# Faculty of Computing

Computer Architecture

Spring 2025

LAB # 5

**Instructor**

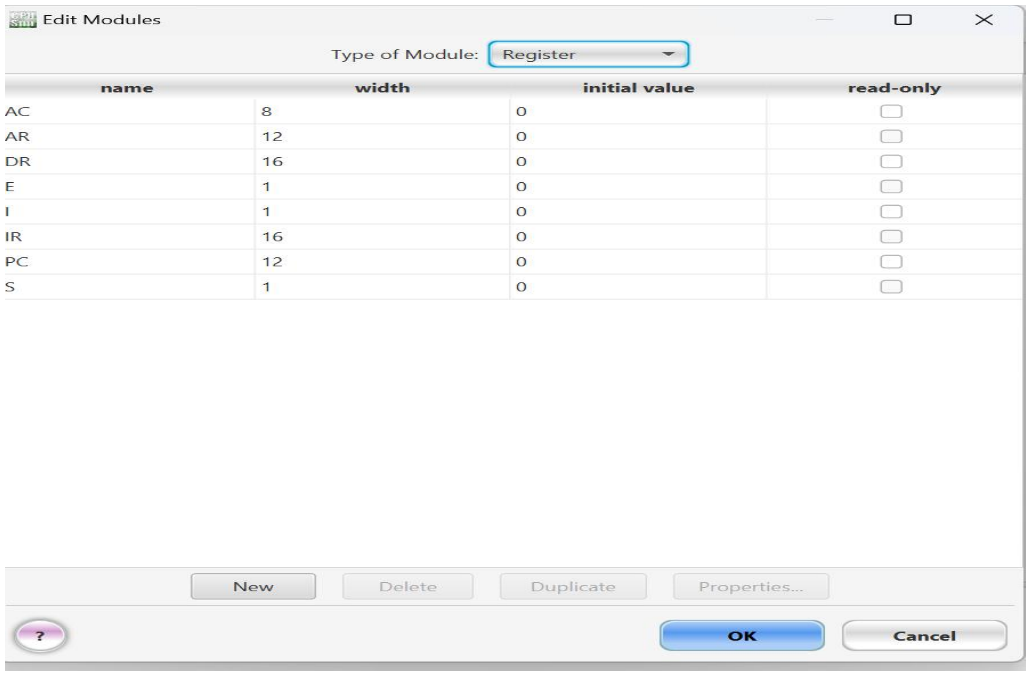
Ayesha Akram

TF, Faculty of Computing,

Riphah International University, Islamabad

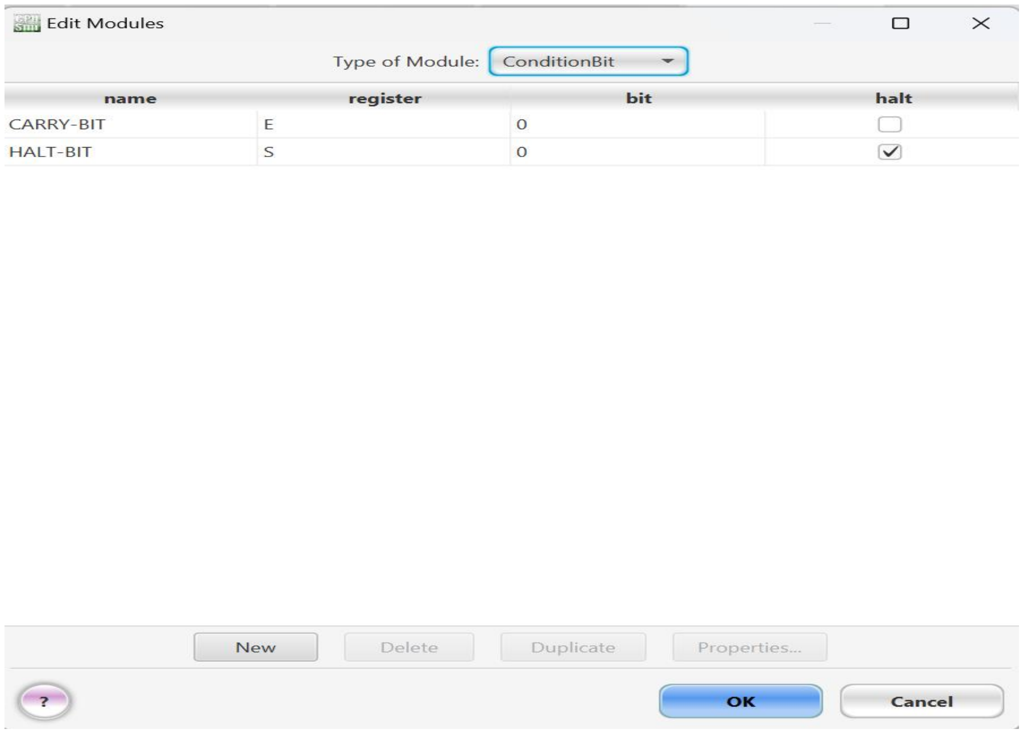
## Question No.05

Create a new base machine and change the bit width of a register (e.g., make AC 8-bit instead of 16-bit)



