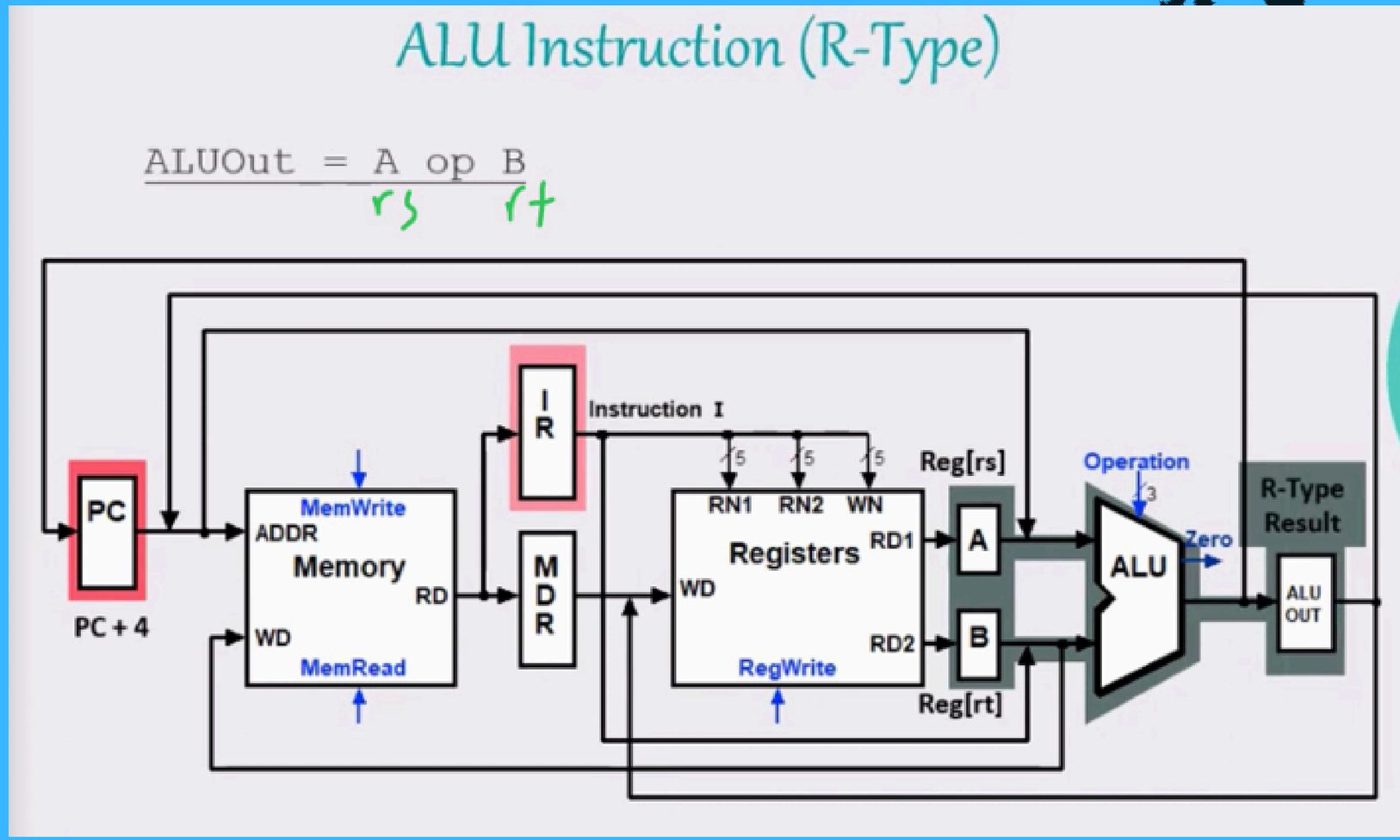
Execute

- Address calculation (for load/store).
- Arithmetic/logic (for R-type).
- Branch Instructions.
- Jump Instructions.

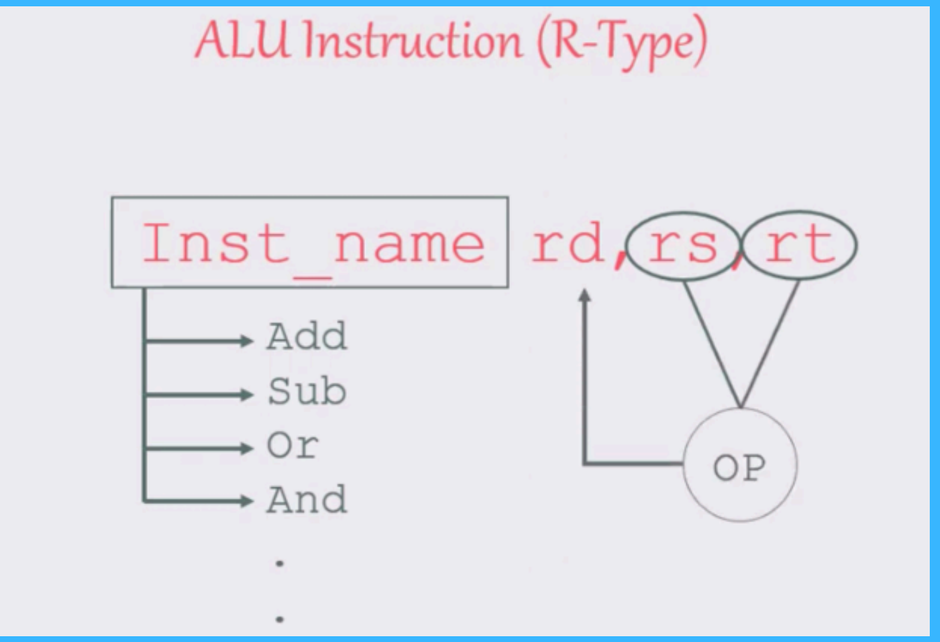


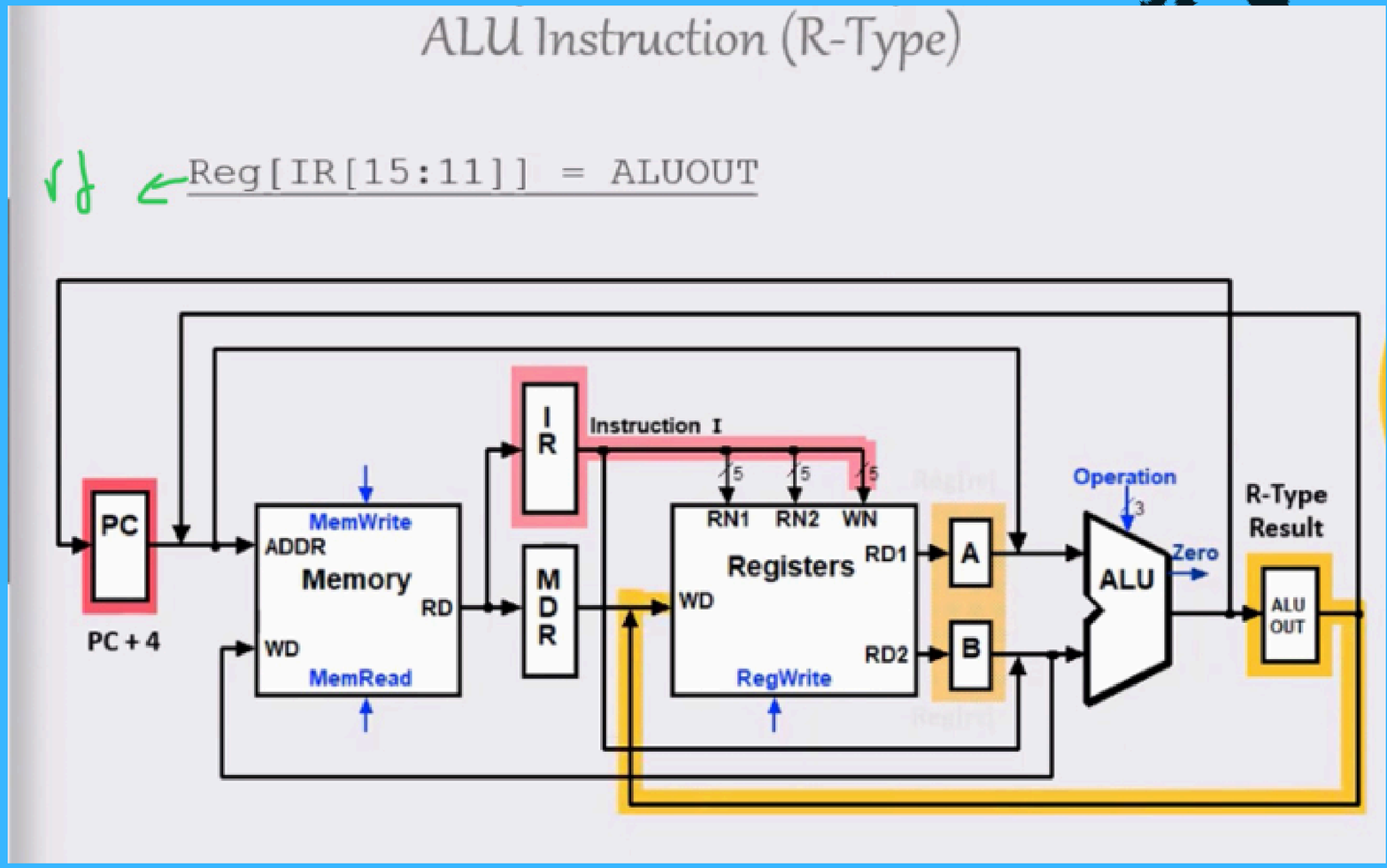
Execute (R_type)



