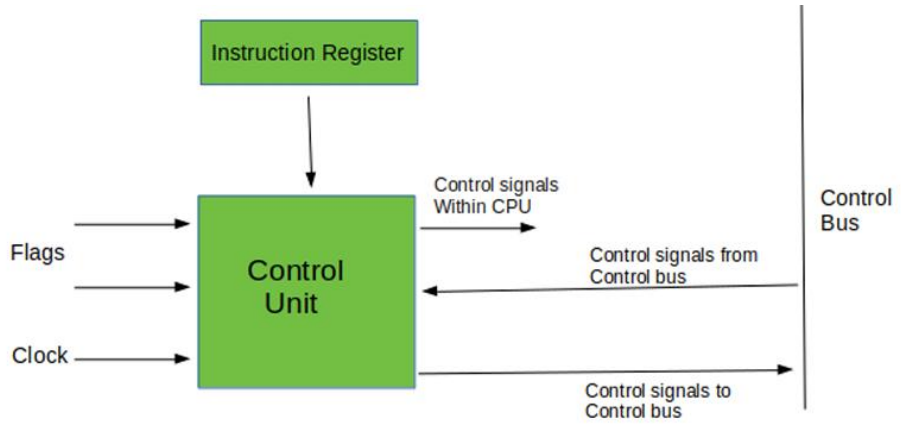
COMPUTER ORGANIZATION AND ARCHITECTURE

Control Unit

Control Unit (CU) in CPU

The **Control Unit (CU)** is a key component of the **CPU (Central Processing Unit)** that directs the operation of the processor. It tells the computer's memory, ALU, and input/output devices how to respond to the instructions of a program.



Block Diagram of the Control Unit

Functions of the Control Unit:

1. **Coordination of Data Movement:** The CU coordinates the transfer of data between the CPU's sub-units, memory, and input/output devices.
2. **Instruction Interpretation:** It fetches the program instructions from memory and decodes them. After decoding, the CU generates the necessary control signals for execution.
3. **Control of Data Flow:** The CU ensures that data moves properly within the processor. It controls data transfer between registers, the Arithmetic Logic Unit (ALU), and other components.
4. **Conversion of Instructions:** It converts external instructions or commands into sequences of control signals that allow the CPU to respond appropriately.
5. **Management of Execution Units:** The CU supervises various execution units like the ALU, data buffers, and registers within the CPU. It ensures each component works together as per the needs of the instruction.
6. **Task Handling:** The CU handles multiple tasks, including **fetching**, **decoding**, **executing**, and **storing results** after execution. This ensures the proper sequence of operations for smooth functioning.

How the Control Unit Works:

1. **Fetching Instructions:** The CU retrieves instructions from the system's main memory.

2. **Decoding Instructions:** It decodes the instruction, understanding what operation needs to be performed.
3. **Generating Control Signals:** Based on the decoded instruction, the CU generates control signals, which dictate the operations to be performed by other components (e.g., ALU, registers).
4. **Execution and Data Movement:** The CU directs the execution units, such as the ALU, and manages the movement of data between different parts of the CPU.
5. **Storing Results:** After execution, it handles storing the results either back in memory or in the appropriate registers.

