Unit 2

Control unit

Topic:- control memory

Certainly, here's an easy-to-understand explanation of control memory in computer system organization in bullet points:

- **Traffic Cop**: Control memory is like a traffic cop inside the computer. It directs and manages the flow of operations to keep everything running smoothly.

- **Instruction Interpreter**: It reads and understands the instructions given to the computer by programs.

- **Micro-Operations**: Breaks down complex instructions into tiny steps or micro-operations that the computer can handle.

- **Sequencing**: Makes sure instructions are executed in the correct order, like following a recipe step-by-step.

- **Timing**: Controls the timing of tasks, ensuring everything happens at the right time, just like a conductor in an orchestra keeps the musicians playing in sync.

- **Decision Maker**: Handles choices, like deciding whether to add two numbers or subtract them based on the instruction.

- **Updates**: Can be modified to improve how the computer works, like updating software on your phone for better performance.

- **Critical for CPU**: It's a critical part of the CPU (central processing unit), which is like the brain of the computer.

- **Overall, it's like the computer's boss, making sure everything runs smoothly and efficiently**

Topic :- Address sequencing

Address sequencing in computer system organization is like finding and following a path on a map. It's all about how a computer locates and works with data stored in its memory. Here's a simple explanation:

1. **Addresses are Like Street Names**: In a computer's memory, data is stored at specific locations, just like houses have addresses on streets. Each piece of data has its own unique address in memory.

2. **Finding Data**: When a computer needs to use or change data, it must first find the right address where that data is stored. Think of it like looking up an address on a map.

3. **Sequencing Steps**: Address sequencing is the process of following a sequence of steps to get to the right data address. It's like following directions to reach a destination: "First, go straight, then turn left, and finally, it's the third house on the right."

4. **Accessing Data**: Once the computer reaches the correct memory address, it can access or change the data stored there. This is similar to arriving at a house after following directions and being able to enter or modify what's inside.

5. **Important for Programs**: Address sequencing is crucial for programs because they rely on finding and working with different pieces of data in memory. It ensures that the computer goes to the right place to get the information it needs.

6. **Efficiency Matters**: Just like taking the shortest or quickest route on a map saves time, efficient address sequencing helps the computer perform tasks faster and more effectively.

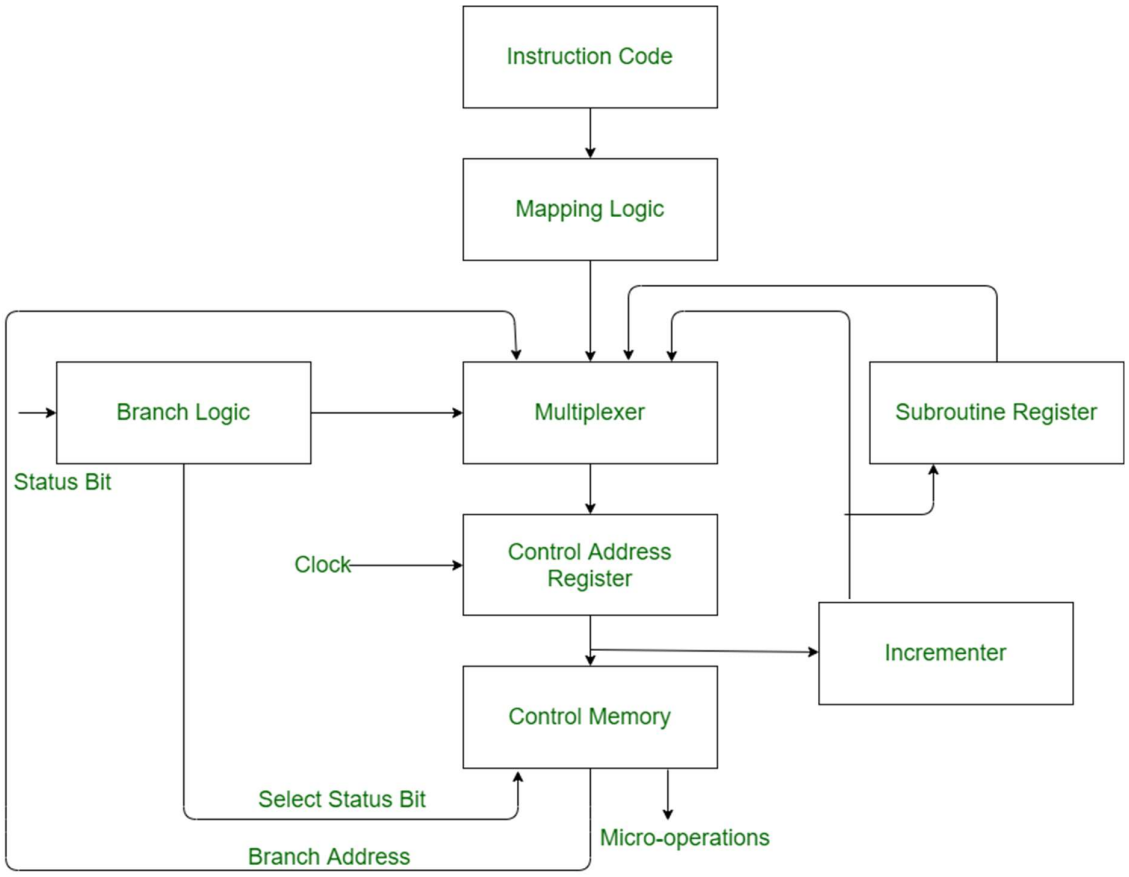
In a nutshell, address sequencing is the computer's way of navigating to specific data locations in memory so it can read, write, or manipulate that data. It's like a set of directions that guides the computer to the right "addresses" where the information it needs is stored.