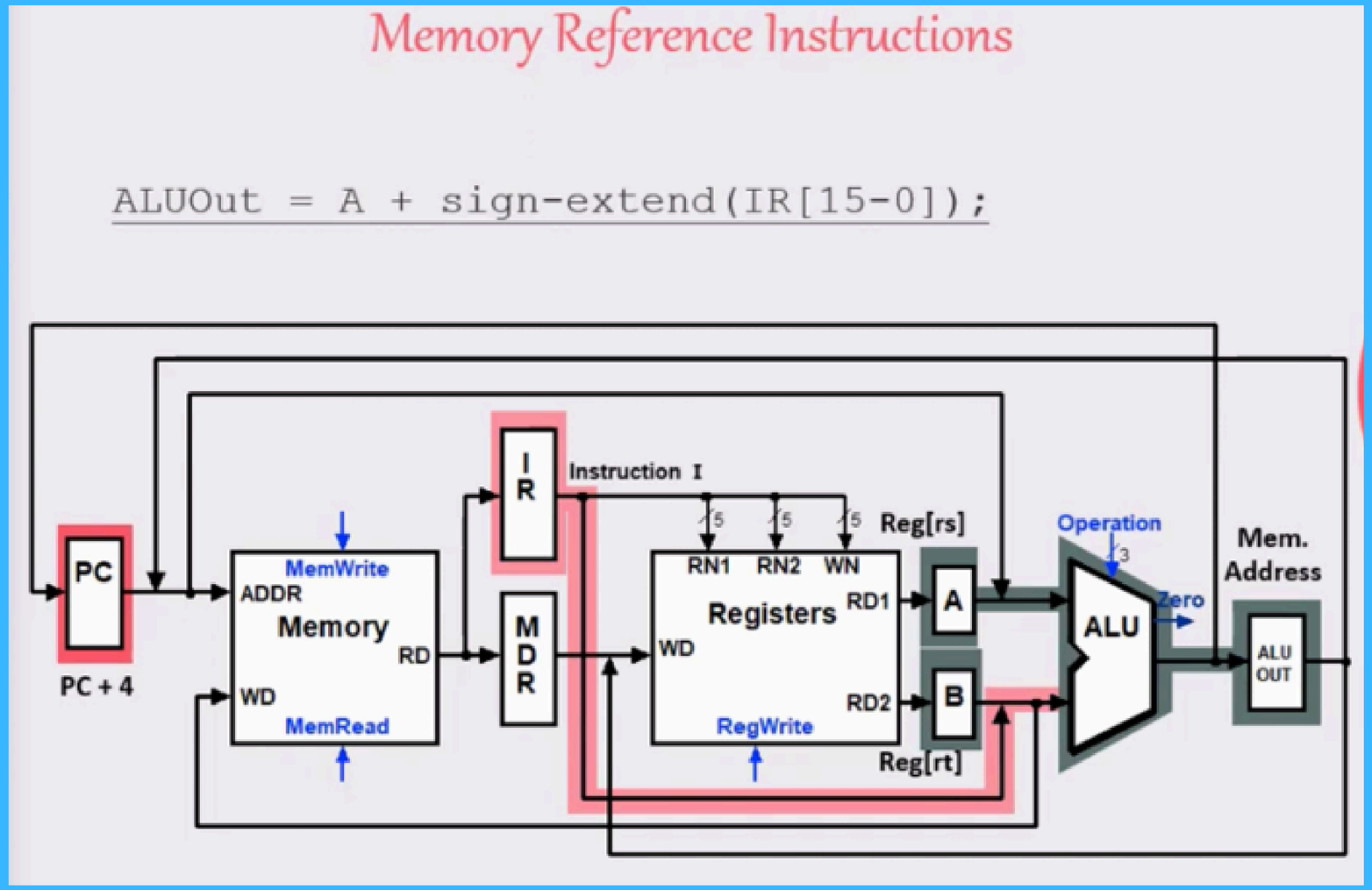


Execute (R_type)



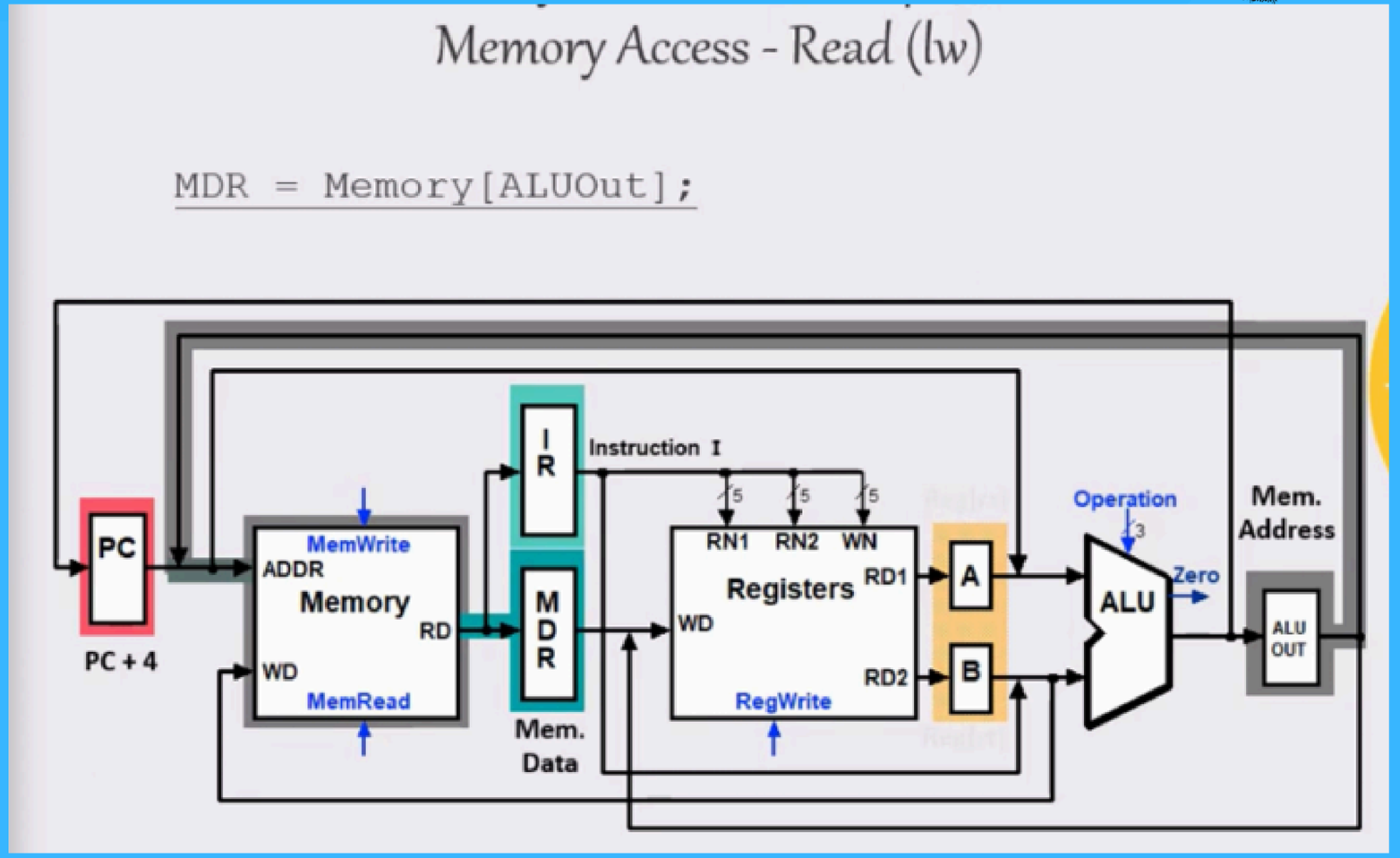


Execute (I_type)