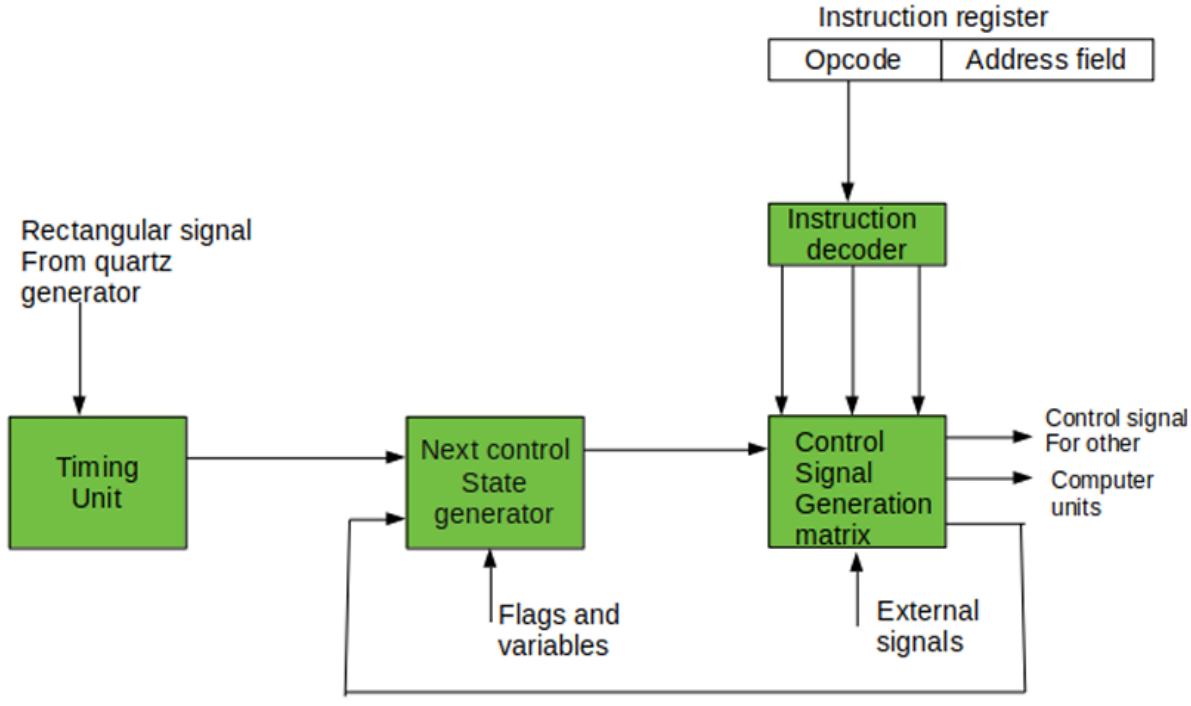
Types of Control Units

There are two primary types of control units in a CPU:

1. **Hardwired Control Unit**
2. **Microprogrammed Control Unit**

1. Hardwired Control Unit:

A **Hardwired Control Unit (HCU)** generates control signals for instruction execution using **dedicated hardware circuits**. These circuits are specifically designed for a particular task and cannot be modified without physically changing the structure of the circuit.



Block diagram of a hardwired control unit of a computer

How It Works:

- The **operation code (opcode)** of an instruction is the primary input for control signal generation in a hardwired control unit.
- The **instruction decoder** decodes the opcode and outputs active signals corresponding to different fields of the instruction.
- These output lines are connected to a **control signal matrix**, which generates the control signals necessary for various execution units in the CPU.
- The control signals are generated based on a combination of the decoded opcode and external signals (e.g., interrupts). The matrix functions similarly to **Programmable Logic Arrays (PLA)**.
- The control signals are sent through the instruction execution cycle. The control unit is designed to generate signals not only for a specific point but also for the entire execution time of the instruction.
- The **timing unit** supplies rectangular timing signals to synchronize the control unit's internal states.

Working of Hardwired Control Unit (Short Version):

1. **Instruction Fetch**: Instruction is loaded into the **Instruction Register (IR)** from memory.
2. **Decoding**: The **opcode** is decoded using **combinational logic** (like decoders).
3. **Control Signal Generation**: Logic circuits generate specific **control signals** based on:
    - Decoded instruction
    - Clock/timing signals
    - Status flags
    - External inputs (e.g., interrupts)
4. **Execution**: These signals control CPU components like **ALU**, **registers**, and **memory** to perform operations.
5. **Next Cycle**: Updates the **Program Counter (PC)** and starts the **next instruction fetch**.

Execution Cycle:

1. **Instruction Fetching:** When a new instruction arrives, the control unit is in the initial state of fetching the instruction.

2. **Instruction Decoding:** The opcode is decoded, and the control unit enters the first state related to executing the instruction.
3. **Control Signal Generation:** The control unit's state changes based on flags, state variables, and input signals like interrupts.
4. **Instruction Execution:** The cycle continues with appropriate control signals being generated to manage the execution stages.
5. **Interrupt Handling:** If an interrupt occurs, the control unit transitions to a state dedicated to handling the interrupt.
6. **Fetching the Next Instruction:** When the current instruction completes, the control unit fetches the next instruction by updating the program counter and loading the instruction into the instruction register.
7. **Program Termination:** If the instruction is a stop command, the control unit enters the operating system state, waiting for further directives.

Key Characteristics:

- The control unit is built using **fixed hardware logic circuits**.
- **Fast execution** as the control signals are hardcoded for specific tasks.
- **Limited flexibility** because the control structure cannot be easily modified without physical changes.