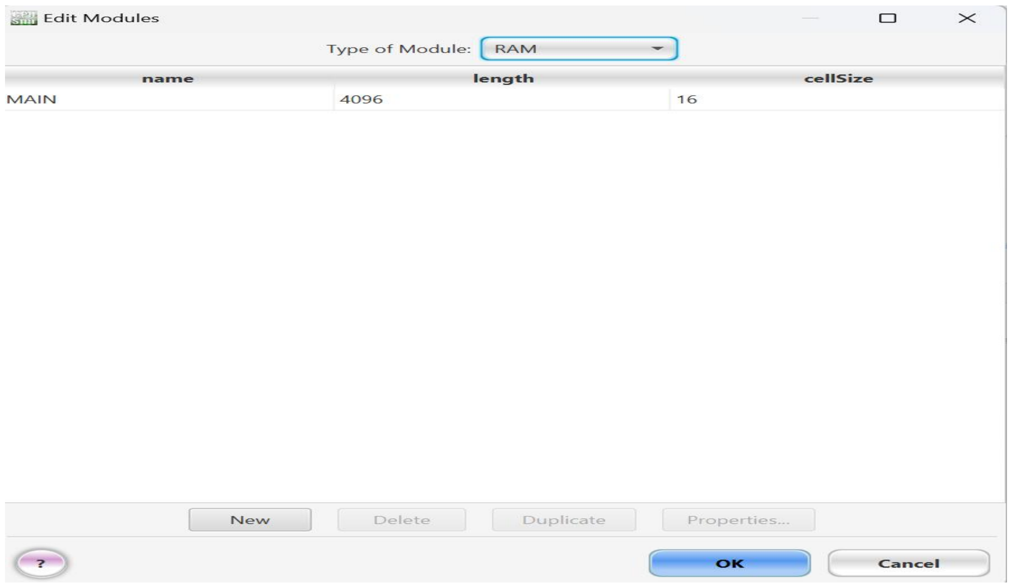
Type of Module: Register

| name | width | initial value | read-only |
|---|---|---|---|
| AC | 8 | 0 | ☐ |
| AR | 12 | 0 | ☐ |
| DR | 16 | 0 | ☐ |
| E | 1 | 0 | ☐ |
| I | 1 | 0 | ☐ |
| IR | 16 | 0 | ☐ |
| PC | 12 | 0 | ☐ |
| S | 1 | 0 | ☐ |