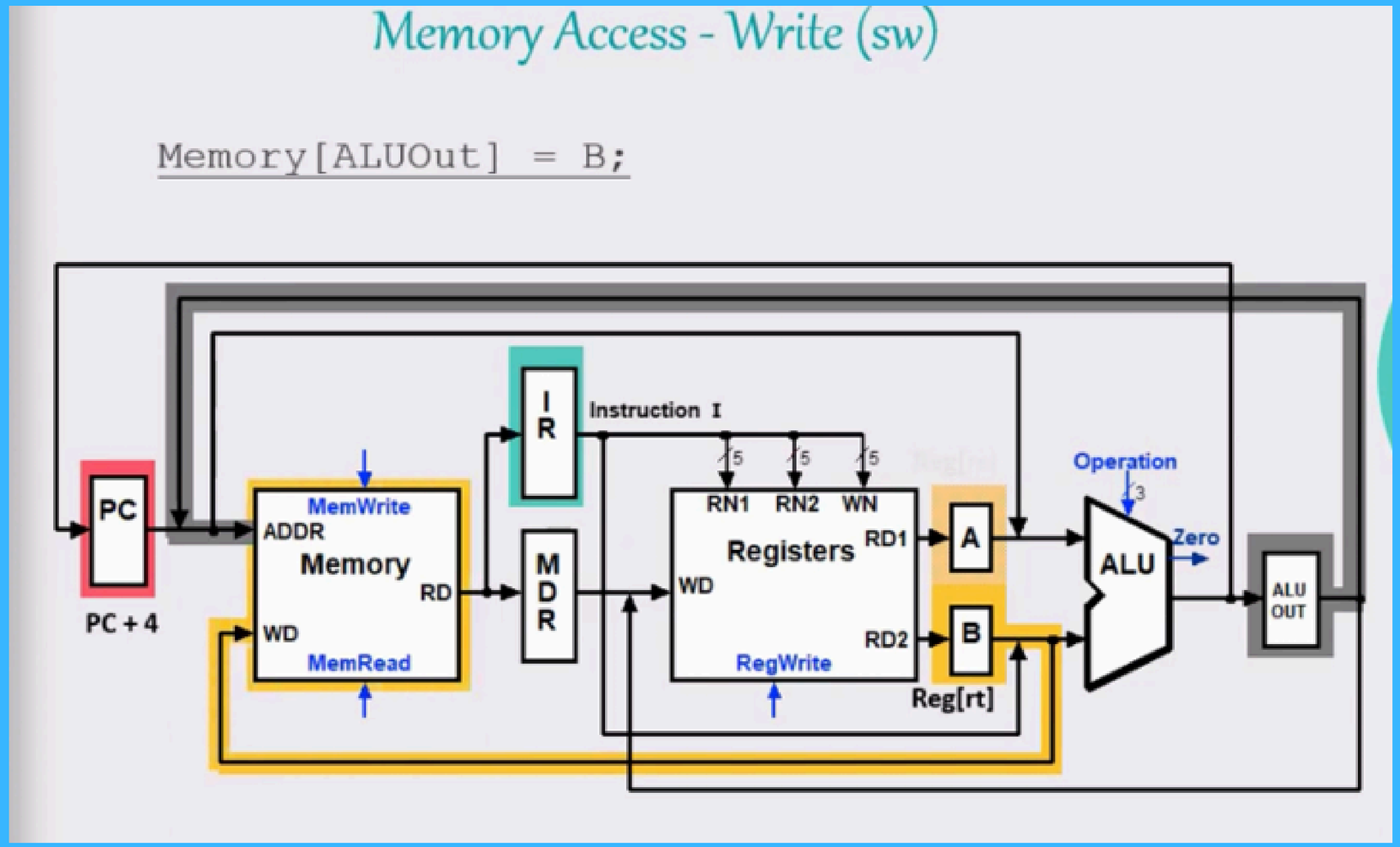


Memory Access (I_type)

- Access memory if it's a load (lw) or store (sw).
- Load: Read data from memory.
- Store: Write data to memory.
- Other instructions pass through this stage without using memory.



Memory Access (I_type)