Advantages of Hardwired Control Unit:

- **Speed:** As the control logic is fixed, it can execute instructions faster.
- **Simplicity:** It is easier to design for simple processors with a smaller instruction set.
- **Efficiency:** Since the signals are generated directly by hardware, there is less overhead.

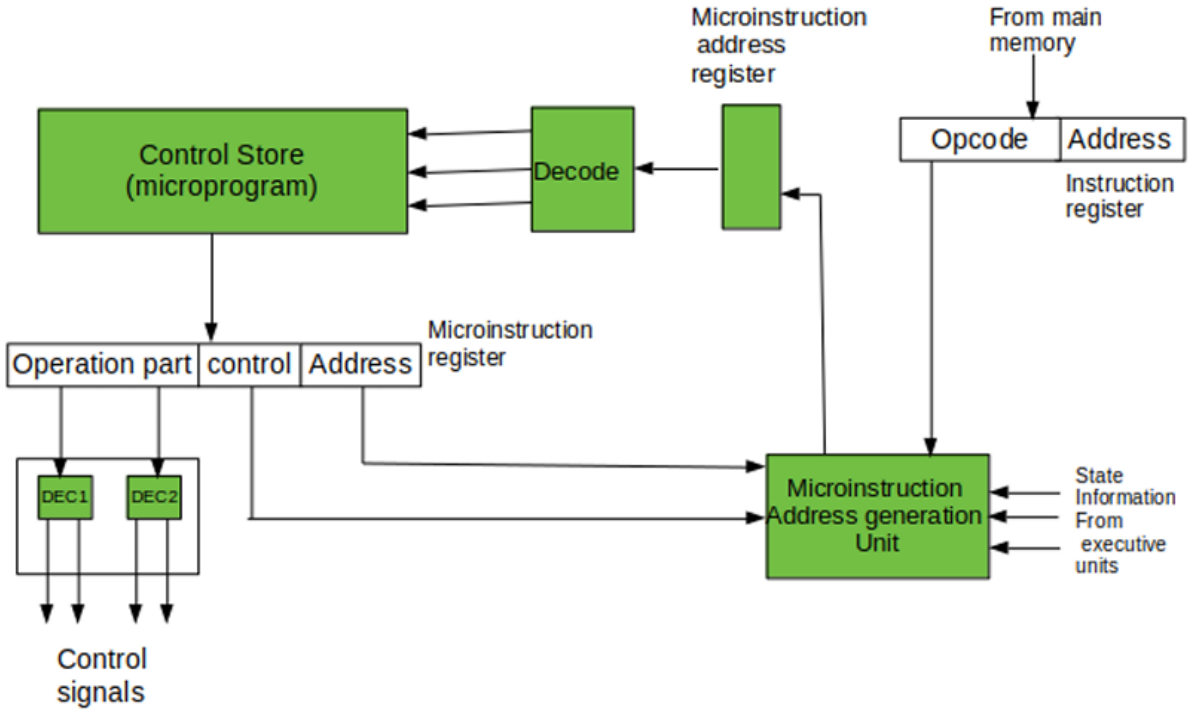
Disadvantages of Hardwired Control Unit:

- **Less Flexibility:** Any change in the control logic requires physical modifications.
- **Complex for Complex Instructions:** It is difficult to design for complex or evolving instruction sets.

Microprogrammed Control Unit

A **microprogrammed control unit** generates control signals using **microinstructions** stored in a special memory called the **control store**. These microinstructions define the sequence of **micro-operations** required to execute each machine instruction.

The **key difference** from a **hardwired control unit** is that control logic is implemented **in software (microprograms)** rather than fixed hardware, making it **easier to modify and more flexible**.



Microprogrammed control unit with a single level control store

How It Works:

1. **Instruction Fetch**: The CPU fetches an instruction from memory and loads it into the instruction register.
2. **Microinstruction Address Generation**: The opcode from the instruction is used to determine the starting address of the corresponding microprogram in the control memory.
3. **Fetch Microinstruction**: The control memory (microprogram) provides the microinstruction at that address, which is loaded into the microinstruction register.
4. **Control Signal Generation**: The microinstruction is decoded to generate the necessary control signals for executing the instruction step.
5. **Sequencing**: The address field in the microinstruction (along with status signals) determines the address of the next microinstruction, repeating the process until the instruction is fully executed.