Type of Module: ConditionBit

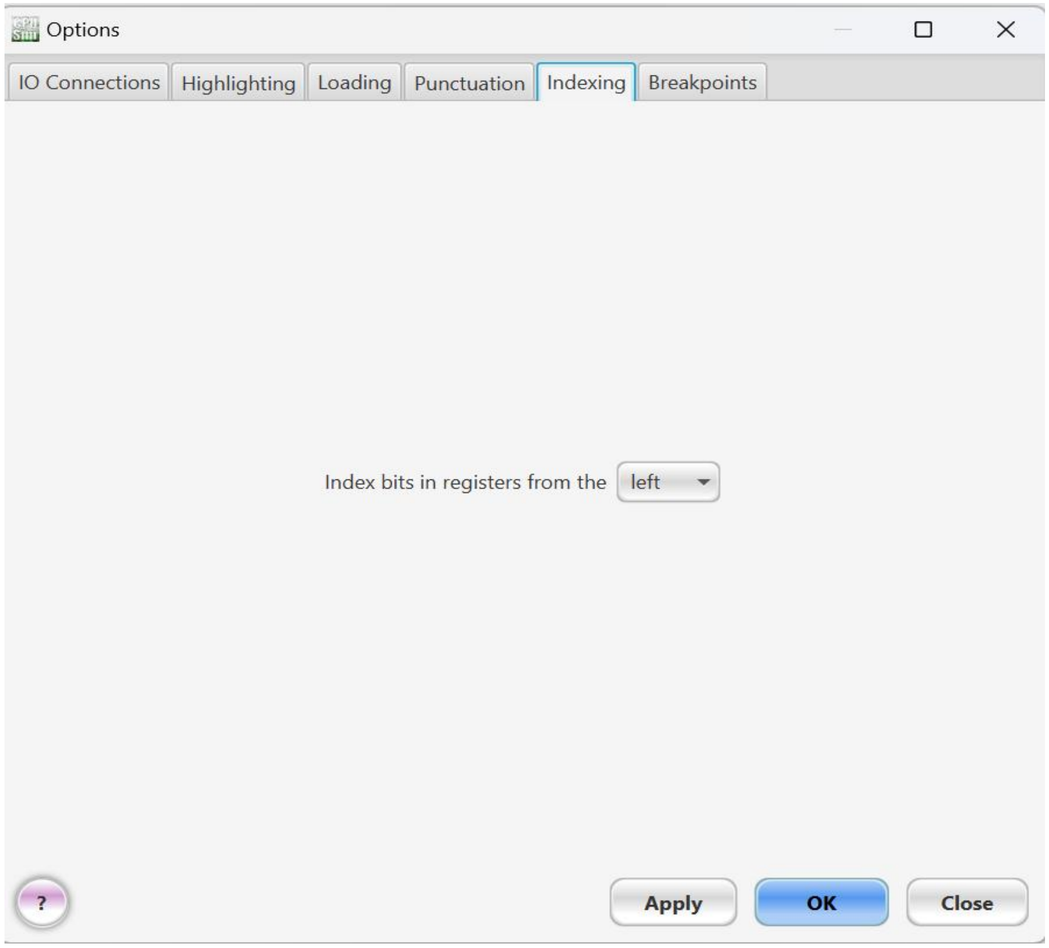
| name | register | bit | halt |
|---|---|---|---|
| CARRY-BIT | E | 0 | ☐ |
| HALT-BIT | S | 0 | ✓ |



