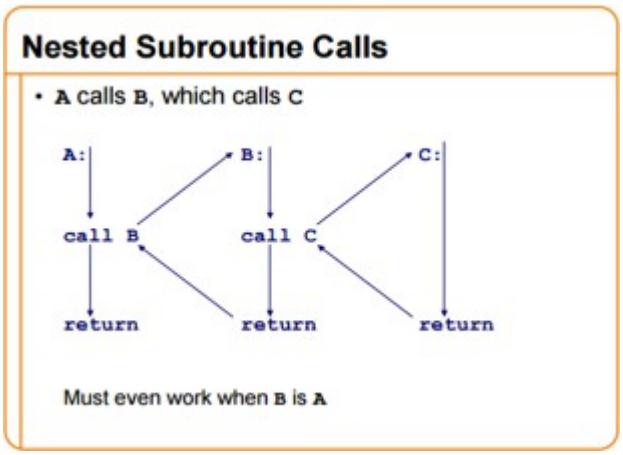
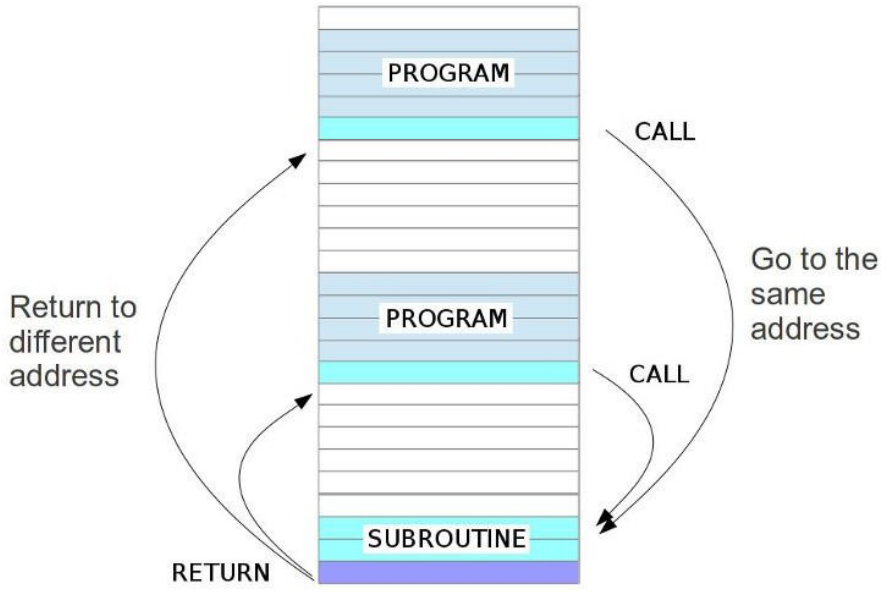
1. Describe the operation of nested subroutine calls and support your explanation with a clear diagram.

Subroutine is a small set of instructions stored in memory to perform a specific task. (Different from main program/function)

The processor can **jump to the subroutine,** execute it, and then **return** to the main program. This makes the portion of code that may be called and executed anywhere and anytime in a program.

Instead of writing the same steps again and again, it can be used in multiple usage of same process.





2. Discuss the two major tasks carried out by a control unit

The control unit (CU) of a CPU has two main tasks:

Instruction Fetch:

The Control Unit **goes to the computer's memory** and looks at the address given by the **Program Counter (PC)**.
The Program Counter is like a pointer that tells the Control Unit where the next instruction is stored.
Then, the CU **reads the instruction from that memory location** and **stores it in the Instruction Register (IR)** so it can use it in the next step.

Instruction Decode:

Once the instruction is fetched and stored in the **Instruction Register (IR)**, the Control Unit **reads and understands** it. The Control Unit After understanding the instruction, it **sends control signals** to these parts, telling them exactly what to do.

3. List the main features of a microcomputer.

- *Small Size – Easy to fit on a desk or carry around.*
- *Low Cost – Affordable for individuals and small businesses.*
- *Reliable – Works well and doesn't crash easily.*
- *Versatile – Can do many different tasks (like games, office work, etc.).*
- *Low Power Dissipation – Uses less electricity and produces less heat.*
- *Expandable and Upgradeable – You can add more memory, storage, or other parts.*
- *Multimedia Features – Can play videos, music, and display graphics.*
- *Security Features – Keeps your data safe with passwords and protection.*
- *Energy Efficient – Saves power, good for the environment.*
- *Cost Effective – Gives good performance without being expensive.*

4. How much data can be accessed using a 32-bit address bus, assuming the memory is byte addressable?

To calculate how much data can be accessed using a **32-bit address bus** with **byte-addressable memory**, you can use the formula:

$$\text{Total addressable memory} = 2^{\text{number of address lines}} \text{ bytes}$$

So for a **32-bit address bus**:

2^{32} locations = $2^2 * 2^{10} * 2^{10} * 2^{10}$ locations

1 GB = 1024 MB — 2^{32} locations = $2^2 * 2^{10} * 2^{10}$ * 1 KB [2^{10} = 1 KB]

1 MB = 1024 KB — 2^{32} locations = $2^2 * 2^{10}$ * 1024 KB

1 KB = 1024 B — 2^{32} locations = 2^2 * 1024 MB

1 B = 8 bits — 2^{32} locations = 4 GB of location

5. What are two key functions of commonly used CPU registers?

1. Data Storage (Accumulator):

- Stores temporary data that the CPU is currently working with.
- Holds intermediate results during arithmetic and logic operations.
- Speeds up processing by avoiding frequent memory access.

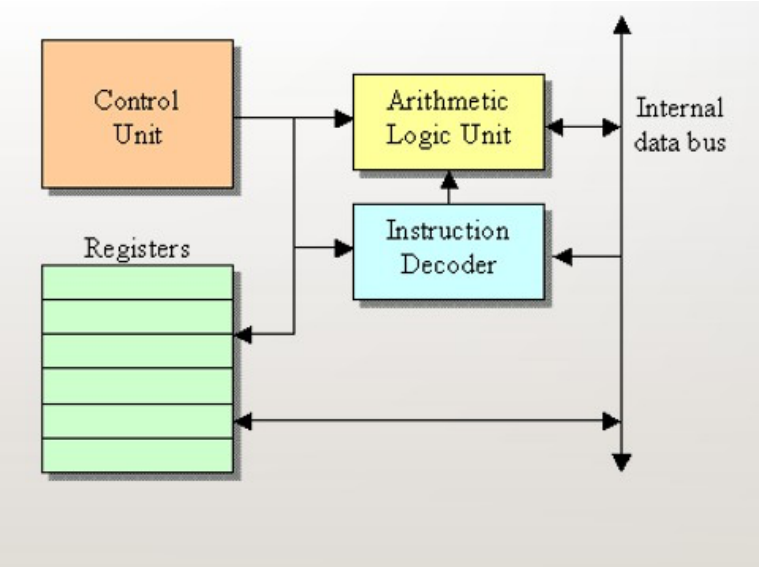
---

2. Instruction Control (PC & IR):

- Program Counter (PC) holds the address of the next instruction to execute.
- Instruction Register (IR) stores the current instruction being decoded and executed.
- Helps in step-by-step execution of the program.
- Ensures the CPU follows the correct order of instructions.

6. With the help of a neat diagram, explain the internal components of a microprocessor.

A **microprocessor** is a small chip that works as the **brain of a computer**.
It **reads instructions**, **processes data**, and **controls** all the main tasks of the computer.



- **Control Unit (CU):**
  - Directs the processor's operations by fetching, decoding, and executing instructions (Controls the whole processor).
  - Tells the other parts what to do.
- **Arithmetic Logic Unit (ALU):**
  - Does math operations (like add, subtract).
  - Performs logical checks (like AND, OR).
- **Registers:**
  - Small storage inside the CPU.
  - Temporarily holds data, instructions, and results.
- **Instruction Decoder:**
  - Understands the instruction fetched.
  - Breaks it down and passes signals to other units.
- **Internal Data Bus:**
  - Carries data between all parts of the processor.

7. What are the two important characteristics of a multicycle implementation?

Multicycle = One instruction is split into many smaller steps → each done in separate clock cycles.

Instruction Execution in Multiple Steps:

Each instruction is broken into smaller steps like **fetch, decode, execute, memory access, and write- back**, and each step takes **one clock cycle**.

Reuse of Hardware Components:

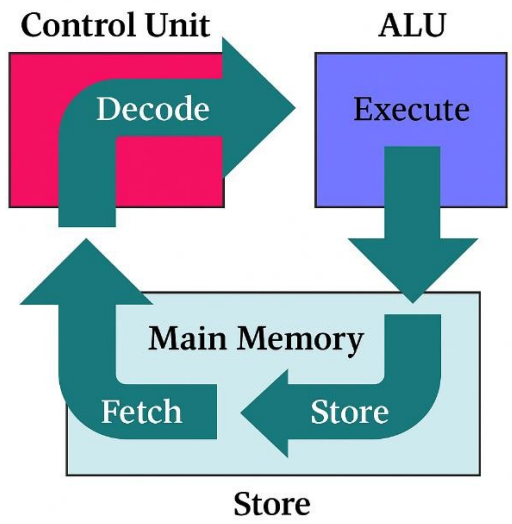
The same hardware units (like ALU, memory, etc.) are **used for different steps** of different instructions, reducing the overall hardware required.

potentially faster execution times for certain instructions

- Reuse of Hardware Components
- Variable Execution Time
- Control Logic Complexity
- Efficiency and Performance

8. Define the various phases involved in a machine cycle.

A **machine cycle** is the basic operation cycle of a computer, in which the CPU fetches, decodes, executes, and stores instructions. The various **phases involved in a machine cycle** are:



Fetch:

- The CPU reads the instruction (or) get the operand from memory.
- The Program Counter (PC) provides the address of the next instruction.
- The instruction is loaded into the Instruction Register (IR).