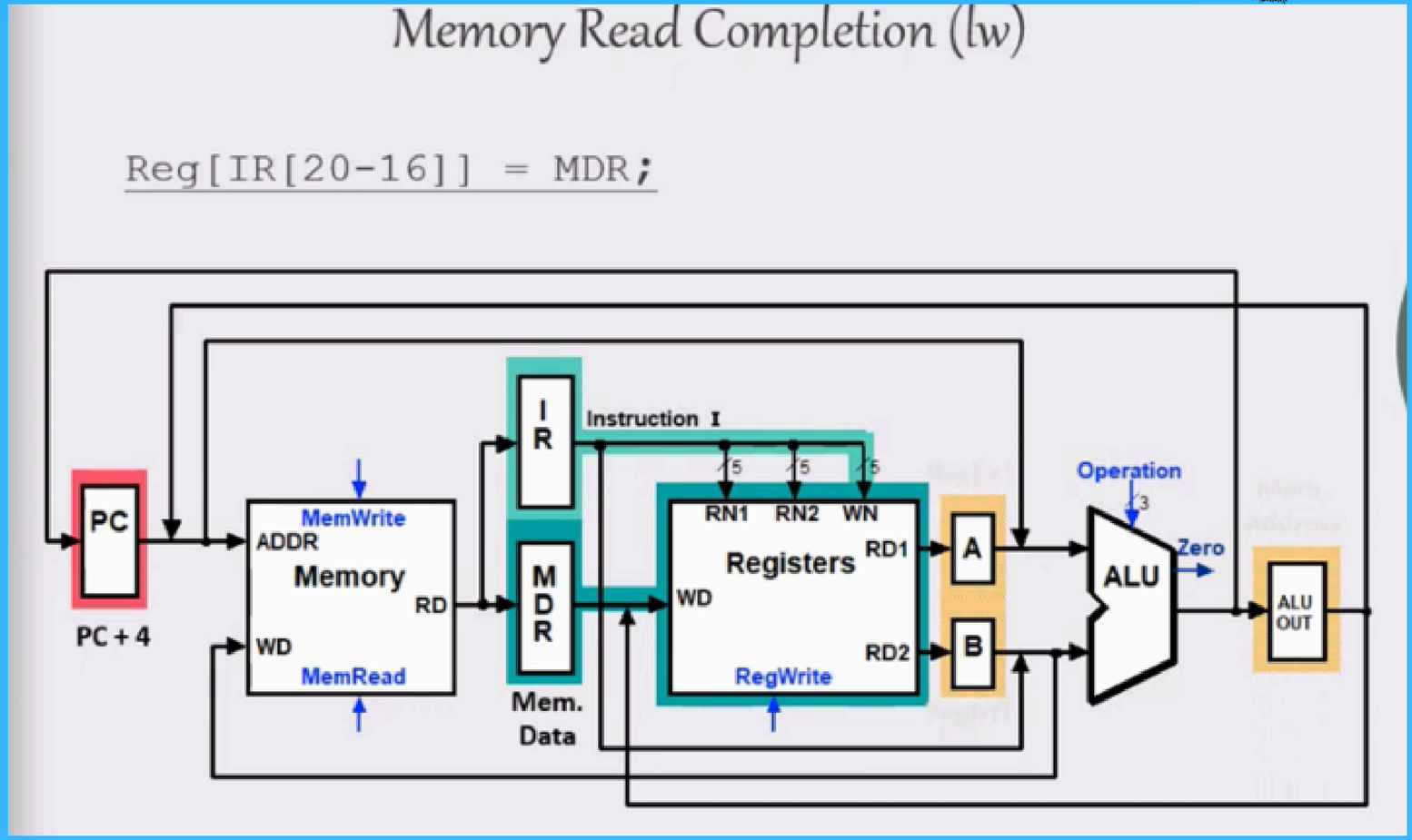


Write Back to register

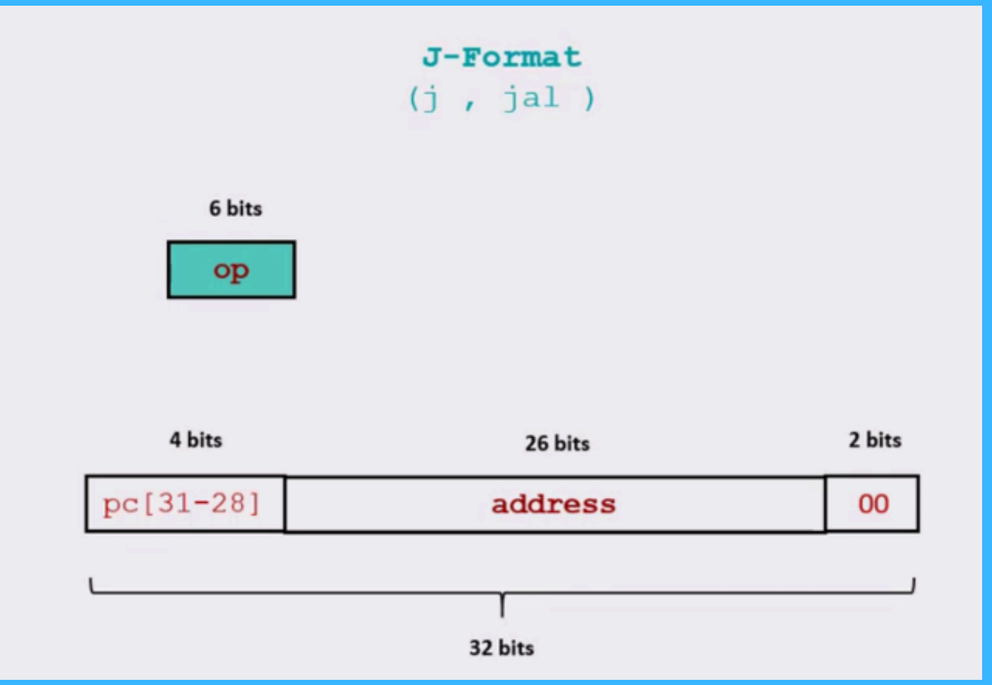
Write the result back to the register file.

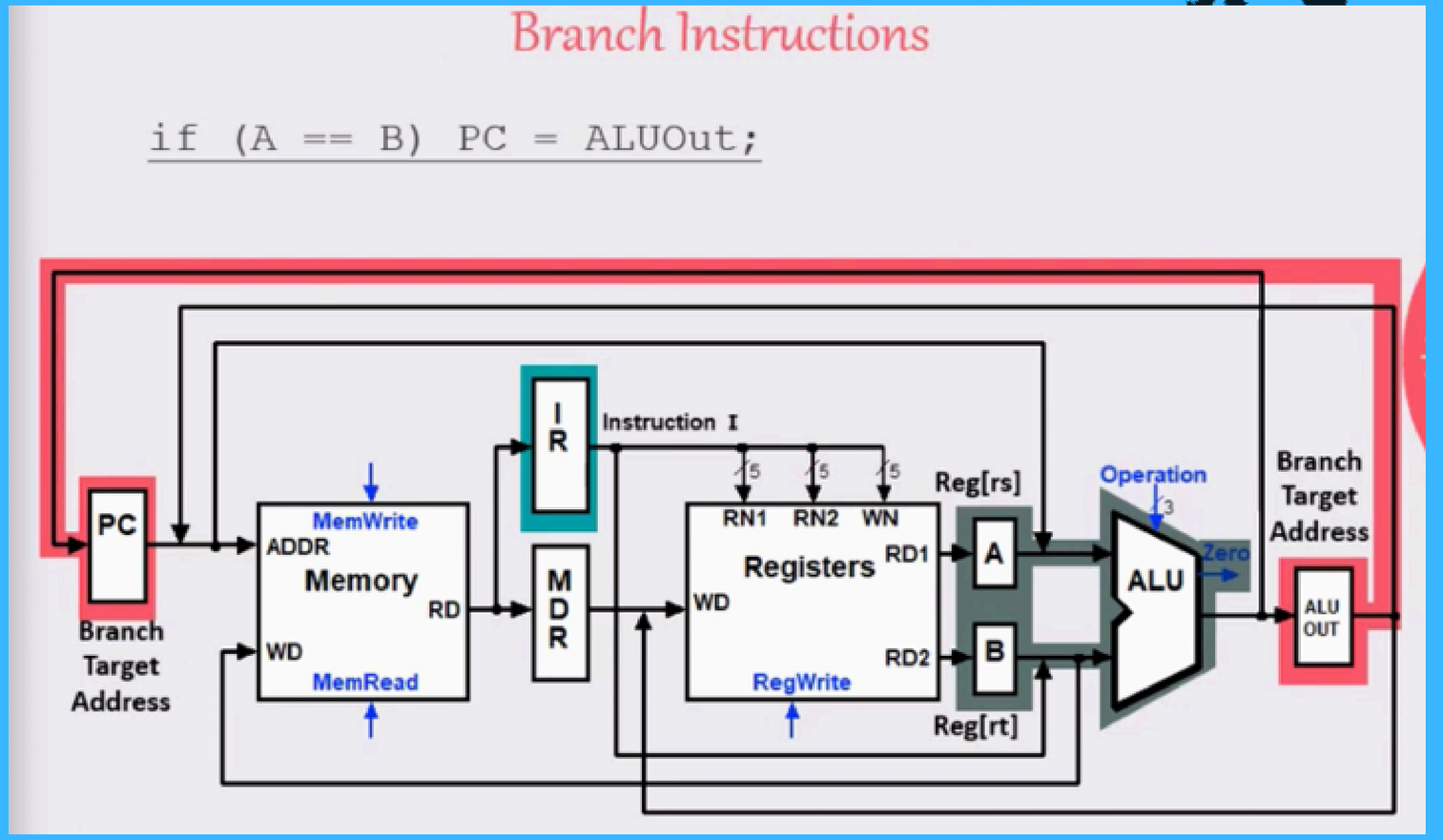
- R-type: write ALU result to rd.
- Load: write memory data to rt.

Other instructions pass through this stage without using memory.