The address is used by a microprogram sequencer to decide which microinstruction has to be performed next. Microprogram sequencing is the name of the total procedure. The addresses needed to step through a control store's microprogram are created by a sequencer, also known as a microsequencer.
The control unit's main job is to communicate with the computer's memory, input/output device, and ALU on how to respond to or carry out a set of instructions. Processing is not done within the Control unit itself. It only directs and manages the task. It serves as the computer's supervisor, regulating all operations including retrieving instructions from main memory and putting them to use.

Micro Instructions Sequencer

Micro Instructions Sequencer is a combination of all hardware for selecting the next micro-instruction address. The micro-instruction in control memory contains a set of bits to initiate micro-operations in computer registers and other bits to specify the method by which the address is obtained.

Implementation of Micro Instructions Sequencer



- **Control Address Register(CAR) :** Control address register receives the address from four different paths. For receiving the addresses from four different paths, Multiplexer is used.
- **Multiplexer :** Multiplexer is a combinational circuit which contains many data inputs and single data output depending on control or select inputs.
- **Branching :** Branching is achieved by specifying the branch address in one of the fields of the micro instruction. Conditional branching is obtained by using part of the micro-instruction to select a specific status bit in order to determine its condition.
- **Mapping Logic :** An external address is transferred into control memory via a mapping logic circuit.
- **Incrementer :** Incrementer increments the content of the control address register by one, to select the next micro-instruction in sequence.
- **Subroutine Register (SBR) :** The return address for a subroutine is stored in a special register called Subroutine Register whose value

is then used when the micro-program wishes to return from the subroutine.

- **Control Memory :** Control memory is a type of memory which contains addressable storage registers. Data is temporarily stored in control memory. Control memory can be accessed quicker than main memory.

Micro program

Microprogramming is a technique used in computer system organization to control the internal operations of a CPU (Central Processing Unit). It involves using a set of microinstructions to execute complex instructions from a higher-level programming language. Here are some examples of microprograms:

1. **Arithmetic Operation**:

- Microinstructions can be used to perform arithmetic operations like addition or subtraction. For example, a microprogram can include microinstructions to fetch two numbers from memory, add them, and store the result back in memory.

2. **Data Movement**:

- Microprograms can handle data movement operations, such as loading data from memory into a CPU register or transferring data between different registers.

3. **Conditional Branching**:

- Conditional branching in microprogramming allows the CPU to make decisions based on the results of previous operations. For instance, a microprogram can include instructions to compare two values and branch to different microinstructions depending on the comparison result.

4. **Function Calls**:

- Microprograms can manage function calls and returns. When a high-level programming language function is called, the microprogram can save the return address and set up a stack to manage the function's local variables.

