



 **Sequencing Logic**: Determines the order of operations.

 **Control Unit Registers and Decoders**: Store control information and decode instructions.

 **Control Memory(non-volitile)**: Stores microinstructions.

Control Unit

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.

Functions of this unit are −

* It is responsible for controlling the transfer of data and instructions among other units of a computer.
* It manages and coordinates all the units of the computer.
* It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
* It communicates with Input/Output devices for transfer of data or results from storage.
* It does not process or store data.

## ALU (Arithmetic Logic Unit)

## An ALU circuit has the integration of four key components inputs, operands, outputs, and storage.

## ALU

### Summary of ALU Operations

### Arithmetic Operations:

* **Addition**: Computes the sum of two numbers.
* **Subtraction**: Computes the difference between two numbers.
* **Multiplication**: Computes the product of two numbers.
* **Division**: Computes the quotient of two numbers.

### Bitwise Logic Operations:

* **AND**: Performs a bitwise AND operation between two binary numbers.
* **OR**: Performs a bitwise OR operation between two binary numbers.
* **XOR (Exclusive OR)**: Performs a bitwise XOR operation, returning true if only one of the bits is set.
* **NOT**: Performs a bitwise NOT operation, inverting all the bits of a binary number.

### Additional Specialized Operations:

* **Bitwise Shift Operations**:
  + **Left Shift** (<<): Shifts bits to the left, filling with zeros on the right.
  + **Right Shift** (>>): Shifts bits to the right, filling with the sign bit or zeros.
* **Comparison Operations**:
  + **Equal to** (==): Compares two numbers for equality.
  + **Greater than** (>): Checks if one number is greater than another.
  + **Less than** (<): Checks if one number is less than another.
  + **Greater than or equal to** (>=): Checks if one number is greater than or equal to another.
  + **Less than or equal to** (<=): Checks if one number is less than or equal to another.
  + **Not equal to** (!=): Checks if two numbers are not equal.

### Logical Operations:

* **Logical AND** (&&): Returns true if both operands are true.
* **Logical OR** (||): Returns true if at least one operand is true.
* **Logical NOT** (!): Returns true if the operand is false, and vice versa.

? MicroInstruction vs MicroOperation

**I, Micro-Operations** are the basic operations that occur within the processor. They are the smallest units of work that the CPU can perform. These operations are very low-level and involve fundamental tasks such as transferring data between registers, performing arithmetic or logic operations, and managing control signals.

**II, A microinstructions** are encoded representations of a micro-operation. They are stored in the control memory and specify the sequence of micro-operations that need to be performed to execute higher-level machine instructions (e.g., LOAD, STORE, ADD). Each microinstruction is part of a microprogram, which collectively defines the control logic for the processor.

**Components of a Microinstruction:**

* **Opcode:** Specifies the type of operation to be performed.
* **Source/Destination:** Identifies the registers or memory locations involved.
* **Control Signals:** Bit fields that directly control the hardware components to perform the specified micro-operations.

| Opcode | Source/Destination | Control Signals |

| 0100 | IR, MAR | MAR Load |

| 0101 | MEM[MAR], R1 | Memory Read, Register Write |

*In an actual implementation, the fields for source/destination and control signals in a microinstruction would be represented as binary codes rather than human-readable characters.*

-> The entire process of fetching, decoding, and executing an instruction is orchestrated by control signals.

III, The control unit can be divided into several key components:

* **Instruction Decoder**: decodes the machine code instructions stored in RAM to determine the operation type and operands.
* **Control Memory (Microprogram Memory)**: Stores the microinstructions.
* **Control Address Register (CAR)**: Holds the address of the current microinstruction to be executed.
* **Control Buffer Register (CBR)**: Holds the current microinstruction fetched from control memory.
* **Sequencer**: Determines the next address of the microinstruction to be executed, potentially based on conditions.

### **IV,** Step-by-Step Execution Including Microinstructions

#### 1. Fetching and Decoding

* **Step 1**: The program counter (PC) provides the address of the next machine code instruction. The CPU fetches this instruction from RAM.
* **Step 2**: The fetched machine code instruction is sent to the instruction decoder within the control unit.
* **Step 3**: The instruction decoder interprets the binary instruction, identifying the operation (e.g., ADD, SUB) and the operands (e.g., registers or memory addresses).

#### 2. Mapping to Microinstructions

* **Step 4**: Based on the decoded instruction, the control unit uses the machine code instruction's opcode to look up the starting address of the corresponding microinstruction sequence in control memory. This address is loaded into the Control Address Register (CAR).

#### 3. Fetching Microinstructions

* **Step 5**: The control unit fetches the first microinstruction from the control memory using the address in the CAR. This microinstruction is loaded into the Control Buffer Register (CBR).

#### 4. Decoding and Executing Microinstructions

* **Step 6**: The microinstruction in the CBR is decoded by the control unit to generate specific control signals.

-> These control signals orchestrate the operation of various parts of the CPU, such as the ALU, registers, buses, and memory.

* **Step 7**: The control signals direct the execution of low-level operations required to perform the decoded machine code instruction.

### Example Walkthrough: ADD R1, R2 (R1, R2 is CPU register)

1. **Fetching and Decoding the Machine Code Instruction**:
   * The machine code instruction ADD R1, R2 is fetched from RAM.
   * The instruction decoder interprets this instruction, identifying it as an ADD operation involving registers R1 and R2.
2. **Mapping to Microinstructions**:
   * The instruction decoder maps the ADD instruction to a starting address in control memory. Suppose the address is 0x100.
3. **Fetching and Executing Microinstructions**:
   * **Microinstruction 1**: Load the value of R2 into the ALU input register.
     + **Control Signals**: Enable read on R2, route data to ALU input register.
   * **Microinstruction 2**: Load the value of R1 into the ALU.
     + **Control Signals**: Enable read on R1, route data to ALU input register.
   * **Microinstruction 3**: Perform the addition operation in the ALU.
     + **Control Signals**: Set ALU to add mode, execute ALU operation.
   * **Microinstruction 4**: Store the result back into R1.
     + **Control Signals**: Route ALU output to R1, enable write on R1.

### Summary

* **Machine Code Instructions**: High-level binary instructions stored in RAM.
* **Instruction Decoder**: Within the control unit, it decodes machine code instructions to identify operations and operands.
* **Microinstructions**: Low-level instructions stored in control memory, providing detailed steps for executing machine code instructions.
* **Control Signals**: Generated from microinstructions, these signals control the CPU's hardware to perform operations.

Yes, to complete a machine code instruction, we often need one or more microinstructions !

### EXAMPLE to differentiate 3 concepts ->

 **Machine Code Instruction:** ADD R1, R2 (High-level instruction to add R2 to R1).

 **Microinstructions:** Encoded steps that direct the CPU on how to execute the ADD operation.

1. Load R1 to ALU input.
2. Load R2 to ALU input.
3. Perform addition in ALU.
4. Store result in R1.

 **Microoperations:** The actual operations performed by the CPU components.

1. IR <- MEM[PC] (Fetch instruction)
2. Decode IR (Decode instruction)
3. ALU\_input1 <- R1 (Load R1 into ALU input)
4. ALU\_input2 <- R2 (Load R2 into ALU input)
5. ALU\_output <- ALU\_input1 + ALU\_input2 (Add values in ALU)
6. R1 <- ALU\_output (Store result in R1)