Edit Modules

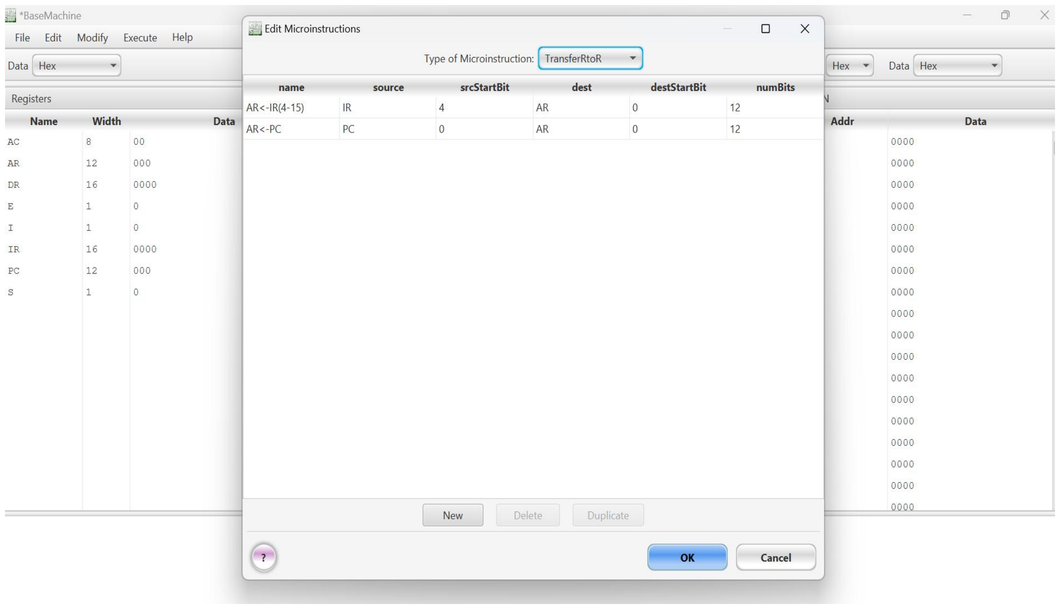
Type of Module: RAM

| name | length | cellSize |
|---|---|---|
| MAIN | 4096 | 16 |



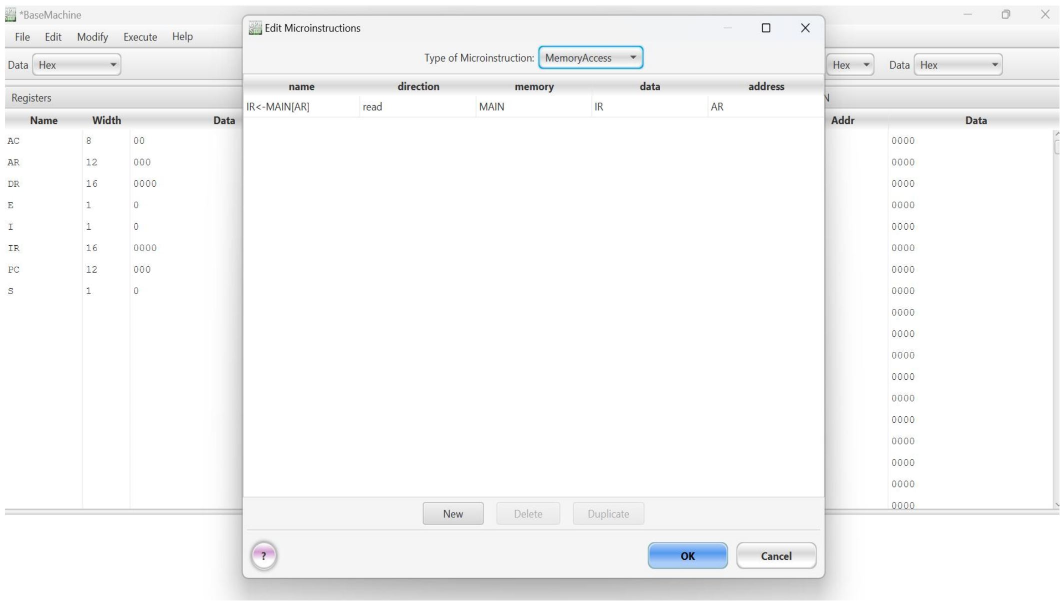
Options

Index bits in registers from the left


Type of Microinstruction: TransferRtoR

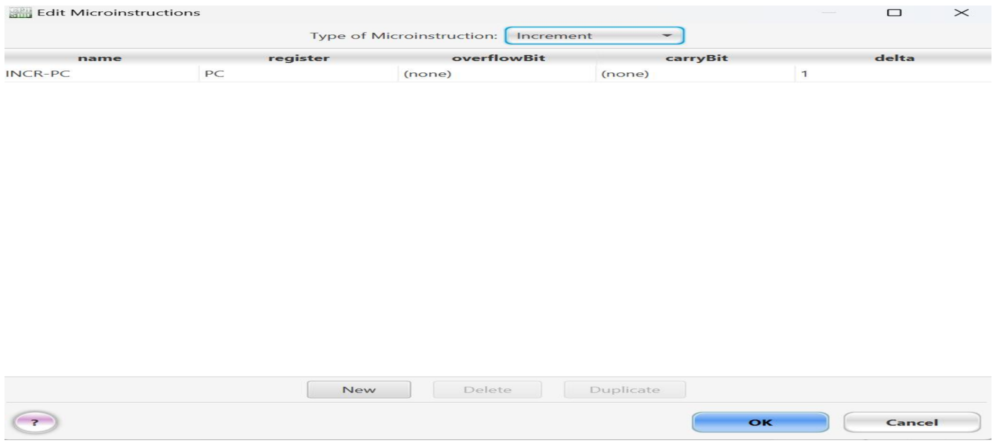
| name | source | srcStartBit | dest | destStartBit | numBits |
|---|---|---|---|---|---|
| AR<-IR(4-15) | IR | 4 | AR | 0 | 12 |
| AR<-PC | PC | 0 | AR | 0 | 12 |

| Name | Width | Data |
|---|---|---|
| AC | 8 | 00 |
| AR | 12 | 000 |
| DR | 16 | 0000 |
| E | 1 | 0 |
| I | 1 | 0 |
| IR | 16 | 0000 |
| PC | 12 | 000 |
| S | 1 | 0 |