Decode:

- The CPU decodes the fetched instruction to understand what action is needed.
- The control unit interprets the opcode and determines the necessary operations.

Execute:

- The decoded instruction is executed.
- This may involve arithmetic/logical operations, data transfer.

Store:

- The result of the executed instruction is stored in memory or a register.
- This ensures data is saved for later use or for the next instruction.

9. Explain the role played by the stack during a subroutine call.

Storing the Return Address

When a subroutine is called, the **address of the next instruction** (where the program should continue after the subroutine ends) is **pushed onto the stack**.

After the subroutine finishes, the processor **pops** this address from the stack to return to the main program.

Handling Nested

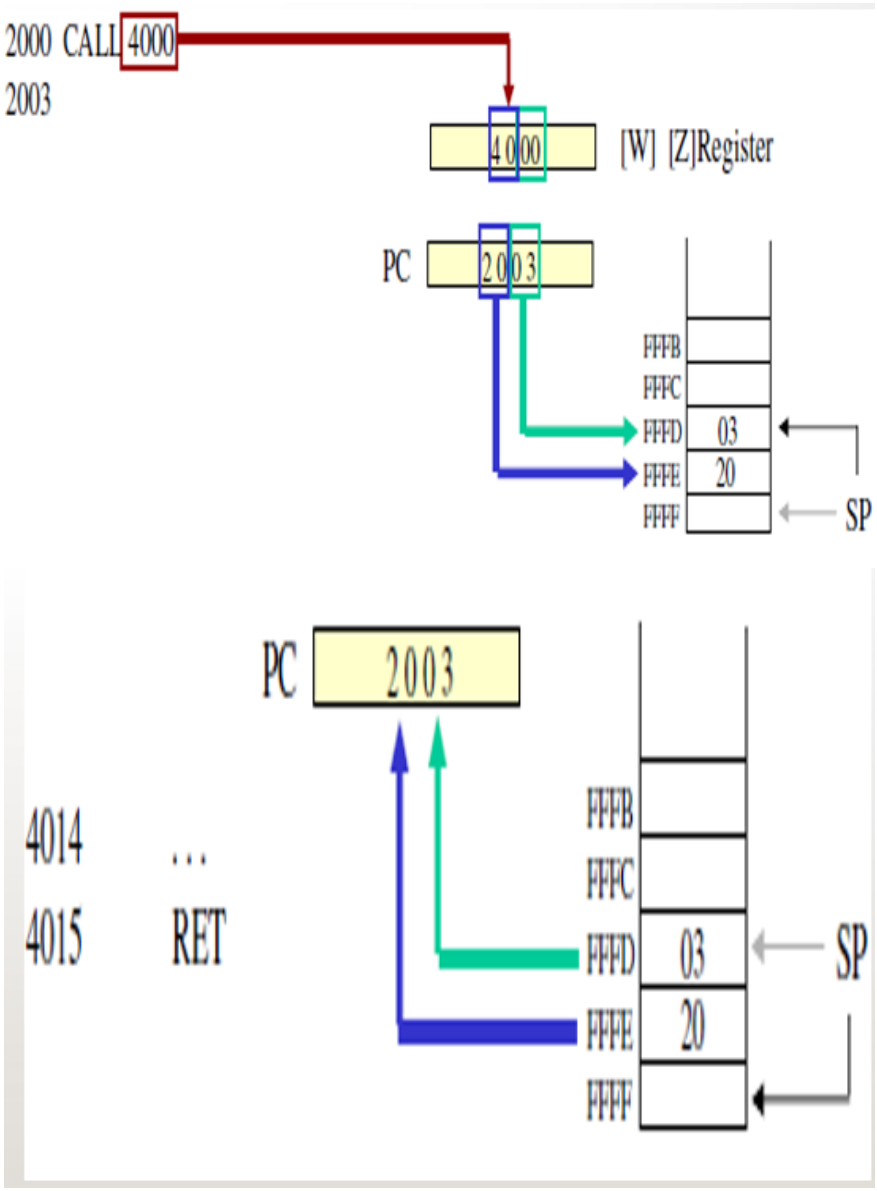
The stack allows multiple subroutine calls to happen one after another.

Each call's **return address and local data** are kept separately in the stack, maintaining proper execution.

The stack **ensures smooth subroutine execution** by managing return addresses, register data, local variables, and allowing nested calls.

The return address is stored in the stack in **reverse order**—first the **higher byte**, then the **lower byte**

—as the stack operates in **Last-In-First-Out (LIFO)** manner.

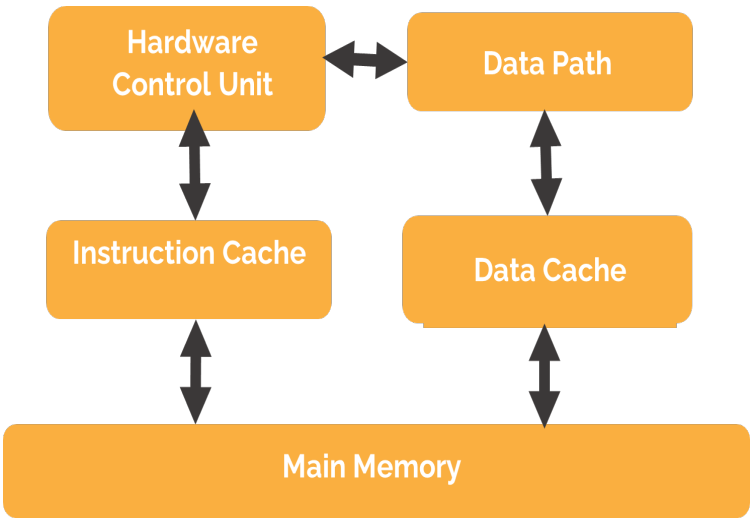


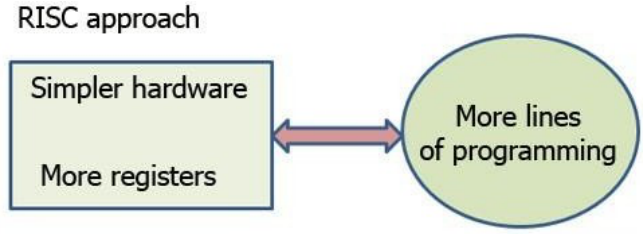
It ensures the main program can resume correctly after the subroutine completes.

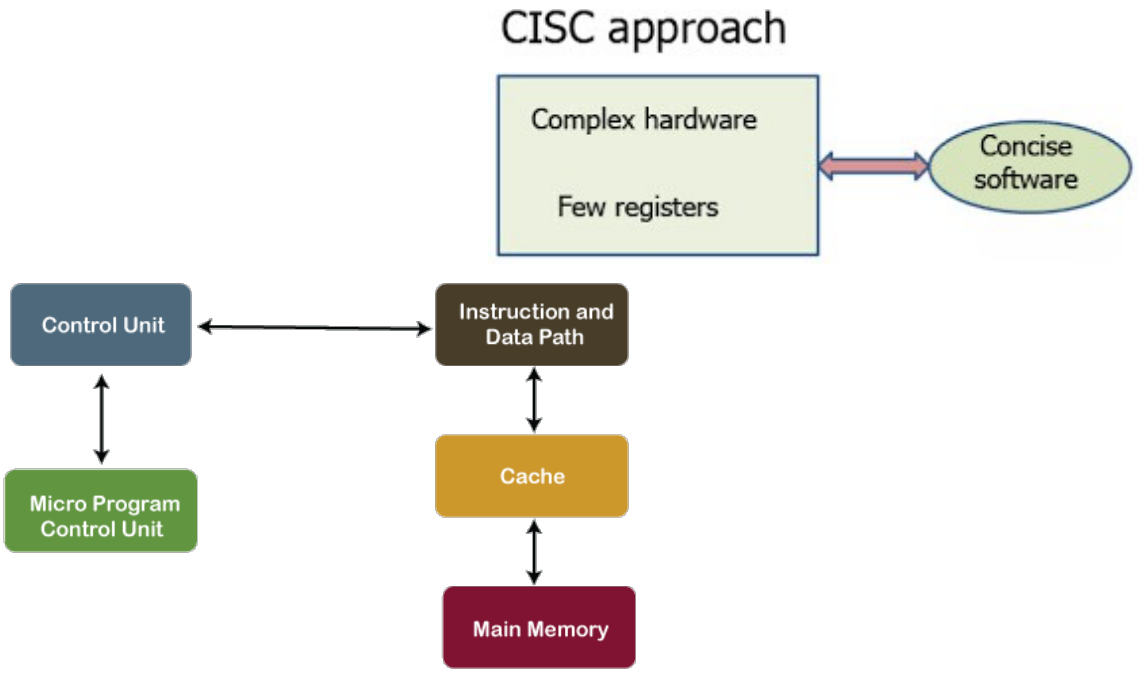
10. Differentiate between the features of RISC and CISC architectures.