5. **Interrupt Handling**:

- Microprograms can be used to handle interrupts from external devices. When an interrupt occurs, the microprogram can save the current state of the CPU, execute a specific routine to handle the interrupt, and then return to the interrupted program.

6. **Floating-Point Arithmetic**:

- Complex floating-point arithmetic operations can be implemented using microprogramming. This allows the CPU to perform calculations involving real numbers with high precision.

7. **Memory Access Control**:

- Microprograms can control how the CPU interacts with memory, including fetching data, writing data, and handling cache operations.

8. **Pipeline Control**:

- In pipelined processors, microprograms can manage the different stages of instruction execution, ensuring that instructions flow smoothly through the pipeline.

9. **Input/Output Operations**:

- Microprogramming can be used to control input/output operations, including sending and receiving data to and from peripheral devices.

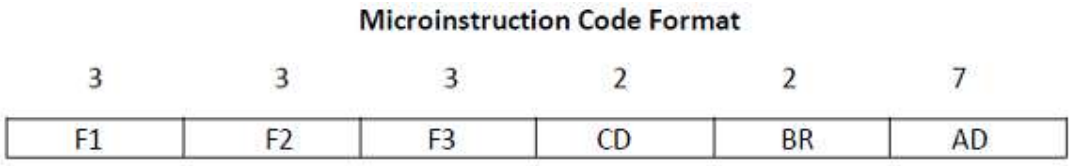
10. **Exception Handling**:

- Microprograms are essential for handling exceptions and errors in the CPU, such as divide-by-zero errors or illegal instruction exceptions.

Microprograms are stored in a control memory or microcode ROM within the CPU. They provide a way to control the intricate internal operations of the CPU, translating high-level instructions into a series of microinstructions that the CPU can execute. This allows for flexibility and customization in designing CPU architectures to support different instruction sets and operations.

Micro instruction format

A microinstruction format includes 20 bits in total. They are divided into four elements as displayed in the figure.



F1, F2, F3 are the micro-operation fields. They determine micro-operations for the computer.

CD is the condition for branching. They choose the status bit conditions.

BR is the branch field. It determines the type of branch.

AD is the address field. It includes the address field whose length is 7 bits.

The micro-operations are divided into three fields of three bits each. These three bits can define seven different micro-operations. In total there are 21 operations as displayed in the table.

Symbols with their Binary Code for Microinstruction Fields

	Name:	Code	Symbol

	Name:	Code	Symbol
F1	000	None	NOP
	001	AC ← AC + DR	ADD
	010	AC ← 0	CLRAC
	011	AC ← AC + 1	INCAC
	100	AC ← DR	DRTAC
	101	AR ← DR(0 − 10)	DRTAR
	110	AR ← PC	PCTAR
	111	AC ← AC + DR	WRITE
F2	000	None	NOP
	001	AC ← AC + DR	SUB
	010	AC ← AC ∨ DR	OR
	011	AC ← AC ∧ DR	AND
	100	DR ← M[AR]	READ
	101	DR ← AC	ACTDR
	110	DR ← DR + 1	INCDR
	111	DR(0 − 10) ← PC	PCTDR
F3	000	None	NOP

	Name:	Code	Symbol
	001	AC ← AC ⊕ DR	XOR
	010	AC ← AC’	COM
	011	AC ← shl AC	SHL
	100	AC ← shr AC	SHR
	101	PC ← PC + 1	INCPC
	110	PC ← AR	ARTPC
	111	DR(0 – 10) ← PC	Reserved

As shown in the table, each microinstruction can have only three micro-operations, one from each field. If it uses less than three, it will result in more than one operation using the no operation binary code.

Condition Field

A condition field includes 2 bits. They are encoded to define four status bit conditions. As stated in the table, the first condition is always a 1, with CD = 0. The symbol that can indicate this condition is ‘U’. The table displays the multiple condition fields and their summary in an easy manner.

Condition Field Symbols and Descriptions

	Condition	Symbol	Comments
00	Always = 1	U	Unconditional Branch
01	DR (15)	I	Indirect address bit

	Condition	Symbol	Comments
10	AC (15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

As shown in the table, when condition 00 is connected with **BR** (branch) field, it results in an unconditional branch operation. Then the execution is read from memory the indirect bit I is accessible from bit 15 of **DR**. The status of the next bit is supported by the AC sign bit. If all the bits in **AC** are 1, then it is indicated as Z (its binary variable whose value is 1). The symbols **U, I, S**, and **Z** can indicate status bits while writing microprograms.