Type of Microinstruction: MemoryAccess

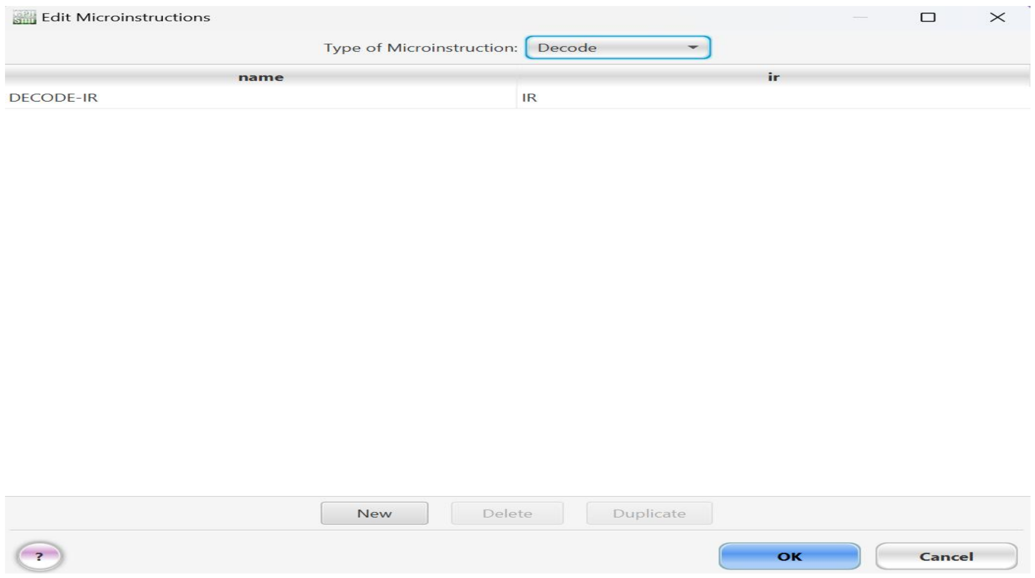
| name | direction | memory | data | address |
|---|---|---|---|---|
| IR<-MAIN[AR] | read | MAIN | IR | AR |

| Name | Width | Data |
|---|---|---|
| AC | 8 | 00 |
| AR | 12 | 000 |
| DR | 16 | 0000 |
| E | 1 | 0 |
| I | 1 | 0 |
| IR | 16 | 0000 |
| PC | 12 | 000 |
| S | 1 | 0 |



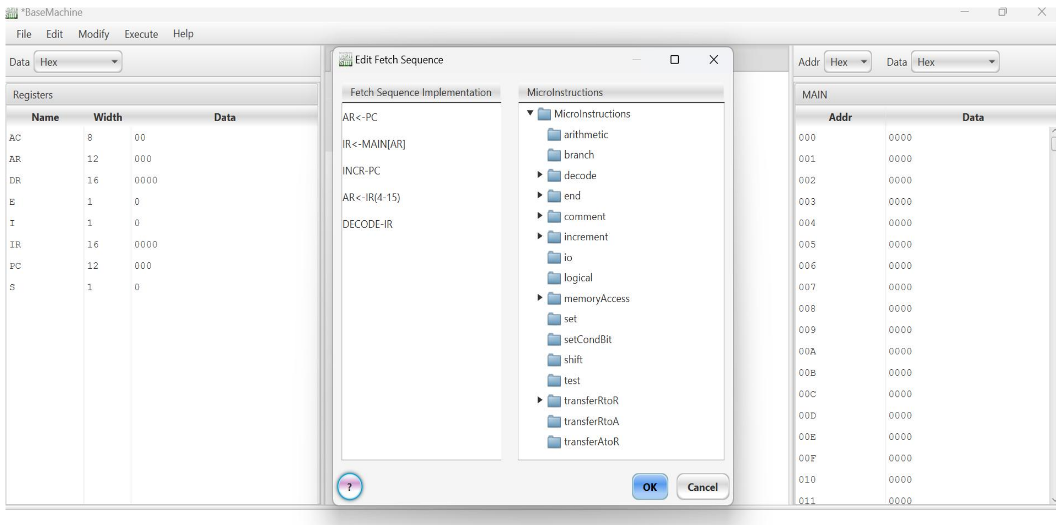
Type of Microinstruction: Increment

| name | register | overflowBit | carryBit | delta |
|---|---|---|---|---|
| INCR-PC | PC | (none) | (none) | 1 |