Feature	*RISC (Reduced Instruction Set Computer)*	*CISC (Complex Instruction Set Computer)*
Instruction Size	Fixed, simple instructions	Variable-length, complex instructions
Instruction Count	Requires more instructions for complex tasks	Fewer instructions as each one does more work
Registers	Less registers	Uses more registers
Clock Cycles per Task	More clock cycles needed due to multiple simple instructions	Fewer clock cycles as complex instructions do more in one step
Cycle Time	Instructions take a varying amount of cycle time	Instructions take one cycle time
Programming	Extensive use of microprogramming	Complexity in compiler
Performance	High performance in applications needing fast instruction throughput	Better for applications that benefit from reduced code size

RISC is widely used in environments where efficiency and performance-per-watt are crucial, such as in mobile devices, tablets, and embedded systems



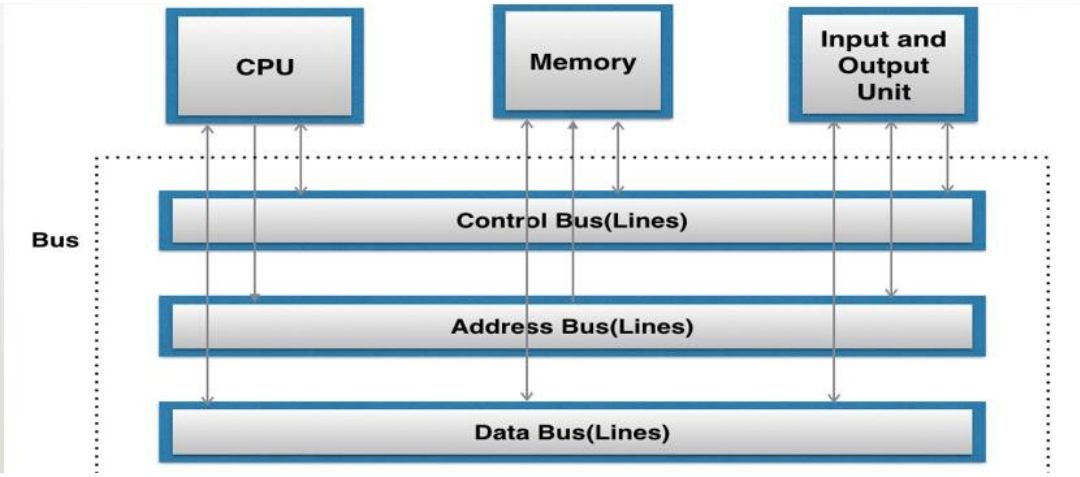




CISC Architecture

CISC is often found in general-purpose computing, where its wide range of instructions and capabilities can simplify software development. Intel's x86 architecture, prevalent in desktop and server processors

11. How does the CPU communicate with memory and I/O devices through different types of buses? Support your answer with a diagram.

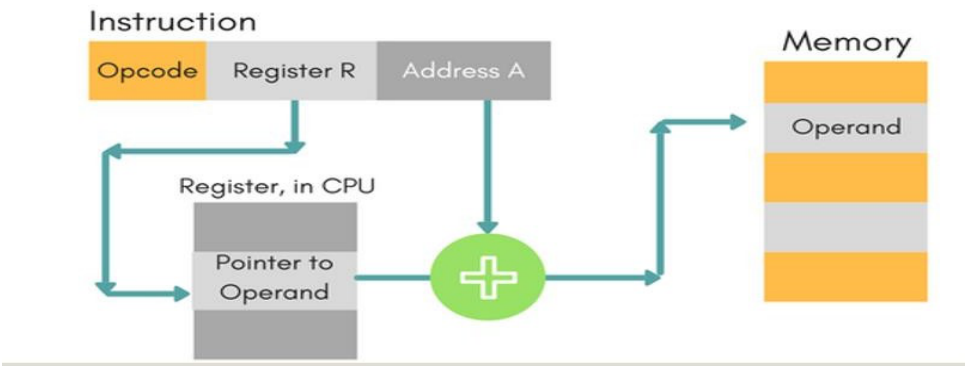


Type of Bus	Function	Direction
Address Bus	Carries the address of the memory location or I/O device to be accessed.	Unidirectional (CPU → Memory/I/O)
Data Bus	Transfers actual data between the CPU, memory, and I/O devices.	Bidirectional (CPU ↔ Memory/I/O)
Control Bus	Sends control signals to manage operations like read, write, or interrupt.	Both directions (Mainly CPU → Others, sometimes CPU ← Status)

12. Explain how displacement addressing mode works and provide a diagram to illustrate it.

Displacement addressing mode is an addressing method where the effective address is calculated by adding a constant (displacement) to the contents of a register.

EA = A + (R)



13. Give examples of two arithmetic and two data transfer instructions, and explain how they operate.

Arithmetic Instructions

1. ADD (Addition)
   - Example: ADD A, B
   - Operation: Adds the value of register B to register A, and stores the result in A.
   - Explanation:
     If A = 05H and B = 03H, after
     ADD A, B, A = 08H.
2. SUB (Subtraction)
   - Example: SUB A, #02H
   - Operation: Subtracts immediate value 02H from register A.
   - Explanation:
     If A = 07H, after SUB
     A, #02H, A = 05H.

Data Transfer Instructions

1. MOV (Move data)
   - Example: MOV A, R1
   - Operation: Copies the contents of register R1 into register A.
   - Explanation:
     If R1 = 04H, after
     MOV A, R1, A = 04H.
2. LDA (Load Accumulator)
   - Example: LDA 2050H
   - Operation: Loads data from memory address 2050H into the accumulator.
   - Explanation:
     If memory at 2050H = 2AH, after
     LDA 2050H, A = 2AH.

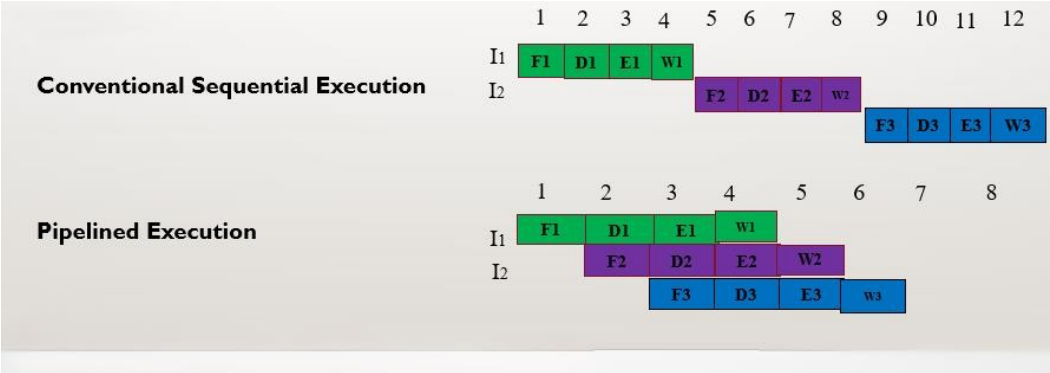
14. Outline the advantages of pipelined execution over traditional sequential execution, with an example.

Increased Throughput

- **Pipelining** allows overlapping of instruction stages (like Fetch, Decode, Execute, etc.), so more instructions are completed in a given time

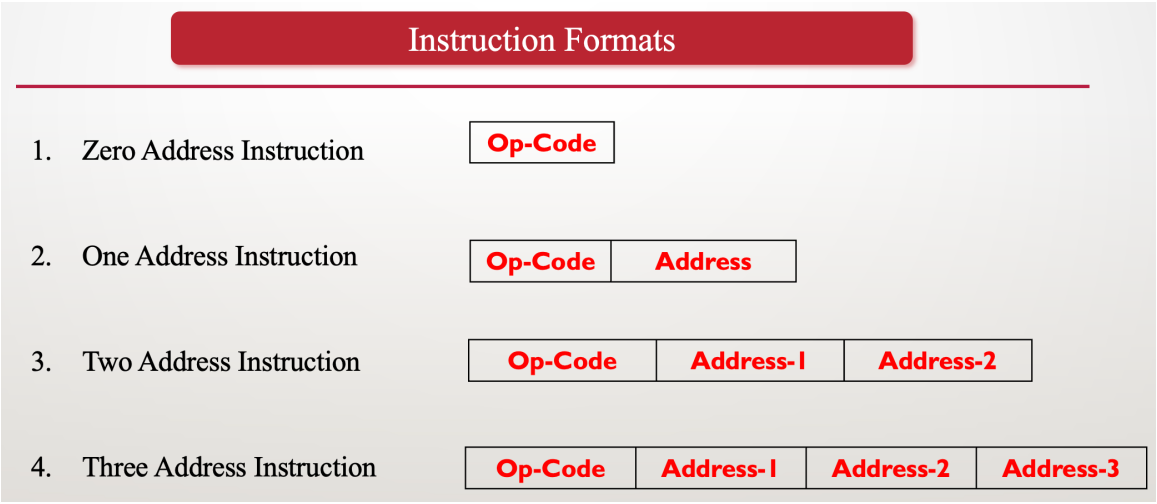
In sequential execution, many units stay idle while waiting for other stages to

complete. Improved Performance without Increasing Clock Speed



15. Categorize instruction formats based on their opcode and addressing fields, and give an example instruction for each category.