Branch Field

The BR (branch) field includes 2 bits. It can be used by connecting with the AD (address) field. The reason for connecting with the AD field is to select the address for the next microinstruction. The table illustrates the various branch fields and their functions.

Branch Field Symbols and Descriptions

BR	Symbol	Comments
00	JMP	CAR ←AD if condition = 1 CAR←CAR + 1 if condition = 0
01	CALL	CAR ←AD , SBR ← CAR +1, if condition = 1 CAR←CAR + 1 if condition = 0
10	RET	CAR ←SBR (Return from subroutine)
11	MAP	CAR(2-5) ←DR(11-14), CAR(0,1,6) ←0

As shown in the table, when **BR = 00**, a **JMP** operation is implemented and when **BR = 01**, a subroutine is called. The only difference between the two instructions is that when the microinstruction is saved, the return address is saved in the **Subroutine Register (SBR)**.

These two operations are dependent on the **CD** field values. When the status bit condition of the CD field is defined as **1**, the address that is next in order is transferred to **CAR.** Else, it gets incremented. If the instruction needs to return from the subroutine, its **BR** field is determined as **10**.

This results in the transfer of the return address from **SBR** to **CAR**. The opcode bits of instruction can be mapped with an address for **CAR** if the **BR** field is **11**. They are present in **DR (11 - 14)** after instruction is read from memory. The last two conditions in the BR fields are not dependent on the **CD** and **AD** field values.

Symbolic microinstructions

What are Symbolic Microinstructions?

Computer ArchitectureComputer ScienceNetwork

The microinstructions can be determined by symbols. It is interpreted to its binary format with an assembler. The symbols should be represented for each field in the microinstruction. The users should be enabled to represent their symbolic addresses. Each line in an assembly language represents symbolic instruction. These instructions are divided into five fields such as label, micro-operations, CD, BR, and AD.

The fields that specify the following information are as follows –

- The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
- The micro-operations field consists of one, two, or three symbols, separated by commas. But each F field includes only a single symbol.
- The CD field has one of the letters U, I, S, or Z.
- The BR field contains one of the four symbols defined.
- The AD field specifies a value for the address field of the microinstruction in one of three possible ways –
    - With a symbolic address, which must also appear as a label.

- With the symbol NEXT to designate the next address in sequence.
- When the BR field includes a RET or MAP symbol, the AD field is left null and is transformed to seven zeros by the assembler.

Fetch Routine

The control unit includes 128 words, each including 20 bits. The value of each bit should be specified to microprogram the control memory. Between the 128 words, the first 64 are composed for the routines of 16 instructions. The remaining 64 can be used for different goals. The best opening location for the fetch routine to start is the 64th address.

The microinstructions necessary for fetch routine are −

AR ← PC

DR ← M[AR], PC ← PC + 1

AR ← DR(0 − 10), CAR(2 − 5) ← DR(11 − 14), CAR(0,1,6) ← 0

The address of the instruction is transferred from PC to AR and the instruction is then read from memory into DR. Since no instruction register is available, the instruction code remains in DR. The address part is transferred to AR and then control is transferred to one of 16 routines by mapping the operation code part of the instruction from DR into CAR.

Microinstructions that are situated in addresses 64, 65, and 66 are important for the fetch routine. There are various symbolic language is as follows −

```
        ORG 64
FETCH:  PCTAR U JMP NEXT
        READ, INCPC U JMP NEXT
        DRTAR U MAP
```

The table shows the results of binary translation for the assembly language.

Binary Translation for Assembly Language

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	101	000	00	11	0000000

Each microinstruction executes the internal register transfer operation displayed by the register transfer representation. The representation in symbols is important while writing microprograms in an assembly language format. The actual internal content which is saved in the control memory is in binary representation.

Another easy to understand what is tech routine,

In computer science and engineering (CSE), a "fetch routine" is a term that describes a specific part of the process that a computer's central processing unit (CPU) goes through when it wants to retrieve or "fetch" an instruction from memory. Let's break down what happens in a fetch routine in easy-to-understand steps:

1. **Start**: The fetch routine begins when the CPU wants to execute the next instruction in a computer program. It needs to get this instruction from memory so it knows what operation to perform next.