Type of Microinstruction: Decode

| name | ir |
|---|---|
| DECODE-IR | IR |



Data: Hex

Registers

| Name | Width | Data |
|---|---|---|
| AC | 8 | 00 |
| AR | 12 | 000 |
| DR | 16 | 0000 |
| E | 1 | 0 |
| I | 1 | 0 |
| IR | 16 | 0000 |
| PC | 12 | 000 |
| S | 1 | 0 |

Fetch Sequence Implementation

AR<-PC
IR<-MAIN[AR]
INCR-PC
AR<-IR(4-15)
DECODE-IR

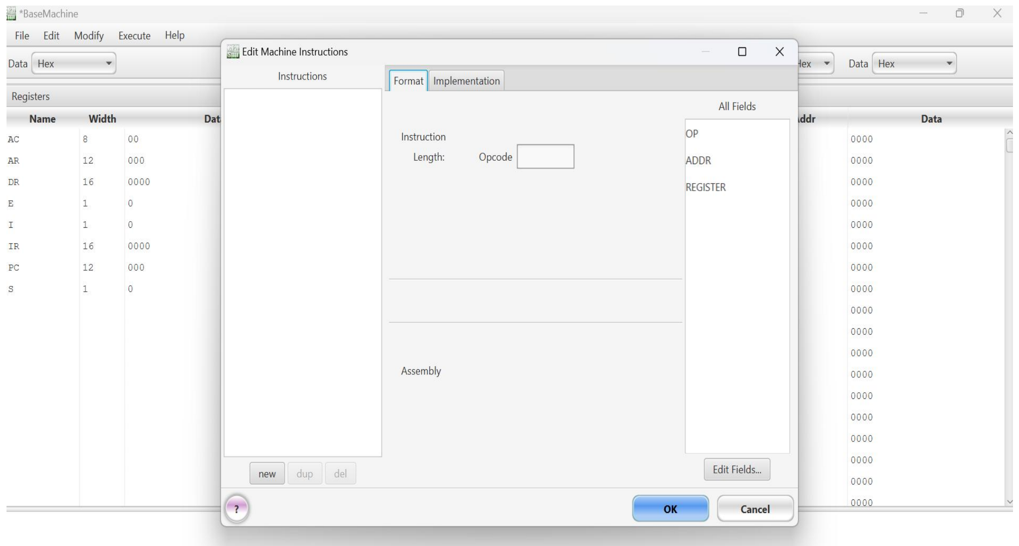
MicroInstructions

- MicroInstructions
  - arithmetic
  - branch
  - decode
  - end
  - comment
  - increment
  - io
  - logical
  - memoryAccess
  - set
  - setCondBit
  - shift
  - test
  - transferRtoR
  - transferRtoA
  - transferAtoR

Addr: Hex Data: Hex

MAIN

| Addr | Data |
|---|---|
| 000 | 0000 |
| 001 | 0000 |
| 002 | 0000 |
| 003 | 0000 |
| 004 | 0000 |
| 005 | 0000 |
| 006 | 0000 |
| 007 | 0000 |
| 008 | 0000 |
| 009 | 0000 |
| 00A | 0000 |
| 00B | 0000 |
| 00C | 0000 |
| 00D | 0000 |
| 00E | 0000 |
| 00F | 0000 |
| 010 | 0000 |
| 011 | 0000 |



| Name | Width | Data |
|---|---|---|
| AC | 8 | 00 |
| AR | 12 | 000 |
| DR | 16 | 0000 |
| E | 1 | 0 |
| I | 1 | 0 |
| IR | 16 | 0000 |
| PC | 12 | 000 |
| S | 1 | 0 |

## Question No.01

**Write a detailed explanation of how the Fetch-Decode-Execute cycle works**.

The CPU runs programs using a repeating process called the **Fetch-Decode-Execute cycle**. It retrieves instructions from memory, figures out what to do, and then performs the action.

---

## 1 Fetch Stage

The CPU **fetches** the next instruction from memory.