Format	Structure	Example	Meaning
Zero-Address	Opcode	ADD	TOS ← TOS + Next TOS
One-Address	Opcode + 1 Address	ADD B	AC ← AC + M[B]
Two-Address	Opcode + 2 Addresses	ADD R1, B	R1 ← R1 + M[B]
Three-Address	Opcode + 3 Addresses	ADD R1, A, B	R1 ← M[A] + M[B]



Zero Address Instructions

$X = (A + B) * (C + D)$

Assembly Language Instruction	Stack Transfer Operation	Explanation
PUSH A	TOS ← A	Push A To The Top Of Stack (TOS).
PUSH B	TOS ← B	Push B To The Top Of Stack (TOS).
ADD	TOS ← (A + B)	Add (A + B) And Push To The Top Of Stack (TOS).
PUSH C	TOS ← C	Push C To The Top Of Stack (TOS).
PUSH D	TOS ← D	Push D To The Top Of Stack (TOS).
ADD	TOS ← (C + D)	Add (C + D) And Push To The Top Of Stack (TOS).
MUL	TOS ← (C + D) * (A + B)	Multiply (C + D) And (A + B) And Push To The Top Of Stack (TOS).
POP X	M [X] ← TOS	Store the contents of the Top Of Stack TOS ((C + D) * (A + B)) Into memory address X

One Address Instructions

LOAD	A	A C <- M [A]
ADD	B	A C <- A C + M [B]
STORE	T	M [T] <- A C
LOAD	C	A C <- M [C]
ADD	D	A C <- A C + M [D]
MUL	T	A C <- A C • M [T]
STORE	X	M [X] <- A C

Two Address Instructions

MOV R I ,A	R I <-- M [A]
ADD R I , B	R I <-- R I + M [B]
MOV R2 , C	R 2 <-- M [C]
ADD R2 , D	R 2 <-- R 2 + M [D]
MUL R I , R2	R I <-- R I * R 2
MOV X , R I	M [X] <-- R I

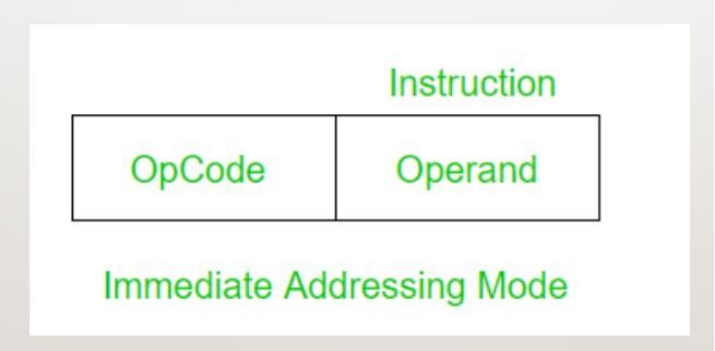
Three Address Instructions

ADD R I ,A , B	R I <-- M [A] + M [B]
ADD R2 , C , D	R 2 <-- M [C] + M [D]
MUL X , R I , R2	M [X] <-- R I * R 2

16. Demonstrate the operations of different addressing modes with example instructions.

Addressing Mode
Immediate
Direct
Indirect
Register
Register Indirect
Displacement
Stack (Implied Addressing)

1. *Immediate Addressing Mode*



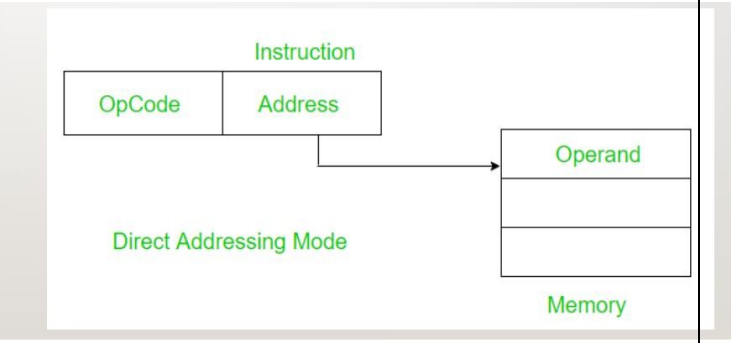
- Operand is part of the instruction.
- Data is directly given in the instructi

Example: MOV AX, 2000H

Operation: AX ← 2000H

---

2. *Direct Addressing Mode*



Instruction contains the memory address of th operand.

- Example: MOV AX, [1592H]

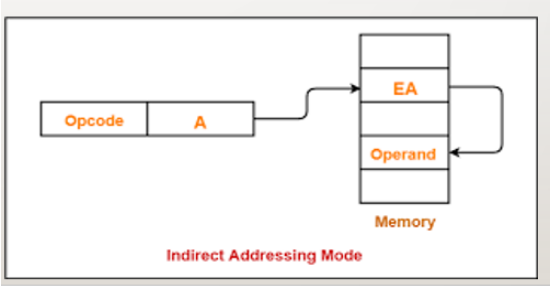
Operation: AX ← M[1592H]

Fetches value from memory location 1592H.

3. *Indirect Addressing Mode*

Instruction gives address of a memory location that contains the effective address. Example: MOV AX, @2005H

Operation: AX ← M[M[2005H]]



Double memory access: first fetch address from 2005H, then use that address to get data.

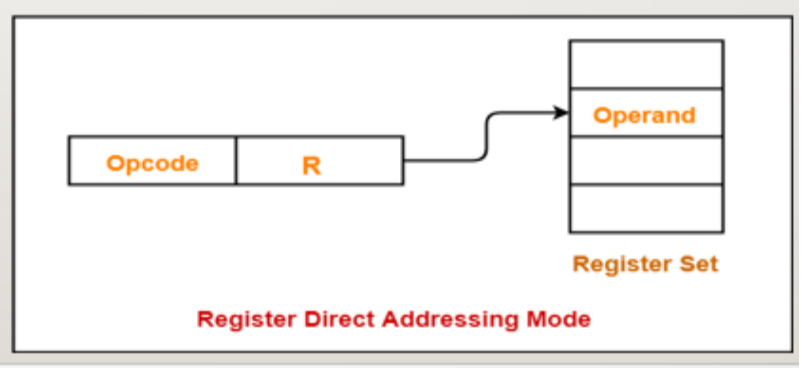
4. *Register Addressing Mode*

Operand is stored in a regist

Example: MOV AX, BX

Operation: AX ← BX

Direct transfer between CPU regist

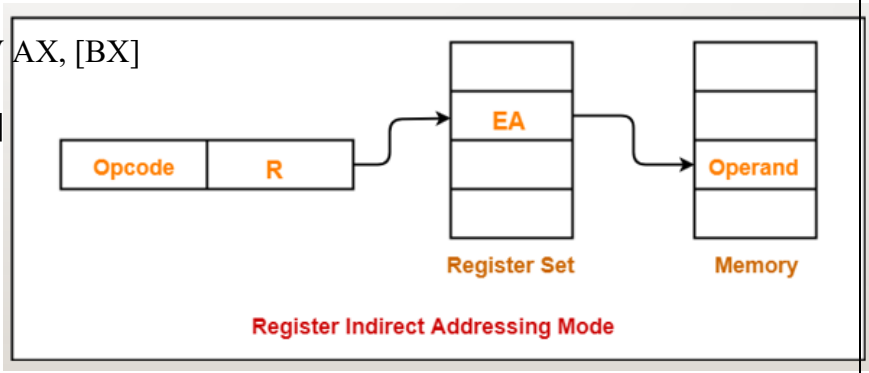


5. *Register Indirect Addressing Mode*

Register contains the address of the operand. Example: MOV AX, [BX]

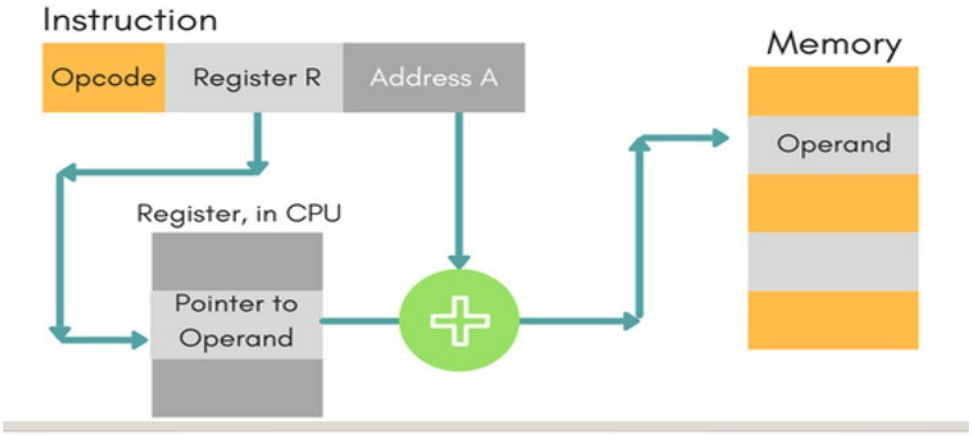
Operation: AX ← M[BX]

Fetch data from memory address stored in BX.



6. *Displacement (Base + Offset) Addressing Mode*

- Effective Address = Constant Address + Contents of a Register
- Example: (Imaginary) MOV AX, 1000H[BX]
  - Operation: AX ← M[1000H + BX]



7. *Stack Addressing Mode*

- In stack addressing, the instructions automatically use the **top of the stack**. You don't need to give any memory address.
- When you **PUSH** something (store it):
  - The **stack pointer (SP)** moves down first (decrements).
  - Then the data is stored at that new top location.

- When you **POP** something (take it out):
  - The data is taken from the current top of the stack.
  - Then the **stack pointer moves up** (increments).