2. **Memory Address**: The CPU knows the memory address (location) of the next instruction it needs. Think of this like knowing the page number in a book where you want to read something.

3. **Memory Access**: The CPU communicates with the computer's memory system and says, "I need the instruction at this memory address." It's similar to asking a librarian to find a specific page in a book.

4. **Instruction Retrieval**: The memory system then goes to that memory address, grabs the instruction stored there (like taking a book off the shelf), and sends it back to the CPU.

5. **Instruction Loading**: The CPU receives the instruction and loads it into a special place in its own memory called the "instruction register." This is like opening the book to the page you wanted to read.

6. **Ready to Execute**: With the instruction now in the instruction register, the CPU is ready to execute it. It knows what operation to perform, such as adding two numbers or moving data from one place to another.

7. **Repeat**: The CPU goes through this fetch routine over and over again, one instruction at a time, to carry out the steps of a computer program.

In simple terms, a fetch routine is like a computer's way of picking up instructions from memory, just as you pick up a book from the library shelf to read. It ensures that the CPU always knows what to do next in a program by retrieving the right instruction from memory.

Working of fetch routine

A fetch routine is a critical part of the instruction cycle in a computer's central processing unit (CPU). It's responsible for fetching the next instruction from memory and preparing it for execution. Let's describe how a fetch routine works in a step-by-step manner:

1. **Program Counter (PC)**: The fetch routine begins with the Program Counter (PC), which is a special register in the CPU. The PC keeps track of the

memory address of the next instruction to be executed. It's like a bookmark in a book, telling the CPU where to find the next instruction.

2. **Memory Request**: The CPU sends a request to the computer's memory system, specifying the memory address pointed to by the PC. This request asks for the instruction located at that address. Think of it as asking the librarian to fetch a specific book from the library's shelves.

3. **Memory Access**: The memory system receives the request and goes to the specified memory address, where it retrieves the instruction. It's similar to the librarian finding the book you asked for on the library shelf.

4. **Instruction Fetch**: The retrieved instruction is then sent back to the CPU. This is akin to the librarian handing you the book you requested.

5. **Instruction Storage**: The CPU loads the fetched instruction into a special storage area within itself called the "instruction register" (IR). The IR holds the current instruction that the CPU is working on. It's like opening the book to the page you want to read and placing it on a stand for easy access.

6. **Program Counter Update**: After the fetch is complete, the PC is updated to point to the memory address of the next instruction. This prepares the CPU for the next fetch. It's like moving the bookmark in your book to the next page you intend to read.

7. **Execution**: With the instruction now in the IR and the PC updated, the CPU is ready to execute the fetched instruction. It interprets the instruction, performs the specified operation (e.g., arithmetic, data transfer, or branching), and potentially updates the PC again to prepare for the next instruction in the sequence.

8. **Repeat**: This fetch routine is repeated continuously, allowing the CPU to execute instructions in the correct order, one after another, as dictated by the program.

In summary, a fetch routine is a sequence of steps that the CPU follows to retrieve the next instruction from memory, store it in the instruction register, update the program counter for the next instruction, and then execute the fetched instruction. This cycle continues until the program is complete, allowing the computer to perform the tasks specified by the program's instructions.

Symbolic micro program and design of control unit

A symbolic microprogram and the design of a control unit are closely related concepts in computer organization and architecture. Let's break down each concept separately:

1. Symbolic Microprogram:

A symbolic microprogram is a way to represent the microinstructions that control the operation of a computer's central processing unit (CPU) using symbols and mnemonics instead of binary code. It provides a more human-readable and abstract way of specifying how the CPU should execute instructions.

Here's how symbolic microprogramming works:

- **Microinstructions**: In a CPU's control unit, there are microinstructions that define the low-level operations required to execute each machine-level instruction. These microinstructions control various aspects of the CPU, such as data movement, arithmetic operations, and branching.

- **Symbolic Representation**: Instead of using binary code to represent these microinstructions, symbolic microprogramming uses human-friendly symbols and mnemonics. For example, instead of writing a binary sequence like "100110101001," you might use a symbolic representation like "LOAD A" to indicate loading a value into register A.