- The **Program Counter (PC)** holds the memory address of the next instruction.
- The address is copied to the **Memory Address Register (MAR)**.
- The instruction is fetched from memory and stored in the **Memory Data Register (MDR)**.
- The **MDR** transfers the instruction to the **Instruction Register (IR)**.
- The **PC** increases to point to the next instruction.

✓ **Result:** The CPU now has the instruction ready to decode.

---

## 2 Decode Stage

The CPU **decodes** the instruction to understand what to do.

- The **Control Unit (CU)** reads the instruction in the **IR**.
- It breaks the instruction into:
  - **Opcode:** Tells the CPU what action to perform (e.g., ADD, LOAD).
  - **Operands:** The data or memory locations involved.

- The **CU** prepares the necessary control signals for the next stage.

✅ **Result:** The CPU knows what operation to perform and what data to use.

---

## 3 Execute Stage

The CPU **executes** the instruction.

- The **Arithmetic Logic Unit (ALU)** performs calculations or logic operations if needed.
- Data may be moved between registers or memory.
- The result is stored in the correct place (register or memory).
- The **Status Flags** (e.g., Zero flag, Carry flag) are updated if necessary.

✅ **Result:** The instruction is completed successfully.

---

### Cycle Repeats

The cycle repeats — the CPU goes back to the **Fetch** stage to handle the next instruction, continuing until the program ends or a **HALT** instruction is reached.

## Question No.02

**Use a simple instruction as an example and describe each step.**

### Instruction: LOAD R1, 5

(→ Load the number **5** into register **R1**)

---

### Fetch

- **Program Counter (PC)** holds the address of the next instruction.
- CPU fetches `LOAD R1, 5` from memory and stores it in the **Instruction Register (IR)**.
- **PC** increases to point to the next instruction.

✅ **Now the CPU has the instruction.**

---

### Decode

- The **Control Unit (CU)** reads the instruction from the **IR**.
- It breaks it into:
  - **Opcode:** `LOAD` (the action — load data).
  - **Operands:** `R1` (where to store the data) and `5` (the data to load).

✅ **The CPU knows what the instruction wants to do.**

---

### Execute

- The CPU moves the number **5** into register **R1**.
- **R1** now holds **5**.

✅ **Instruction complete — the CPU moves on to the next one!**

## Question No.03

**Explain the role of PC, AR, IR, AC and DR in your own words.**

**Program Counter (PC)** — Keeps track of where the CPU is in the program. It holds the address of the next instruction to fetch. After fetching, it moves to the next instruction's address.

**Address Register (AR)** — Holds the **memory address** the CPU is currently working with — whether it's fetching an instruction or accessing data.

**Instruction Register (IR)** — Stores the **current instruction** that's being processed after fetching. The Control Unit decodes the instruction from here.

**Accumulator (AC)** — A special register that holds intermediate results from calculations and the final result of arithmetic or logic operations. For example, if you add two numbers, the result usually ends up in the AC.

**Data Register (DR)** — Temporarily holds data fetched from memory or data that needs to be written back to memory. It's like a holding area for data moving between memory and the CPU.

## Question No.04

**What is the function of the Arithmetic Logic Unit (ALU) in CPU operations? How does ALU interact with registers and memory?**

### Function of the ALU (Arithmetic Logic Unit)

The **ALU** is the part of the CPU that handles all the **mathmatical** and **logic** operations. Its handles:

- **Arithmetic operations:** Add, subtract, multiply, divide.
- **Logic operations:** AND, OR, NOT, XOR — used for comparisons and decision-making.
- **Shift operations:** Move bits left or right (e.g., for multiplication or division by powers of 2).
- **Comparisons:** Check if numbers are equal, greater, or less (helps with decision-making in programs).

## How the ALU interacts with registers and memory:

**Fetch Data from Registers:** The **Control Unit (CU)** directs the ALU to get data from specific registers — like the **Accumulator (AC)** or **Data Register (DR)** — to perform calculations.

**Perform Operation:** The ALU processes the data according to the **opcode** in the instruction (e.g., add two numbers).

**Store the Result:** The result goes back into a register — often the **Accumulator (AC)** — or sometimes directly to memory.

**Update Flags:** The ALU updates **status flags** (e.g., Zero Flag if the result is 0, Carry Flag for overflow). This helps with decision-making in later instructions.