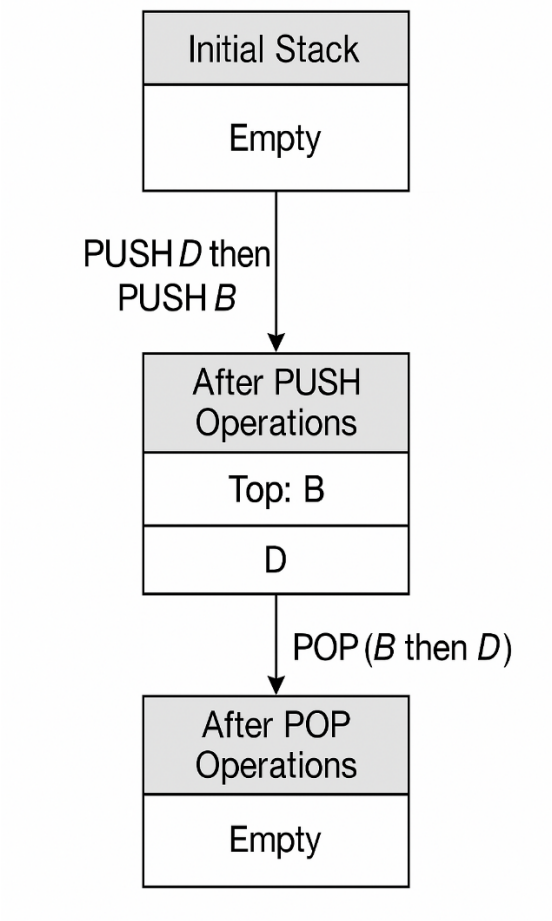
- The **SP always points to the top of the stack**.

Important Points

- PUSH and POP should happen in **reverse order**. For example:

```
PUSH B
PUSH D
...
...
POP D
POP B
```

- If you POP in the wrong order, the data will go to the wrong place (e.g., B and D values will get mixed up).


Initial Stack
Empty
PUSH D then
PUSH B
After PUSH
Operations
Top: B
D
POP (B then D)
After POP
Operations
Empty

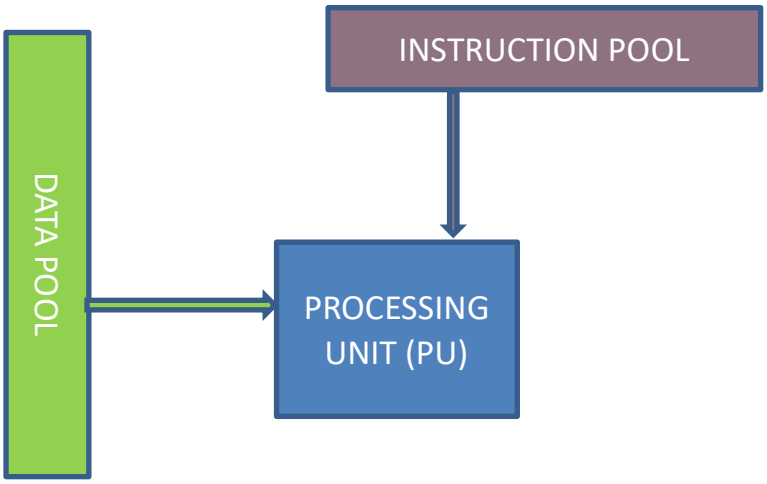
17. Classify the concept of parallel processing architecture and provide a brief overview of various parallel processing architectures with neat sketches.

- SISD: One control unit → one processor → one memory
- SIMD: One control unit → many processors → many data streams
- MISD: Multiple control units → different operations on same data
- MIMD: Multiple processors and control units handling different data and instructions

1. **SISD (Single Instruction, Single Data)**

Classic CPU model. One instruction at a time on one piece of data.

- A **single processor** executes **one instruction stream**.
- Operates on **one data stream** stored in memory.
- One **instruction pool** → One **processing unit (PU)** → One data stream.



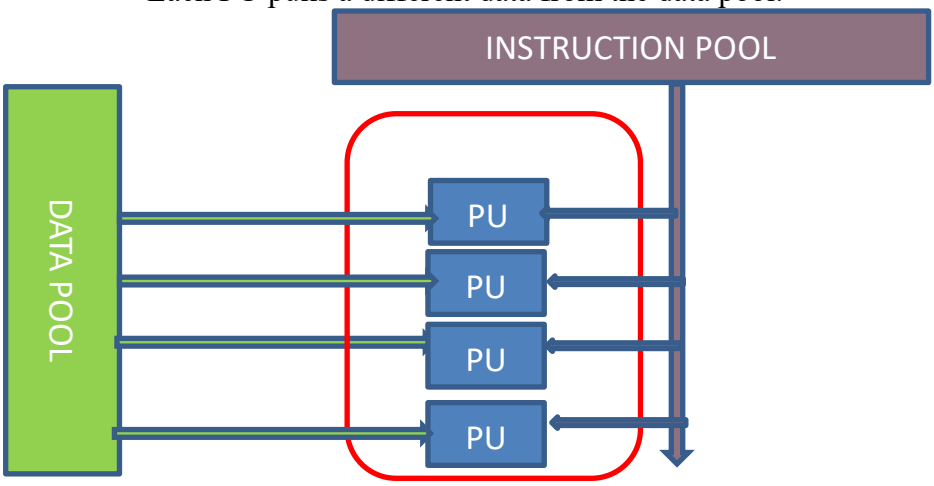
2. **SIMD (Single Instruction, Multiple Data)**

One instruction operates on multiple pieces of data simultaneously.

- A **single instruction** controls **multiple processors**.
- Each processor works on a **different data set**.
- Used in **vector processors and GPUs**.

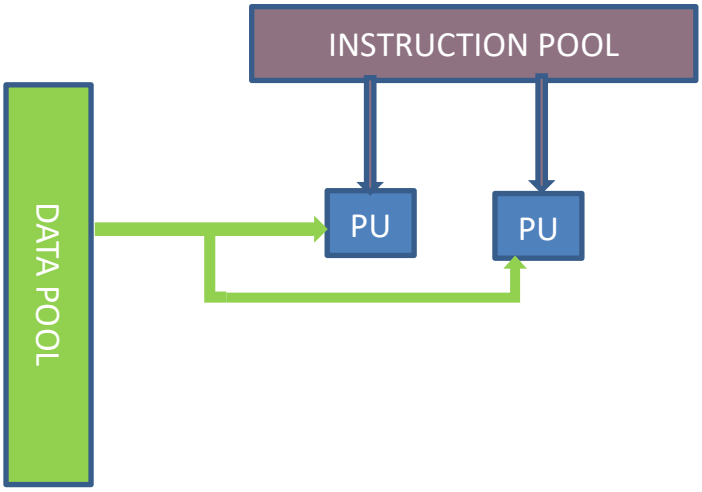
Diagram Summary:

- One **instruction pool** → Multiple **processing units (PU)**.
- Each PU pulls a different data from the data pool.



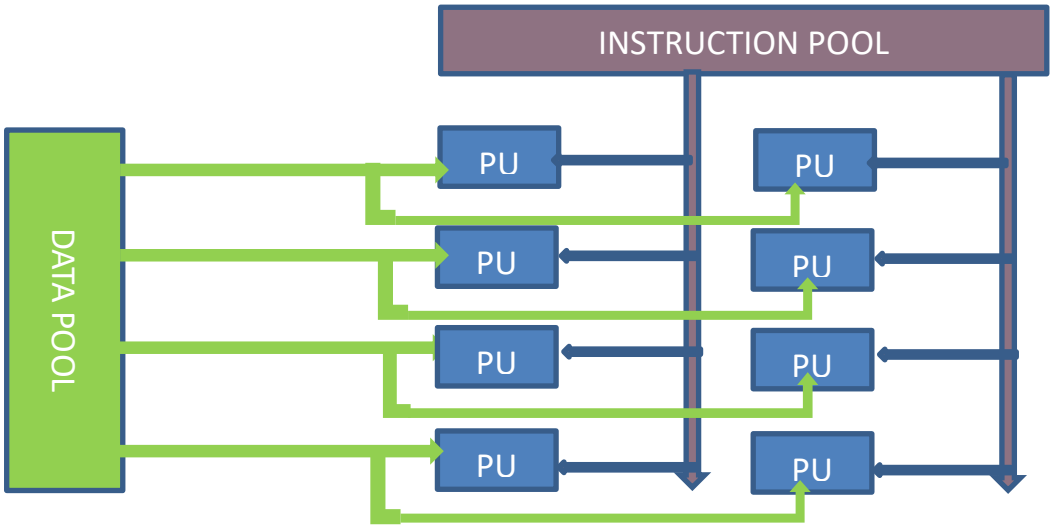
3. **MISD (Multiple Instruction, Single Data)**
   Multiple instructions are applied to the **same piece of data**.
   - Not practically implemented in real systems.
   - Conceptual model for redundancy and fault-tolerant systems.

   Example:
   Hypothetical — this architecture is mainly theoretical.
   Diagram Summary:
   - One data stream → Multiple **PUs**, each receiving **different instructions**.



4. **MIMD (Multiple Instruction, Multiple Data)**
   Multiple processors execute **different instructions** on **different data**.
   - Each processor can work independently.
   - Common in **multi-core systems**, clusters, and **SMP/NUMA architectures**.