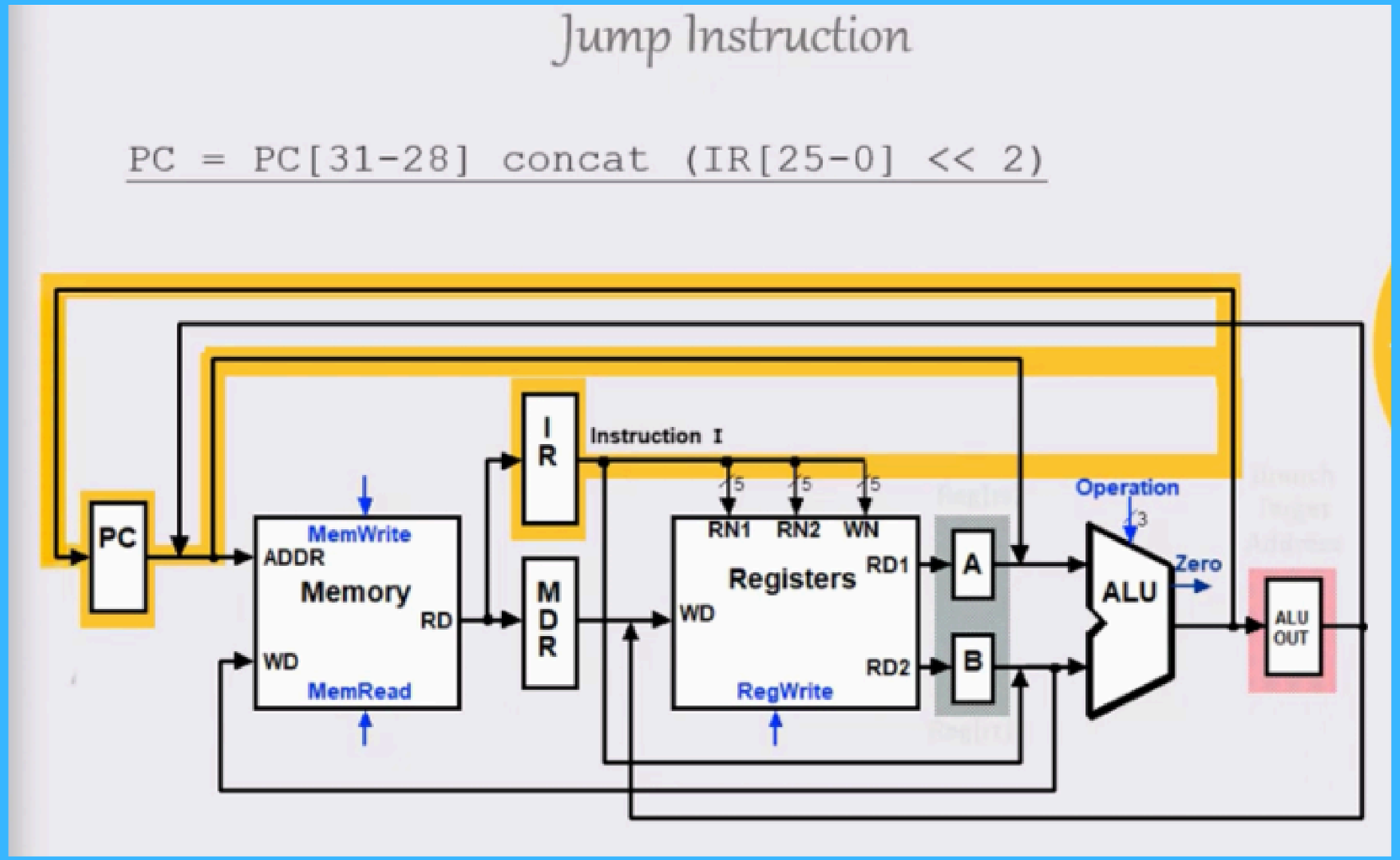


Execute (Branch) (J_type)



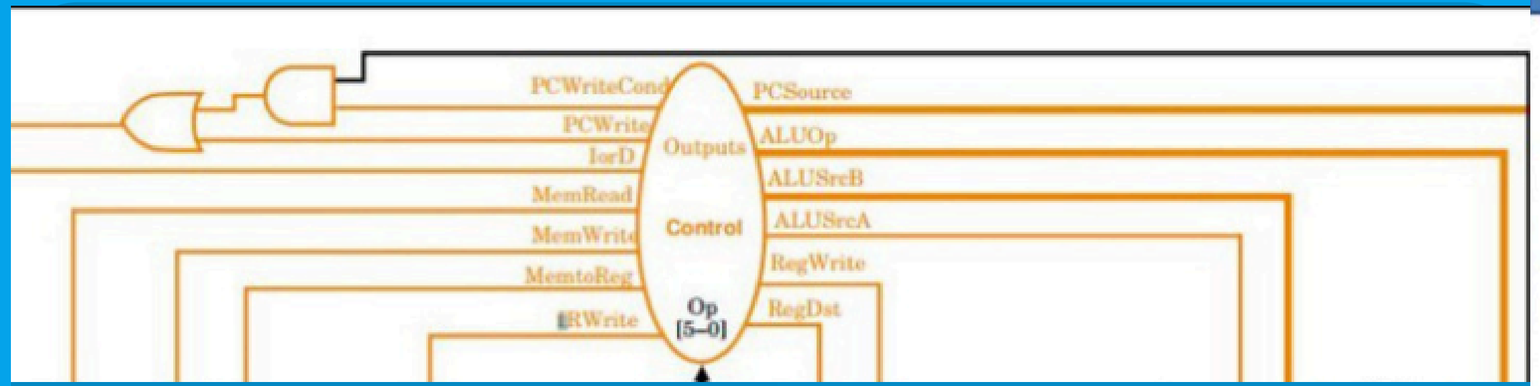


JUMP (J_type)


Jump Instruction
PC = PC[31-28] concat (IR[25-0] << 2)
PC
MemWrite
ADDR
Memory
RD
WD
MemRead
IR
Instruction I
MDR
25
25
25
RN1
RN2
WN
Registers
RD1
RD2
WD
RegWrite
A
B
Operation
3
ALU
Zero
ALU OUT

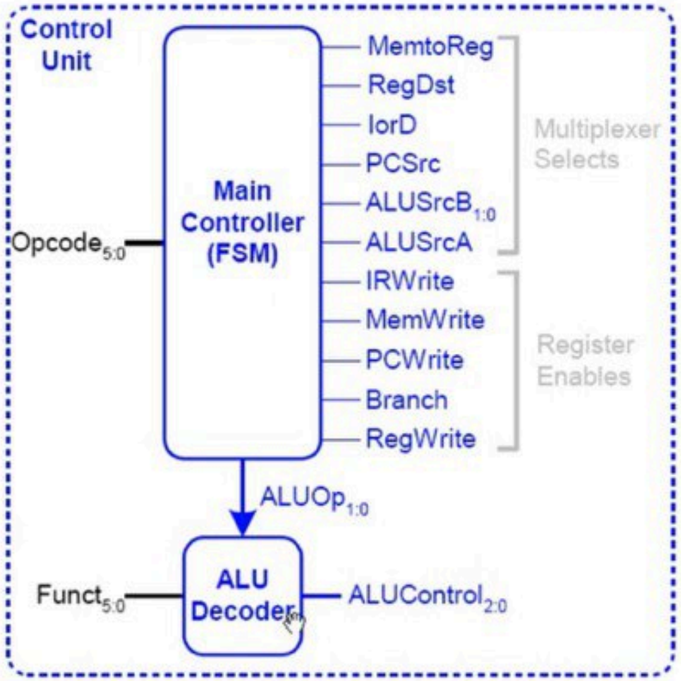
Control Unit

It is responsible for generation a sequence if control signals over multiples clock cycles to coordinate the operation of the datapath components



- **Inputs**
  - Operation (op)
  - Function (Funct)
- **Outputs**
  - Multiplexer Select Signals
  - Register Enable Signals
  - ALU Control Signal