- **Microinstruction Sequences**: Symbolic microprogramming also involves defining sequences of microinstructions to perform more complex operations or execute machine-level instructions. These sequences are often represented as symbolic macros or subroutines.

- **Advantages**: Symbolic microprogramming makes it easier for computer architects and designers to understand and modify the control unit's behavior. It simplifies the design process and allows for greater flexibility in designing CPU architectures. Additionally, it aids in debugging and maintenance.

7.3 Microprogram example

TABLE 7-2 Symbolic Microprogram (Partial)

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

33	0 ADD 86
34	0 BRANCH 40
...	...
40	0 STORE 90
41	1 EXCHNG 77
..	
77	90
..	
90	58

Memory

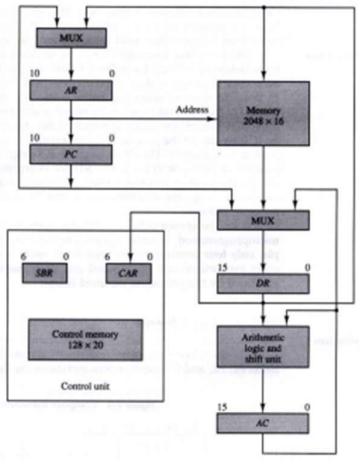


Figure 7-4 Computer hardware configuration.

2. Design of Control Unit:

The control unit is a crucial component of a CPU responsible for managing and coordinating the execution of instructions. It interprets machine-level instructions and generates the necessary control signals to coordinate the operation of various CPU components, such as registers, ALU (Arithmetic Logic Unit), and memory.

Here's how the design of a control unit typically works:

- **Instruction Decoding**: The control unit decodes machine-level instructions to determine the microoperations needed to execute them. Each instruction is associated with a specific microprogram or microinstruction sequence.

- **Microprogram Design**: The control unit's design includes defining the microinstructions that correspond to each microoperation. These microinstructions control data paths and control signals within the CPU.

- **Control Signals**: The control unit generates control signals that instruct various CPU components on what to do. For example, it might signal the ALU to perform addition, the memory to read data, or the registers to store values.

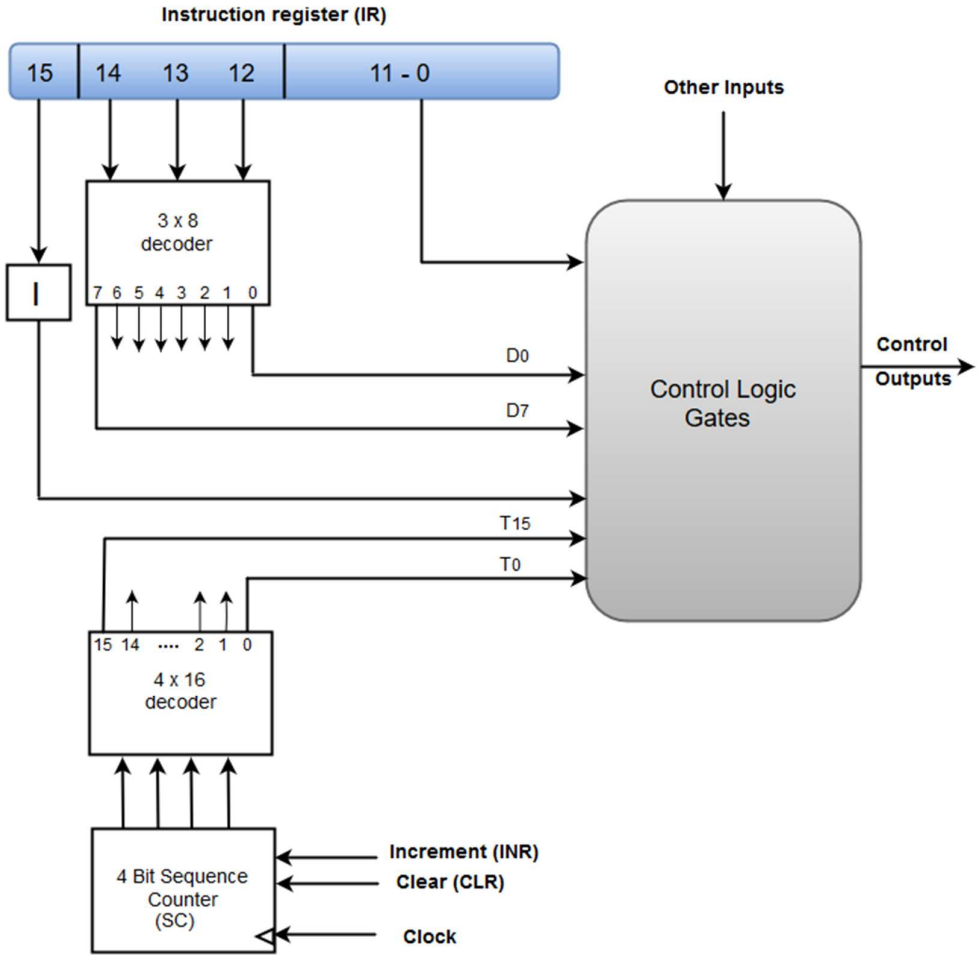
- **Timing and Sequencing**: The control unit ensures that microoperations occur in the correct sequence and at the right time within the CPU. It synchronizes clock signals and data flow to maintain proper timing.