   Example:
   Running multiple applications at the same time on different cores.
   Diagram Summary:
   - Multiple **instruction pools** → Multiple **PUs**, each with its own data stream.



18.Explain the implementation of a hardwired control unit along with its advantages and limitations.

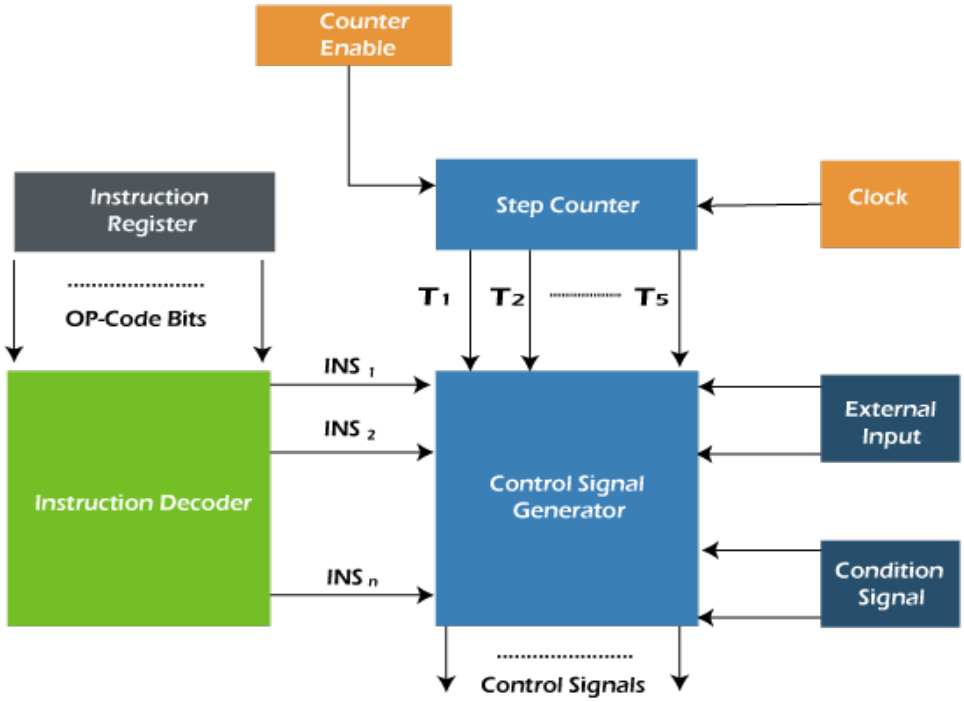
A **Hardwired Control Unit** is like a **pre-built circuit** inside the CPU. It uses **wires, logic gates, and small electronic parts** to send signals and tell the computer what to do step by step.

It controls things like:

- **Fetching** the instruction from memory
- **Decoding** what that instruction means
- **Executing** it (doing the task)
- **Storing** the result
- Has dedicated circuit for each instruction.

Everything is connected through wires — like a fixed pathway so it works **very fast**, but it **cannot be changed easily**.

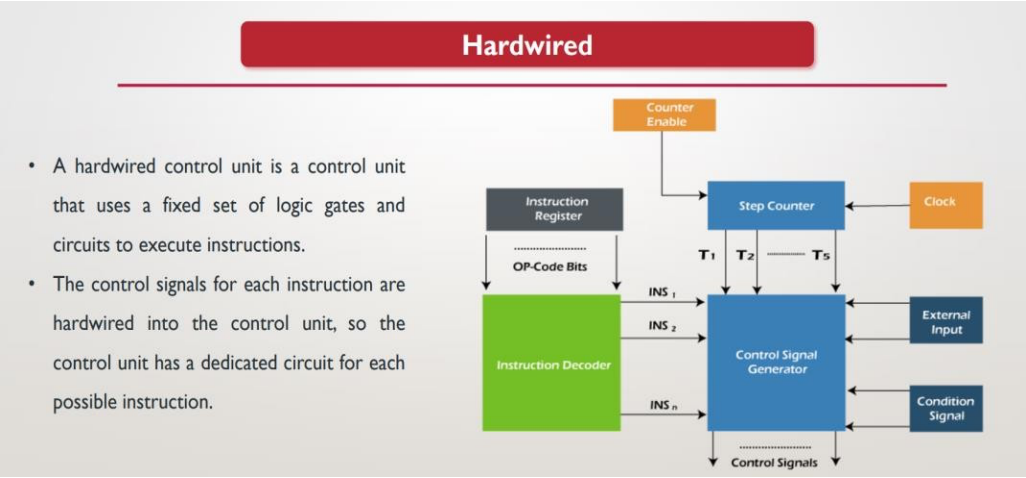
Advantages of Hardwired Control Unit



Advantage	Description
🔶 **Speed**	Much faster than microprogrammed control as it uses fixed logic circuits.
🔶 **Efficient Execution**	Well-suited for simple, fixed-instruction sets (e.g., RISC).
🔶 **Compact Hardware**	Requires fewer memory resources since no control memory is needed.

Limitations of Hardwired Control Unit

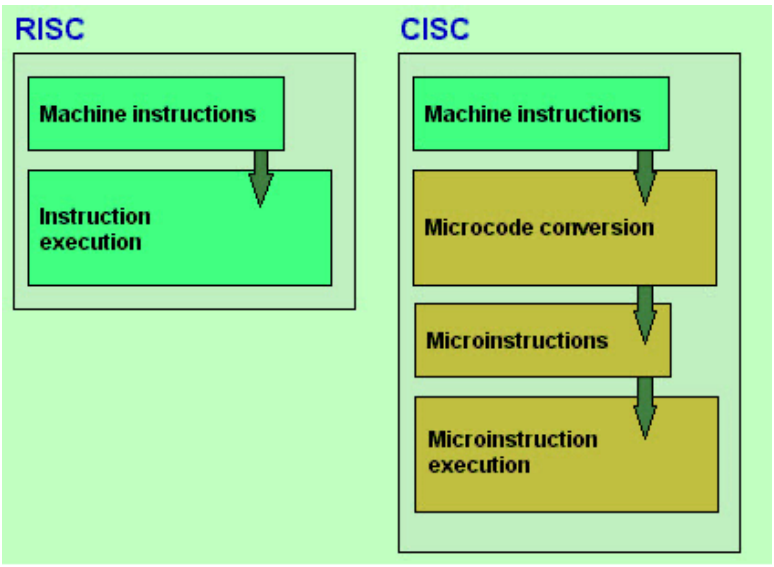
Limitation	Description
🔷 **Complex Design**	Difficult to design and debug for complex instruction sets.
🔷 **Lack of Flexibility**	Hard to modify or upgrade. Any change requires hardware redesign.
🔷 **Not Scalable**	Adding new instructions is challenging and may require complete restructuring.

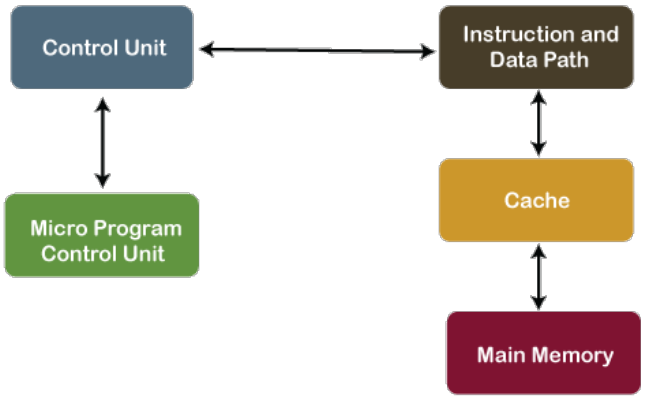


19. Compare and contrast the design philosophies of RISC and CISC processors by providing neat sketches of their architectures

RISC vs CISC – Design Comparison

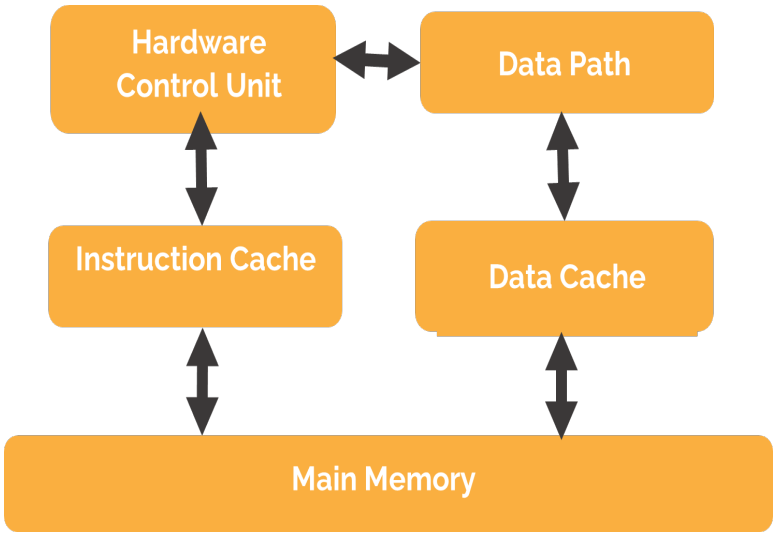
Feature	RISC (Reduced Instruction Set Computer)	CISC (Complex Instruction Set Computer)
Instruction Set	Few, simple instructions	Many complex instructions
Instruction Size	Fixed size	Variable size
Execution Time	One clock cycle per instruction	May take multiple cycles
Memory Usage	More memory for longer programs	Less memory (due to fewer instructions)
Hardware Logic	Simple and fast	More complex and slower
Control Unit	Hardwired	Microprogrammed