Types of Microprogrammed Control Units

- **Single-Level**: Uses **one control memory** where each microinstruction directly generates control signals through decoding. This can lead to larger memory requirements due to repeated storage of similar control signals.
- **Two-Level**: Uses two memories-**microinstruction memory** and **nano-instruction memory**. Microinstructions point to nano-instructions, which actually generate the control signals. This approach reduces redundancy and memory size, as common control signal patterns are stored only once.

1. **Single-Level Control Store:**
    - The instruction opcode is sent to the control store address register.
    - This address fetches the first microinstruction for that instruction into the microinstruction register.
    - Each microinstruction contains:
        - Encoded control signals for hardware actions.
        - Addressing mode control (may depend on processor status flags).
        - The address of the next microinstruction.
    - Microinstructions are executed in sequence until the last one, which fetches the next instruction from memory.

    Advantages:

        - Simple design.
        - Each microinstruction points to the next, enabling smooth execution.

    Disadvantages:

    Requires **more control memory**, since each microinstruction stores both control signals and the next address.

2. **Two-Level Control Store:**
    - A **two-level control store** introduces a **nano-instruction memory** in addition to the regular microinstruction memory.
    - In this setup, **microinstructions** do not contain control signals directly. Instead, the microinstructions point to **nano-instructions** in the **nano-instruction memory**.

- The **nano-instruction memory** holds all possible combinations of control signals, essentially encoding the control signals in a compressed format.
- The **microinstruction** contains the address for the **nano-instruction** that encodes the actual control signals.

Advantages of Two-Level Control Store:

- **Smaller microinstruction size**: Since the microinstructions do not directly contain control signals, they are shorter in length.
- **Smaller control memory**: The nano-instruction memory is smaller as it stores only the control signals, avoiding redundancy.
- **Efficient use of memory**: Redundant microinstruction parts are avoided, as the microprogram uses nano-instructions to generate control signals.

Disadvantages of Two-Level Control Store:

- More **complex structure** with additional layers of control memory.
- **Requires additional processing** to fetch and decode nano-instructions.

Key Components of Microprogrammed Control Unit:

1. **Control Store:** The memory that holds the **microprogram** (a sequence of microinstructions) for interpreting and executing instructions.
2. **Microinstruction Register:** Stores the fetched microinstruction from the control store.
3. **Microprogram Counter:** Holds the address of the next microinstruction to be fetched.
4. **Nano-instruction Memory:** In the two-level control store, this stores encoded control signals (nano-instructions) that are fetched by microinstructions.
5. **Decoders:** Used to decode the microinstruction and map it to the appropriate control signals.
6. **Control Signals:** These signals drive the various components of the CPU to perform operations like reading from memory, writing to registers, or executing arithmetic/logical operations.

Advantages of Microprogrammed Control Unit:

- **Flexibility:** The microprogram can be modified or updated by changing the microinstructions in the control store.
- **Easier to Design:** It's easier to design a control unit using microprogramming as compared to hardwiring the control signals, especially for complex instruction sets.

- **Versatility:** New instructions can be added by simply adding new microprograms to the control store without needing hardware changes.

Disadvantages of Microprogrammed Control Unit:

- **Slower Execution:** Fetching and decoding microinstructions can introduce delays compared to the fast execution of hardwired control units.
- **Complexity in Microprogramming:** Writing and managing microprograms can be complex, especially for very large instruction sets.

Advantages of a Well-Designed Control Unit:

1. **Faster Execution**: Executes instructions quickly by optimizing the pipeline and reducing clock cycles.
2. **Better Performance**: Enhances CPU speed, reduces delays, and boosts throughput.
3. **Handles Complex Tasks**: Supports complex instructions, reducing overall instruction count.
4. **More Reliable**: Detects and corrects errors like memory faults and pipeline stalls.
5. **Energy Efficient**: Saves power by using resources efficiently.
6. **Accurate Branch Prediction**: Reduces mispredictions, improving execution flow.
7. **Scalable**: Can handle growing workloads and complex applications.
8. **Supports Parallelism**: Runs multiple instructions at once for faster performance.
9. **Improved Security**: Adds features like ASLR and DEP to protect against attacks.
10. **Cost-Effective**: Reduces hardware needs, lowering production costs.