Multicycle Control



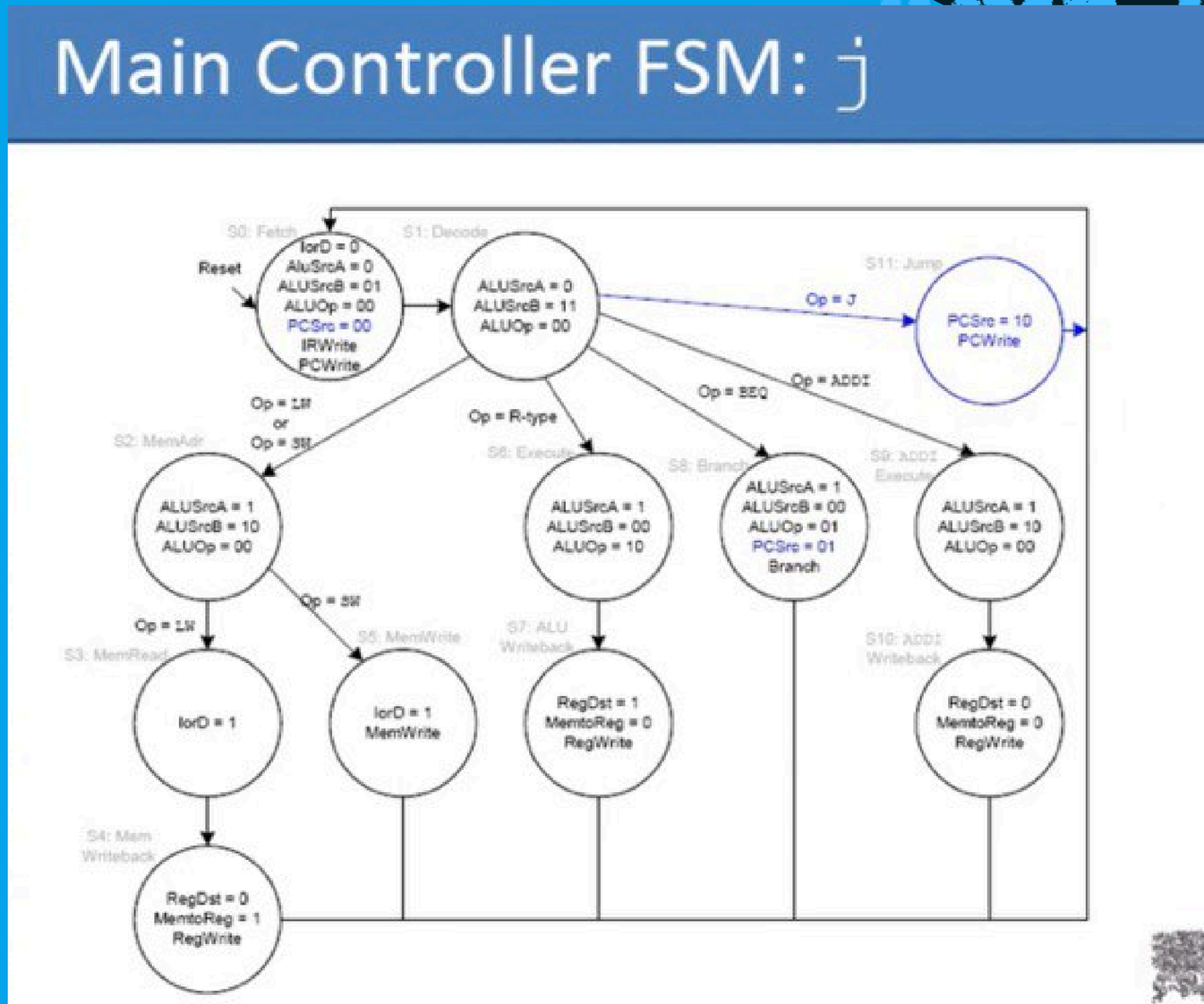
Control Unit & FSM

FSM is typically used to control the execution of instructions in a multi-cycle implementation of the processor.

- **Main States**
  - IF: Instruction Fetch
  - ID: Instruction Decode / Register Fetch
  - EX: Execute / Address Calculation
  - MEM: Memory Access
  - WB: Write Back data to register

FSM States

- **States**
  - S0 --> Fetch state
  - S1 --> Decode state
  - S2 --> MemAdr state
  - S3 --> MemRead state
  - S4 --> MemWriteBack state
  - S5 --> MemWrite state
  - S6 --> Execute state
  - S7 --> ALUWriteback state
  - S8 --> Branch state
  - S9 --> ADDIExute state
  - S10 --> ADDIWriteback state
  - S11 --> JUMP state



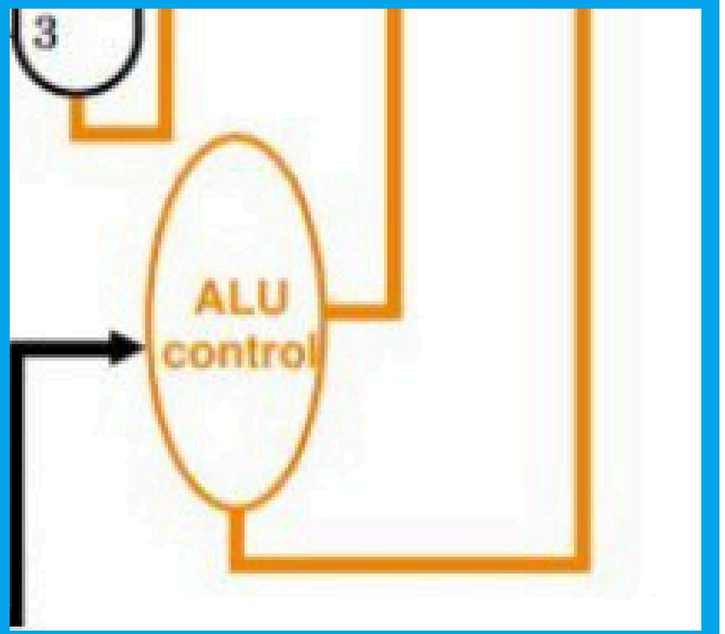
ALU Control Unit

$a_n = a_1 + (n-1)d$

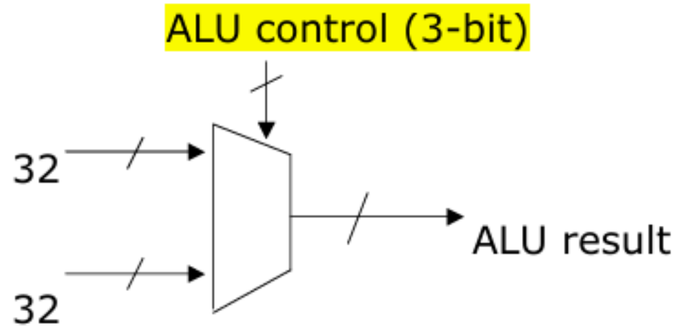
- It determines the specific operation that the ALU should perform
- based on the instruction type.

- **Inputs**
  - ALU_Operation (OP)
  - Function (funct)
- **Outputs**
  - ALU_control

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used



ALU control



ALU control input	ALU function
000	AND
001	OR
010	add
110	sub
111	Set less than

🧮 ARITHMETIC LOGIC UNIT (ALU)

📌 Function:

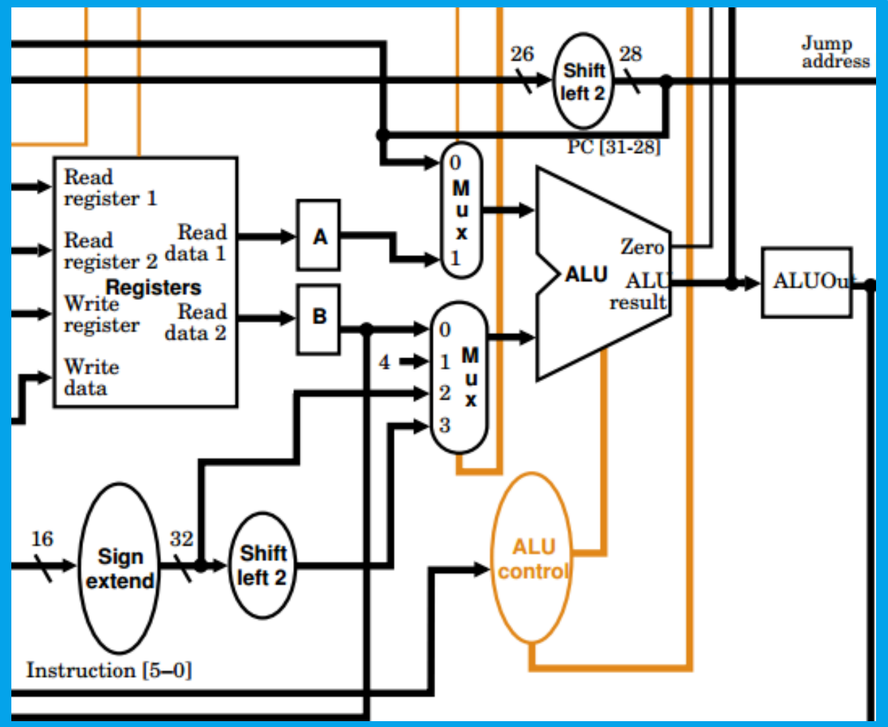
Performs arithmetic and logic operations such as add, subtract, AND, OR,NOR,XOR, etc. based on ALUControl.

🧠 Inputs:

- input1: 32-bit (from Register A or PC)
- input2: 32-bit (from Register B or sign-extended value)
- ALUControl: 4-bit signal from ALU Control

📤 Outputs:

- ALUResult: 32-bit result of operation
- Zero: 1-bit flag, used in branching



🧮 ARITHMETIC LOGIC UNIT (ALU)

🔗 Related Components:

Connected To	Purpose
MUX before ALU	Selects input1 (A or PC) and input2
ALU Control	Tells ALU which operation to perform
ALUOut Register	Stores result for next cycle use
Zero Flag to CU	Used for BEQ decision in control

🔀 MULTIPLEXERS (MUXes)

📌 Function:

Control the data path by selecting between two or more inputs based on control signals.