CISC Architecture

RISC Architecture:



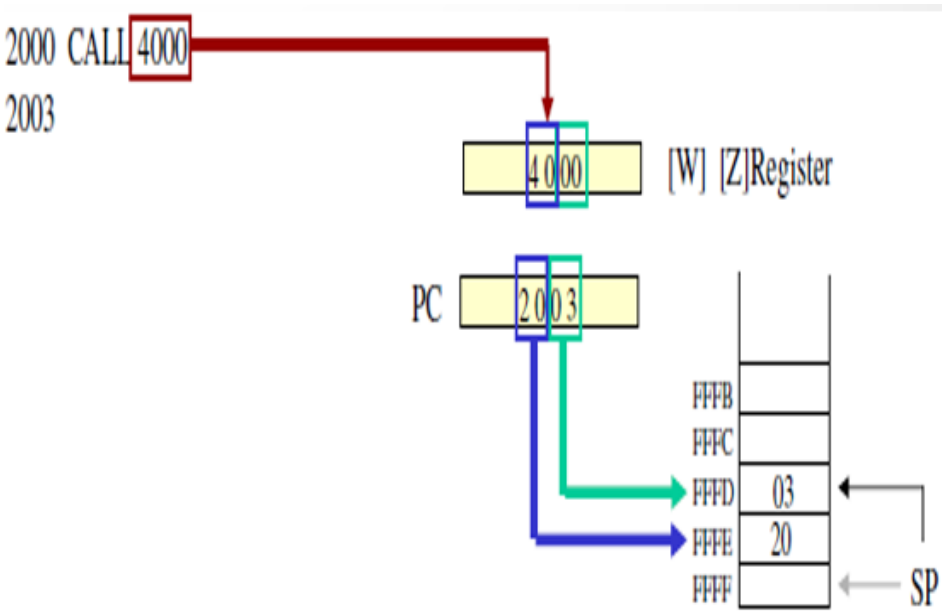
20.Explain the process of executing a CALL 4000H instruction from address 2000H, detailing the effects on the Program Counter and Stack Pointer when executing the RET instruction within the subroutine. Use a diagram to visually represent the process, considering variable stack top addresses.

Initial Conditions

- **Instruction:** CALL 4000H(at memory location 2000H)
- **Program Counter (PC):** 2000H → 2003H (next instruction after CALL)
- **Stack Pointer (SP):** Let's assume initial SP = FFFFH
  (The stack grows **downward** in memory)

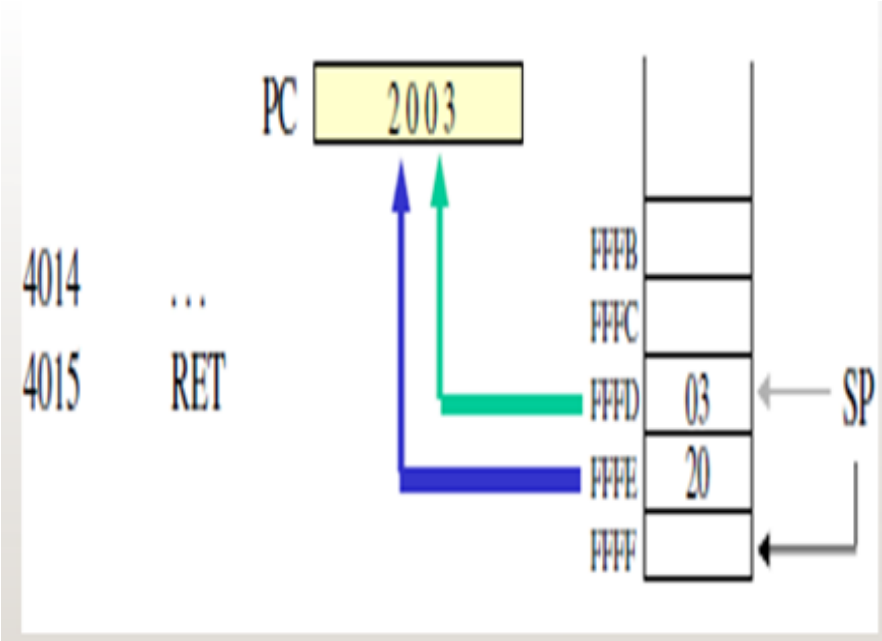
CALL 4000H – What Happens?

1. **PC = 2000H** → Executes CALL 4000H
2. The return address (address of next instruction = 2003H) is **pushed onto the stack**:
   - High byte 20H is pushed first → SP decremented to FFFEH
   - Low byte 03H is pushed next → SP becomes FFFDH
3. **SP after push:** FFFDH
4. **PC is now set to:** 4000H
   (Control transfers to subroutine at 4000H)



Stack after CALL:

Address	Data
FFFEH	20H (high byte of return address)
FFFDH	03H (low byte of return address)



RET Instruction – What Happens?

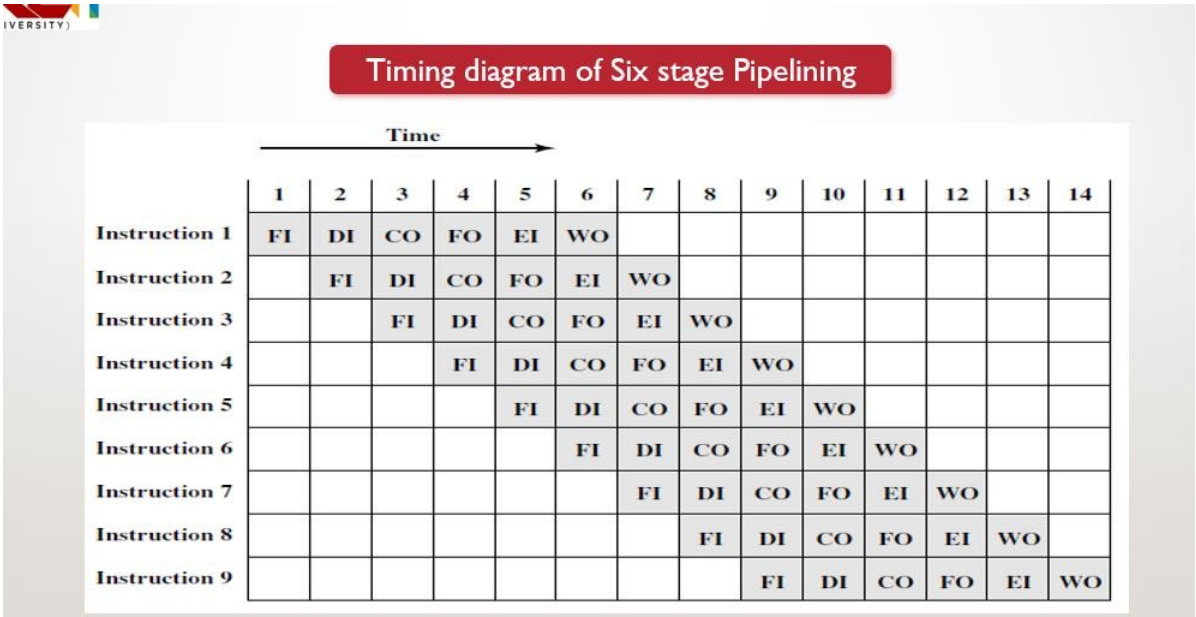
1. Inside the subroutine (at 4000H), when RETis executed:
   - It **pops the return address from the stack**
   - First, low byte 03His popped from FFFDH→ SP = FFFEH
   - Then, high byte 20His popped from FFFEH→ SP = FFFFH
2. **PC is now restored to 2003H**, continuing normal program execution after the CALL.

Stack after RET:

Address	Data
FFFFH	(SP reset here – stack is now empty)

21.Illustrate the distinct phases involved in a six-stage pipelining process alongside a clear timing diagram.

Stage	Description
FI	Fetching the instruction.
DI	Decoding the instruction.
CO	Calculation of effective address of operands.
FO	Fetching the operands.
EI	Executing the instruction.
WO	Writing the results or operands.



22.Identify and discuss the different hazards encountered in pipelining and propose strategies for their elimination.

A **hazard** is a condition that **prevents the next instruction in the pipeline from executing** in its designated clock cycle. Hazards cause **pipeline stalls (bubbles)** and reduce performance. A **hazard** is a condition that **prevents the next instruction in the pipeline from executing** in its designated clock cycle. Hazards cause **pipeline stalls (bubbles)** and reduce performance.