❌ **Disadvantages of a Poorly-Designed Control Unit**

1. **Slower Performance:**
   Causes pipeline stalls and delays, reducing CPU speed.
2. **More Complex:**
   Hard to design, test, and maintain due to added complications.
3. **Wastes Power:**
   Uses more energy by being inefficient with resources and timing.
4. **Less Reliable:**
   Prone to errors and system faults.
5. **Limited Instruction Support:**
   Can't handle complex instructions, reducing CPU functionality.
6. **Poor Resource Management:**
   Wastes memory and registers, slowing down performance.

7. **Not Scalable:**
   Struggles with future demands and larger workloads.
8. **Weak Parallelism:**
   Can't run multiple instructions well, lowering efficiency.
9. **Security Risks:**
   Opens the door to cyber threats like overflows and injection attacks.
10. **Higher Costs:**
    Needs more parts and complex manufacturing, increasing expenses.

Importance of Control Unit (CU)

The **Control Unit (CU)** is one of the most critical components of the **Central Processing Unit (CPU)**. It acts as the **"brain of the CPU"** that **directs the operation** of the processor by coordinating how data moves and how instructions are executed.

1. Directs All Processor Operations

The CU controls the **flow of data** between the **ALU (Arithmetic Logic Unit)**, **memory**, **registers**, and **I/O devices**, ensuring everything operates in sync.

2. Instruction Decoding

It **decodes the fetched instruction** from memory and determines what actions to perform—such as arithmetic operations, memory access, or logical decisions.

3. Timing and Control Signals

The CU **generates control signals** to ensure every component acts at the **right time** in the **correct order**, coordinating internal operations of the CPU.

4. Enforces Instruction Cycle

The CU **manages the instruction cycle**: Fetch → Decode → Execute → Store. This is the basic functioning loop of any processor.

5. Manages Execution of Instructions

It ensures that each instruction is executed **correctly and completely**, handling both **simple and complex** instruction types.

⚙️ Working of Control Unit

The working of the CU follows a sequence known as the **Instruction Cycle**, consisting of the following steps:

🌀 1. Fetch

- The **Program Counter (PC)** provides the address of the next instruction.
- The CU sends a signal to **fetch** the instruction from memory.
- The instruction is stored in the **Instruction Register (IR)**.

🧠 2. Decode

- The CU **interprets (decodes)** the instruction in the IR.
- It determines what operation is to be performed and what components (like ALU or memory) are needed.

🚦 3. Generate Control Signals

- Based on the decoded instruction, the CU **generates control signals** that activate the appropriate parts of the CPU (e.g., enabling ALU, reading from/writing to memory).

⚡ 4. Execute

- The CU instructs the relevant components to **perform the operation** (e.g., add, move data, jump).
- If arithmetic is needed, CU activates ALU.
- If data needs to be moved, CU controls data paths.

🔁 5. Store & Repeat

- The result is stored in the appropriate location (register or memory).
- The CU **updates the PC** to point to the next instruction and **repeats** the cycle.

Aspect	Hardwired Control Unit	Microprogrammed Control Unit
Definition	Uses fixed logic circuits to generate control signals.	Uses a control memory to store microinstructions that generate control signals.
Speed	Faster due to direct hardware implementation.	Slower due to memory access for microinstructions.
Flexibility	Inflexible – changes require rewiring or redesigning hardware.	Flexible – can be easily modified by changing microcode.
Complexity	More complex to design for complex instruction sets.	Easier to design and maintain for complex instruction sets.

Aspect	Hardwired Control Unit	Microprogrammed Control Unit
Cost	Generally higher for complex CPUs due to hardware complexity.	Lower cost for complex CPUs due to ease of implementation.
Control Signal Generation	Generated using combinational logic circuits.	Generated from microinstructions stored in control memory.
Used In	Mostly used in **RISC** (Reduced Instruction Set Computing) processors.	Mostly used in **CISC** (Complex Instruction Set Computing) processors.
Modifiability	Not easily modifiable – physical changes needed.	Easily modifiable – just update the microcode.
Execution	Direct execution of control signals.	Control signals executed through a sequence of microinstructions.
Example	Intel 8085, simple RISC processors.	Intel 8086/8088, IBM System/360.

📘 Microinstructions?

Microinstructions are the **individual instructions** stored in a **control memory** of a **microprogrammed control unit**. Each microinstruction generates a set of **control signals** that direct the internal components of the CPU (like ALU, registers, memory, buses) to perform specific low-level operations.