🧠 Key MUXes in Multi-Cycle MIPS:

MUX Name	Selects Between	Selects Between	Feeds Into
ALUSrcA	PC vs Register A	ALUSrcA signal	ALU input1
ALUSrcB	Register B / Constant / SE value	ALUSrcB (2-bit)	ALU input2
RegDst	rt vs rd	RegDst	Register File write addr
MemToReg	ALU result vs Memory data	MemToReg	Register File write data
PCSource	ALUResult / ALUOut / jump addr	PCSource (2-bit)	PC next value

➕ SIGN EXTEND UNIT

📌 **Function:**

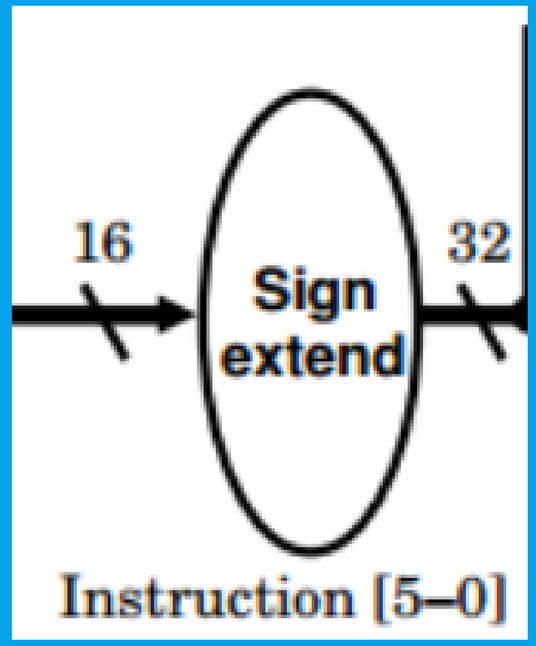
Extends a 16-bit immediate value (from instruction) to 32 bits for arithmetic or address computation.

🧠 **Input:**

- IR[15:0]: immediate field of I-type instruction

📤 **Output:**

- SignImm: 32-bit sign-extended version



+ SIGN EXTEND UNIT

🔗 Related Components:

Connects To	Purpose
ALUSrcB MUX	One of the options for input2
Shift Left 2	For branch address calculation

⏩ SHIFT LEFT UNITS

📌 Two Types of Shift-Left Used:

A. Shift Left 2 (Branch)

Function:

Used to multiply the sign-extended offset by 4 (i.e., shift left 2 bits) for byte address in branch calculation.

Input:

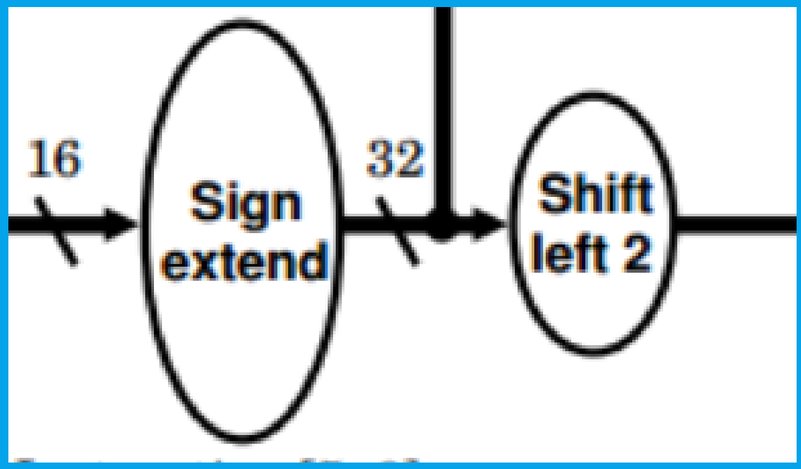
- SignImm (32-bit)

Output:

- SignImmShifted: Shifted left by 2 bits (word to byte address)

Used in:

- PC + 4 + (SignImm << 2) for BEQ



⏩ SHIFT LEFT UNITS

📌 **Two Types of Shift-Left Used:**

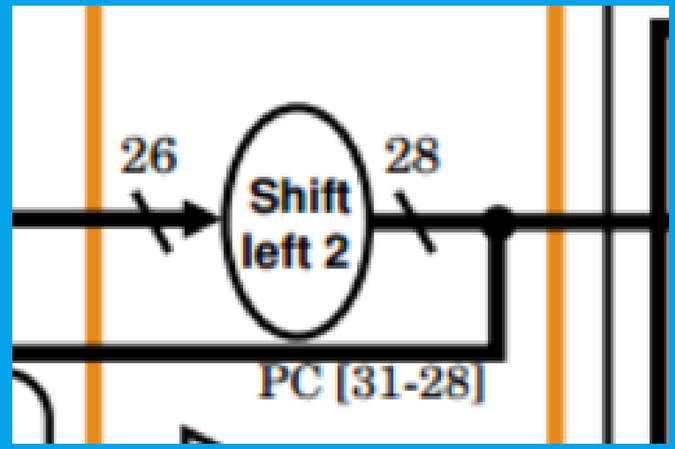
B. Shift Left 2 (Jump)

Function:

Used to build the jump target address.

Input:

- **IR[25:0]: jump target (26 bits)**



Output:

- **JumpAddress: Shifted and concatenated to form PC[31:28] & JumpAddr**

⏩ SHIFT LEFT UNITS

🔗 Related Components:

Component	Used For
ALU	Needs shifted offset for BEQ
PC Update MUX	Uses shift result for jump/branch

🧠Memories & Registers

- PC “Program Counter”
- Memory Unit
- IR “Instruction Register”
- MDR “Memory Data Register”
- Register File

Program Counter (PC)

What it does:

- The PC holds the address of the next instruction to be fetched from memory.
- Manages program flow control

- **Key Features:**
  - Auto-increments by 4 (next instruction)
  - Supports three update modes:
    - Sequential (+4)
    - Branch/jump
      - Manual Address Modification based on Immediate Value

Memory Unit

What it does:

- Stores all program instructions and data
- Acts as the computer's "long-term storage"

- **Key Features:**
  - 1024 addressable locations (1KB)
  - 32-bit data width (MIPS standard)
  - The memory unit typically includes:
  - An address input (to select the memory location).
  - A data input/output (for reading or writing data).
  - Control signals (like MemoryRead or MemoryWrite) to control the operation type.

⚙IR “instruction Register”

What it does:

- The IR stores the current instruction that was just fetched from memory.
- This happens after the Instruction Fetch stage.

- **Analysis:**
  - OP Code => 31 → 26
  - Reg1 => 25 → 21
  - Reg2 => 20 → 16
  - Reg Des => 15 → 11
  - Shmat => 10 → 06
  - FCode => 05 → 00

⚙️MDR “Memory Data Register”

What it does:

- The MDR is a temporary register that serves as an interface between memory and the processor.
- Its role depends on the type of operation:

- **Key Features:**
  - For memory reads:
    - → Data is read from memory into the MDR, then used by the processor.
  - For memory writes:
    - → The data to be written is first stored in the MDR, and then transferred to memory.