1. Data Hazards

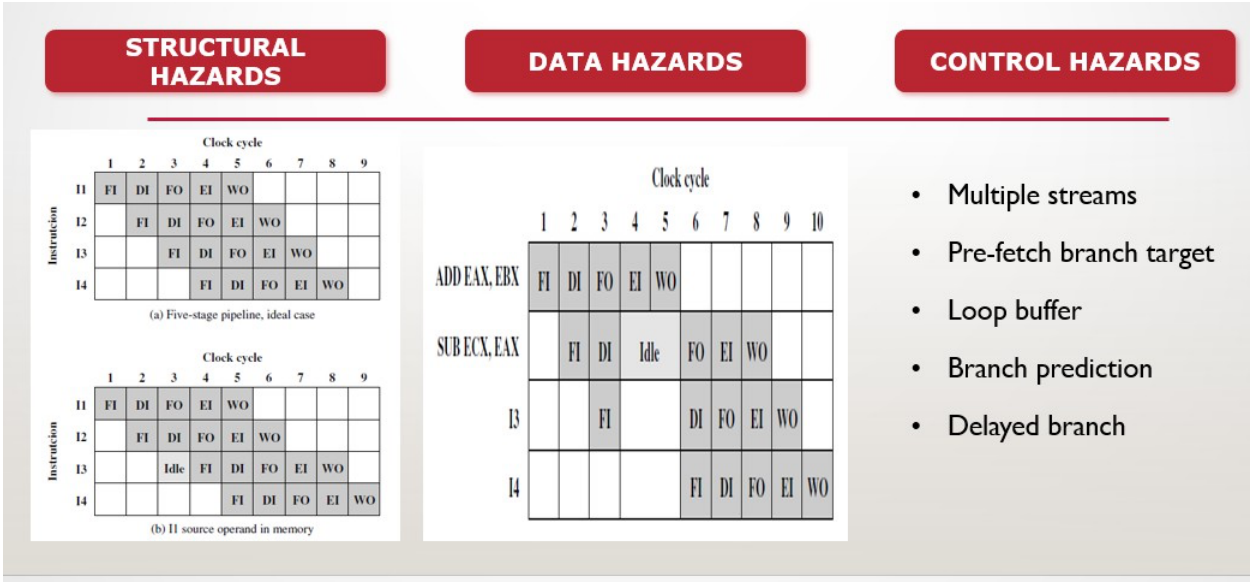
- **Cause:** When an instruction depends on the result of a **previous instruction** that hasn't completed yet.
- Forwarding, Stall.

2. Control Hazards (Branch Hazards)

- **Cause:** Occur with **branch or jump instructions** whose target is not known early in the pipeline.
- Delay Slot, Speculation.

3. Structural Hazards

- **Cause:** When **hardware resources** are not available for simultaneous instruction stages.
- **Example:** Both IF and MEM need access to memory at the same time.
- Separate units, Pipeline scheduling.



23.Develop and analyse an assembly-level program to implement the computation of X = (A+B)*(C+D) while incorporating stack usage.

X = (A + B) * (C + D)		
Assembly Language Instruction	Stack Transfer Operation	Explanation
PUSH A	TOS ← A	Push A To The Top Of Stack (TOS).
PUSH B	TOS ← B	Push B To The Top Of Stack (TOS).
ADD	TOS ← (A + B)	Add (A + B) And Push To The Top Of Stack (TOS).
PUSH C	TOS ← C	Push C To The Top Of Stack (TOS).
PUSH D	TOS ← D	Push D To The Top Of Stack (TOS).
ADD	TOS ← (C + D)	Add (C + D) And Push To The Top Of Stack (TOS).
MUL	TOS ← (C + D) * (A + B)	Multiply (C + D) And (A + B) And Push To The Top Of Stack (TOS).
POP X	M [X] ← TOS	Store the contents of the Top Of Stack TOS ((C + D) * (A + B)) Into memory address X

24.Utilize the diagram to illustrate and elucidate the functions of the following CPU components in the fetch phase: i) Instruction Register ii) Memory Address Register iii) Memory Buffer Register iv) Program Counter and v) Control Unit.

1. Program Counter (PC)
   - Holds the address of the next instruction to be fetched.
   - Sends this address to the Memory Address Register (MAR).
   - After sending, it increments to point to the next instruction in sequence.
2. Memory Address Register (MAR)
   - Receives the address from the Program Counter.
   - Sends this address to memory to retrieve the instruction stored at that location.
3. Memory Buffer Register (MBR) (Also called Memory Data Register - MDR)
   - Temporarily stores the instruction fetched from memory.
   - Acts as a bridge between main memory and the CPU.
   - Passes the fetched instruction to the Instruction Register.
4. Instruction Register (IR)
   - Receives the instruction from the MBR.
   - Holds the current instruction so the Control Unit can decode and execute it.
5. Control Unit (CU)
   - Decodes the instruction stored in the IR.
   - Sends signals to other CPU parts (ALU, registers, buses) to perform the required operation.
   - Controls the flow of data between CPU and memory during this phase.

Summary Table

Component	Role in Fetch Phase
Program Counter (PC)	Points to next instruction; updates after fetch
Memory Address Register (MAR)	Holds address from PC; sends it to memory
Memory Buffer Register (MBR)	Temporarily stores instruction read from memory
Instruction Register (IR)	Holds instruction to be decoded
Control Unit (CU)	Decodes instruction and orchestrates CPU operations

25.Utilize sample instructions to demonstrate operand access with effective address calculations in both direct and register indirect addressing modes, incorporating clear diagrams for illustration.

1. Direct Addressing Mode

- The instruction directly specifies the **memory address** of the operand.
- **Effective Address (EA) = A** (A = address provided in instruction).

MOV AX, [1592H]

- This means: Move the data from memory location 1592H into register AX.

1. The instruction contains the memory address 1592H.
2. The CPU fetches the operand **directly** from memory at 1592H.
3. The value is transferred to the AX register.

Instruction → [Opcode | Address: 1592H]
↓
Memory[1592H]
↓
Operand → AX

---

2. Indirect Addressing Mode

- The instruction contains a **memory location**, which **holds another memory address** where the operand is actually stored.
- **Effective Address (EA) = (A)**

Example Instruction:

MOV AX, @2005H

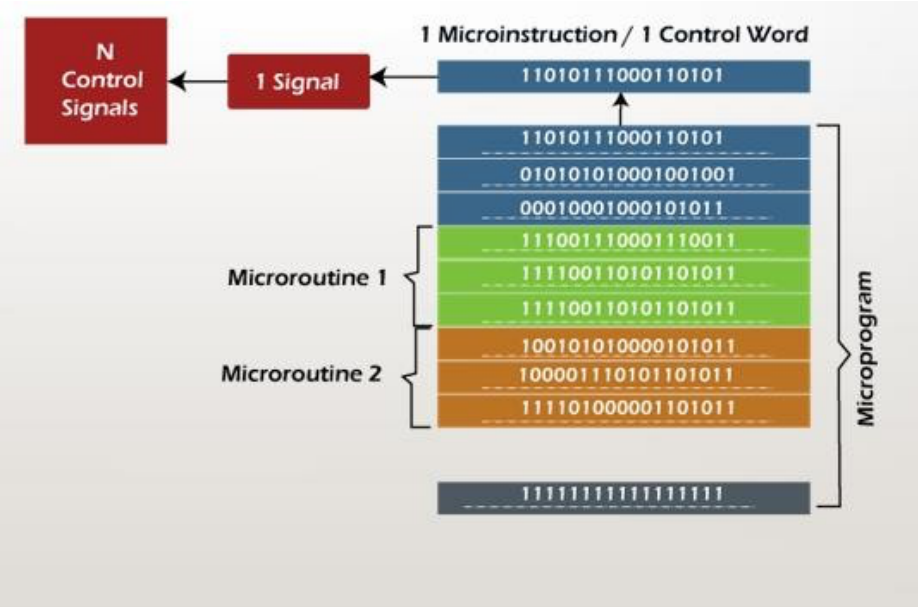
- 2005H contains a value like 1592H, and at that location (1592H) lies the actual operand.

---

1. The instruction points to memory location 2005H.
2. Memory[2005H] = 1592H (the effective address).
3. Memory[1592H] contains the operand.

Instruction → [Opcode | Address: 2005H]
↓
Memory[2005H] → 1592H
↓
Memory[1592H] → Operand → AX

26.Construct a detailed diagram showcasing the implementation of a microprogrammed control unit, and analyse its advantages and disadvantages.



Advantages	*Disadvantages*
Easy to design and modify	Slower than hardwired control
Flexible (firmware can be updated)	Requires more memory
Easier to debug and maintain	Not ideal for real-time systems
Supports complex instruction sets	Limited performance optimization
Consistent and reliable control signals	

27.Utilize the values of AL (10110001) and BL (11010010) to develop a sequence of instructions performing the following tasks and displaying the results: i) Perform a logical AND operation between the contents of AL & BL. ii) Rotate the content of AL rightwards twice. iii) Shift the content of AL leftwards once. iv) Perform a logical OR operation between the contents of AL & BL. v) Perform a logical XOR operation between the contents of AL & BL. vi) Move the content of AL into BL.

Bitwise Operations on AL and BL Registers

Given:
AL = 10110001
BL = 11010010

i: Logical AND Operation (AND AL, BL)

AL = 1 0 1 1 0 0 0 1
BL = 1 1 0 1 0 0 1 0

AND = 1 0 0 1 0 0 0 0

ii: Rotate AL Right Twice (ROR AL, 2)

Start with: AL = 10010000
1st Rotate → AL =
01001000
2nd Rotate → AL =
00100100

iii: Shift AL Left Once (SHL AL, 1)

Before: AL = 00100100
Shifted: 01001000 → 48H

iv: Logical OR Operation (OR AL, BL)

AL = 0 1 0 0 1 0 0 0
BL = 1 1 0 1 0 0 1 0

OR = 1 1 0 1 1 0 1 0

Step v: Logical XOR Operation (XOR AL, BL)

AL = 1 1 0 1 1 0 1 0
BL = 1 1 0 1 0 0 1 0

XOR = 0 0 0 0 1 0 0 0

Step vi: Move AL into BL (MOV BL, AL)
AL = 00001000
→ Copy value into
BL BL = 00001000

CO3 Question Bank-Answers