They form part of a **microprogram**, which defines how a machine-level instruction (like ADD, LOAD, etc.) is executed at the hardware level.

🛠️ Structure of a Microinstruction

A typical microinstruction contains the following fields:

1. **Control Signals** – Encoded or direct bits that activate various hardware components.
2. **Next Address Field** – Specifies where to fetch the next microinstruction.
3. **Condition Field** – For conditional branching (e.g., if zero flag = 1, go to X).
4. **Micro-Operations Field** – Specific operations like Load IR, ALU Add, Increment PC.

🔁 Purpose and Use

- **Control execution flow** within the CPU.
- **Simplify complex instructions** into simpler micro-steps.
- Enable **modular control logic**, useful in CISC processors.
- Allow **easy modification or patching** of instruction behavior (by changing microcode).

📜 **Example**

A single microinstruction might specify:

- Load the instruction register (IR) from memory
- Increment the program counter (PC)
- Set the ALU to perform addition

This would be part of the "fetch" microprogram.

🔁 Microinstruction Sequencing

Microinstruction sequencing refers to the method used to determine the **order of execution** of microinstructions in a **microprogrammed control unit**.

Each instruction in a machine-level program is interpreted as a **microprogram** — a sequence of **microinstructions**. These microinstructions direct the internal operations of the CPU, like data movement, ALU operations, or register control.

Types of Microinstruction Sequencing:

1. **Sequential sequencing** – One microinstruction directly leads to the next.
2. **Branching** – Allows conditional or unconditional jumps to another microinstruction based on flags (like zero or overflow).
3. **Looping** – Repeats a set of microinstructions (useful in instruction decoding or ALU operations).

⭐ Importance of Microinstruction Sequencing

- ✅ Ensures the correct execution of instructions at the micro-level.
- ✅ Helps implement **conditional operations** like branching and loops.
- ✅ Allows **reuse** of microinstruction routines (e.g., fetch, decode, execute).
- ✅ Provides **flexibility** in control logic design, especially in CISC systems.

- ✅ Makes it easier to implement and manage **complex instruction sets** using a structured approach.

📏 Horizontal vs. Vertical Microinstructions

Feature	Horizontal Microinstruction	Vertical Microinstruction
Control Word Length	Long (more bits)	Short (fewer bits)
Format	Each bit directly controls one control signal	Encoded fields control groups of signals
Decoding Requirement	No decoding required (1-bit/1-signal)	Requires decoding to generate control signals
Parallelism	High – many control signals can be activated simultaneously	Lower – fewer control signals activated at once
Flexibility	More flexible – each signal can be independently controlled	Less flexible due to encoding constraints
Memory Usage	Consumes more memory (more bits per instruction)	Uses less memory
Speed	Faster due to direct signal control	Slower due to decoding overhead
Used In	High-performance CPUs (e.g., control of pipeline stages, ALUs)	Simpler or compact control units (e.g., embedded systems)

🔹 Single Address Field Microinstruction

A microinstruction format that includes only one address field, which specifies the address of the next microinstruction to be executed. It is typically used in sequential and simple conditional branching.

- **Contains** only **one address field**.
- Specifies the address of the **next microinstruction**.
- The sequencing logic is **simpler**.
- Common in **sequential** or **conditional branching**.

Example: $[Operation \mid Address \mid Control\ Bits]$

🔹 Dual Address Field Microinstruction

A microinstruction format that includes two address fields: one for the next microinstruction if a specified condition is true, and another if the condition is false. This allows direct implementation of conditional branching without extra logic.

- **Contains two address fields**.
- One for the **next microinstruction** if a condition is **true**, another if **false**.
- Used for **conditional branching** without external logic.
- Adds flexibility but **increases microinstruction size**.

Example: $[Operation \mid True\ Address \mid False\ Address \mid Control\ Bits]$

🔹 Variable Format Microinstruction

A type of microinstruction design in which different microinstructions may have different formats and lengths, depending on their function. This allows efficient use of control memory by using only the necessary fields for each instruction type.

- Microinstructions are of **different lengths** or **formats**.
- Each format has fields tailored for **specific purposes** (e.g., branching, ALU ops).
- Reduces memory usage by **avoiding unused fields**.
- Requires a **format field** to interpret each instruction properly.

Example Formats:

1. Control operations
2. Branching instructions
3. ALU operations
   Each has its own format.