🏛️Register File

What it does:

- Provides ultra-fast temporary storage
- Holds data currently being processed

- **Key Features:**
  - 32 registers (R0-R31)
  - During the Instruction Decode stage:
    - → We read the source operands from the registers.
  - For an ADD instruction like: ADD R1, R2, R3
    - → R1 and R2 are read as inputs
    - → The result is stored in R3 after execution

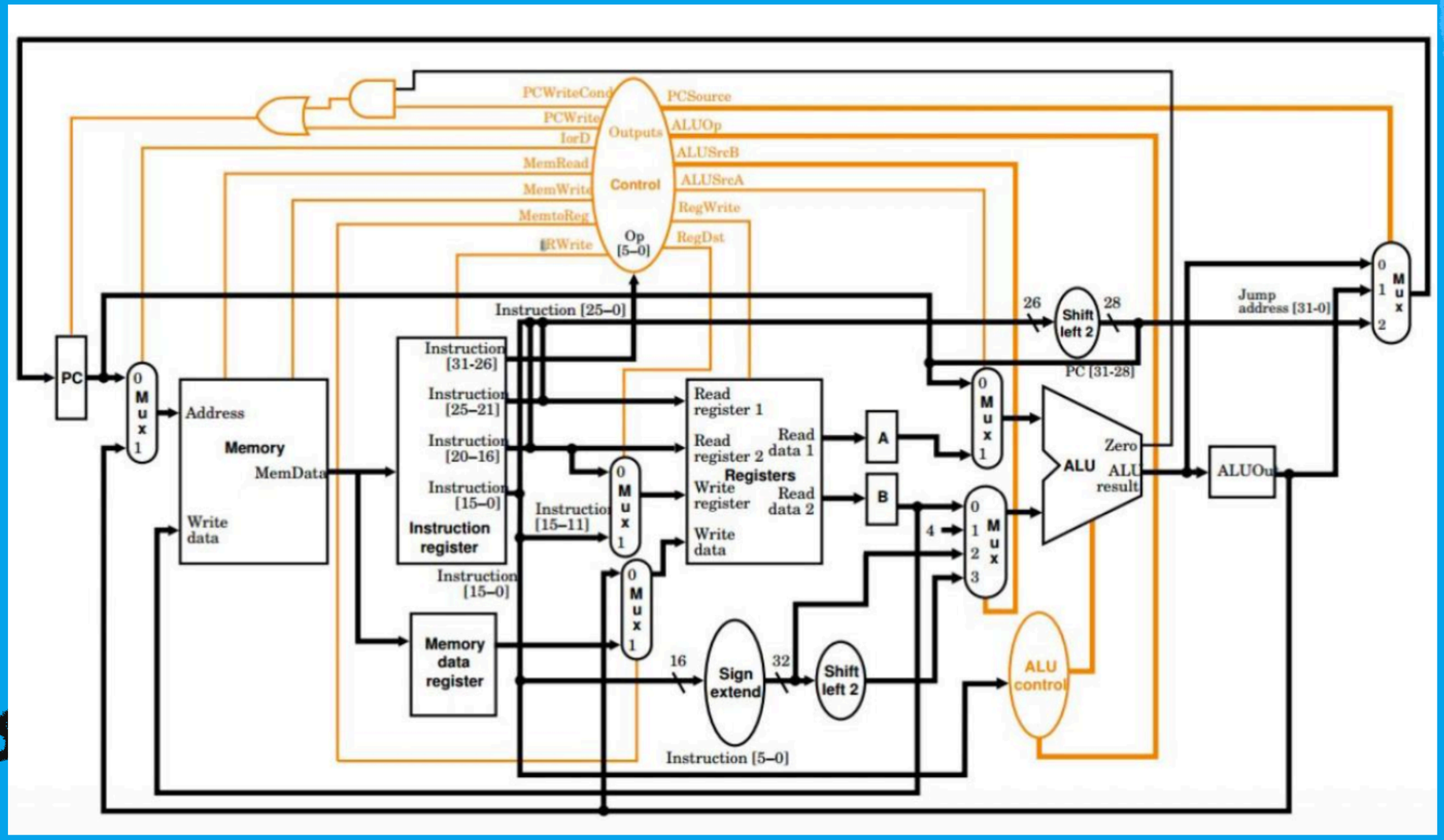
Multi-Cycle MIPS Processor Data Path

Understanding the Component Connections

- **High-Level Overview:**
  - Top-level architecture in VHDL connecting control and data path units
  - Integrates memory, ALU, PC, IR, MDR, register file, and control logic
  - Manages signal flow across multiple cycles

Multi-Cycle MIPS Processor Data Path

Understanding the Component Connections



🕹️Control Signal Coordination

Receives opcode/funct inputs → passes to Control Unit

- **Control Unit generates signals for:**
  - Register selection
  - ALU ops
  - Memory read/write
  - PC updates

🧐Testbench Development Overview

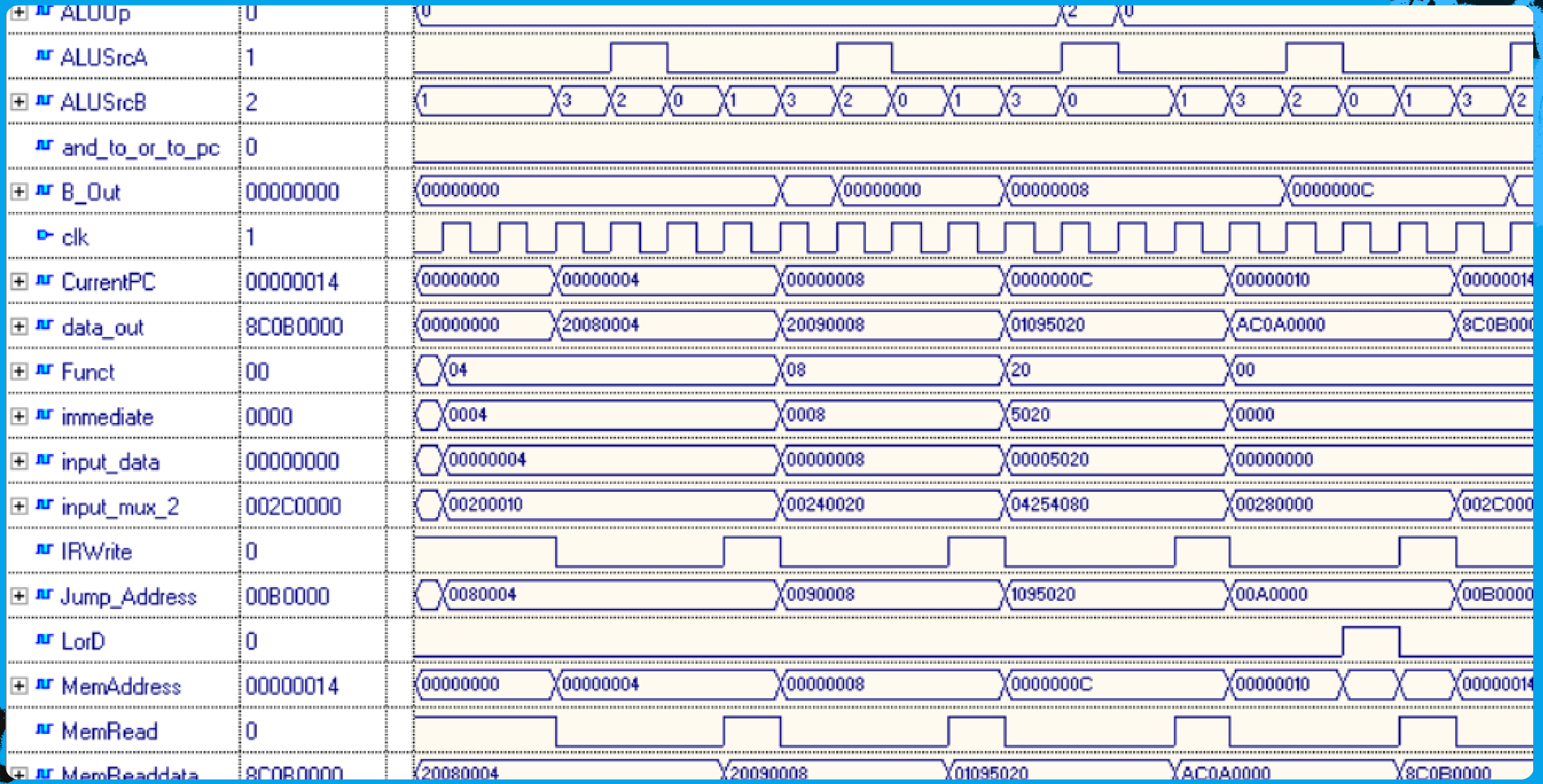
- As part of the verification phase of the MIPS-based processor project, a structured and modular approach was followed to validate the functionality of each component prior to integration.

- The testbenches were designed to simulate various operational scenarios, verify expected behaviors, and ensure each module meets its design specifications.

✍️Modules Covered in the Testbench Phase

- ALU
- ALU Control Unit
- ALUOut Register
- Control Unit
- Register File
- Generic Register
- Instruction Register
- Program Counter (PC)
- Instruction Memory
- Data Memory (MDR)
- Sign Extend Unit
- Shift Left 2
- Shift Left 2 for PC
- MUX 2-to-1
- MUX 3-to-1
- MUX 4-to-1
- AND Gate
- OR Gate
- Connection Path (Integration Testing)

Simulation Output



Thank you!