- **State Machine**: The control unit often operates as a finite state machine, where it transitions between different states based on the current instruction being executed. Each state corresponds to a specific microinstruction sequence.

- **Microprogram Storage**: The microinstructions and microprograms are typically stored in a control memory or control store within the CPU. This is where the symbolic microprogramming representation comes into play.

In summary, the design of a control unit involves creating a set of microinstructions that define how the CPU should operate, and symbolic microprogramming is a way to represent these microinstructions in a more human-readable form. Together, they enable the control unit to effectively control the execution of instructions within the CPU.

Control Unit of a Basic Computer:



Microprogram sequencing

In computer system organization, a microprogram sequencer is a key component of the control unit in a CPU (Central Processing Unit). It plays a critical role in executing machine-level instructions by controlling the sequence of microinstructions stored in the control memory (also known as microcode memory). Let's break down what a microprogram sequencer is and how it works:

1. Control Memory (Microcode Memory): Control memory is a part of the control unit that stores microprograms. A microprogram is a sequence of microinstructions that controls various CPU operations. These microinstructions are responsible for executing machine-level instructions, managing data movement, and controlling the CPU's components.

2. Microprogram Sequencer:

- The microprogram sequencer is responsible for determining the sequence in which microinstructions from the control memory are fetched and executed during the execution of a machine-level instruction.

- It acts as a sort of program counter for microinstructions. Just as the program counter (PC) keeps track of the address of the next machine-level instruction to execute, the microprogram sequencer keeps track of which microinstruction to fetch and execute next.

- The sequencer typically uses the current state of the CPU, the current machine-level instruction being executed, or other control signals to decide which microinstruction to fetch next. It ensures that the microinstructions are executed in the correct order and timing to carry out the desired operation.

3. Microinstruction Fetch and Execution:

- When a machine-level instruction is decoded by the control unit, the microprogram sequencer starts fetching microinstructions from the control memory. It begins with the first microinstruction in the sequence.

- As each microinstruction is fetched, it is executed, controlling various operations in the CPU, such as data movement, arithmetic operations, and memory access.

- After executing a microinstruction, the sequencer determines the next microinstruction to fetch based on the current microinstruction's outcome and the control signals. This sequence continues until all microinstructions for the machine-level instruction have been executed.

4. Flexibility and Control:

- The microprogram sequencer provides flexibility in CPU design because it allows designers to define and modify microinstruction sequences to support different instruction sets or architectures without changing the hardware components of the CPU.

- It also enables conditional branching within microprograms. For example, based on the outcome of a comparison, the sequencer can decide to jump to a different part of the microprogram to handle different conditions or outcomes.

In summary, a microprogram sequencer is a critical component of the control unit in a CPU. It controls the sequence in which microinstructions are fetched and executed from the control memory, ensuring that machine-level instructions are executed correctly and efficiently. It provides the CPU with the ability to execute a wide range of instructions and operations through the use of microcode, offering flexibility and ease of modification in CPU design.

Microprogram Sequencer