Program : **B.Tech**

Subject Name: **Computer Organization and Architecture**

Subject Code: **CS-404**

Semester: **4th**

UNIT – 2 Control Unit Organization - Basic Concept of Instruction, Instruction Types, Micro Instruction Formats, Fetch and Execution cycle, Hardwired control unit, Micro-programmed Control unit - micro program sequencer Control Memory, Sequencing and Execution of Micro Instruction.

BASIC CONCEPT OF INSTRUCTION

The user of a computer can control the process by means of a program. A program is a set of instruction that specify the operations, operands and the sequence by which processing has to occur. The data processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data. A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. Instruction codes together with da ta are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general purpose computer. The operation of the central processing - unit and the computer system is determined by the instructions executed by central processing unit. These instructions are known as machine instructions or computer instructions. Machine instructions are in the form of binary codes. A particular sequence of these binary codes used to perform particular task is known as machine language program. Each instruction of the CPU has specific information fields, which are required to execute it. These information fields of instructions are called elements of instruction.

An instruction code of a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group p of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required f or the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits f or a given 2"(or less) distinct operations.

An operation is part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro-operations in internal computer registers. For every operation code, the control issues a sequence of micro-operations needed f or the hard- ware implementation of the specified operation. For this reason, an operation code is sometime s called a macro-operation because it specifies a set of micro- operations.
The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored. Memory words can be specified in instruction codes by their address. Processor register s can be specified by assigning to the instruction another binary code of k bits that specifies one of 2 k registers. There are many variations f or arranging the binary code of instructions, and each computer has its own particular instruction code format. Instruction code formats are conceived by computer designers who specify the architecture of the computer

INSTRUCTION TYPES & FORMATS

A basic computer has **three types** of instructions:

1. Memory reference instructions
2. Register reference instructions
3. Input-Output instructions

Memory Reference Instructions - The Fig.1 shows the instruction format for memory reference instructions. Each memory reference instruction has 16-bits. Out of 16-bits:

I-Bit (I) specifies addressing mode : Direct or indirect
3-Bits (Opcode) specify the opcode and
12-Bits (Address) specify the address.

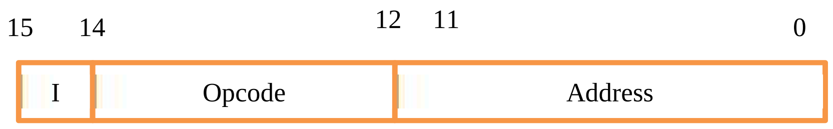


Figure. 1 Memory reference Instruction format

I = 0 : Direct addressing mode
I = 1 : Indirect addressing mode
Opcode : It can have value from 000 through 110 since there are 7 memory reference instructions

Example:

Mnemonic	Description	Instruction Code in HEX	
AND	Logically ANDs the contents of Specified memory location and AC AC ⍰AC + M {AR}	**I=0** 0xxx	**I=1** 8xxx
STA	Store the contents of AC in the specified memory location M [AR]⍰AC	1xxx	9xxx

Register Reference Instructions - A register reference instruction specifies an operation or a test to be performed with AC register. These instructions do not need to access memory and hence 12-bits are used to specify an operation or a test to be performed.

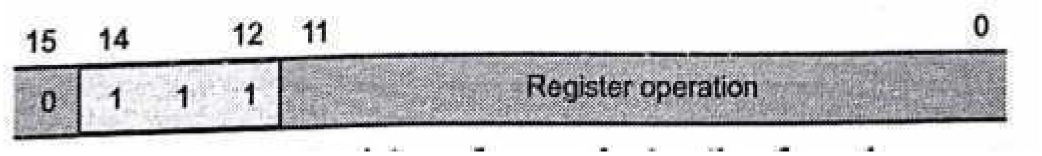


Figure 2. Register reference instruction format

Example:

Mnemonic	Description
CME	Complement E bit

INC	Increment AC register
SPA	Skip next instruction if contents of AC register are +ive

Input-Output Instructions - Like register reference instructions, input-output instructions do not need memory reference. The opcode and I bit for these instructions are 111 and 1, respectively. The remaining 12-bits specify the type of input-output operation or test to be performed.

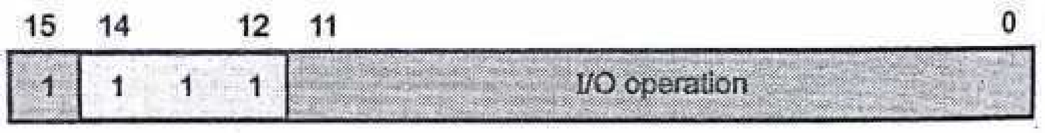


Figure 3. Input-output instruction format

Example:

Mnemonic	**Description**
INP	Load a character in AC register from input port
OUT	Send a character to outpot port from AC register

FETCH AND EXECUTION CYCLE

Instruction Cycle

The most basic unit of computer processing in the simplest form, consists of two parts.

1. ***Opcode*** (operation code) – a portion of a machine language instruction that specifies the operation to be performed.
2. ***Operands*** – a part of a machine language instruction that specifies the data to be operated on

The simplest model of instruction processing can be a two step process. The CPU **reads (fetches)** instructions (codes) from the memory one at a time, and **executes**. Instruction fetch involves reading of an instruction from a memory location to the CPU register. The execution of this instruction may involve several operations depending on the nature of the instruction. Instructions are processed by the control unit in a systematic, step-by-step manner. The sequence of steps in which instructions are loaded from memory and executed is called instruction cycle. Each step in the sequence is referred to as a phase. Fundamentally, there are 6 phases.

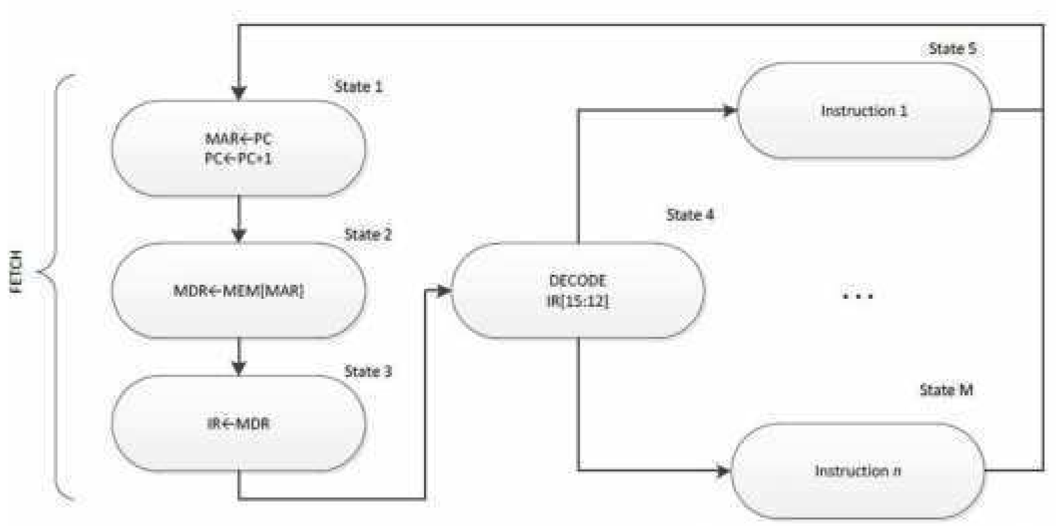
1. **FETCH (instruction) -** This phase obtains the next instruction from memory and stores it in the IR. The address of the next instruction to be executed is stored in the PC register. Proceeds in the following manner

   MAR ← PC (memory address register is loaded with the content of PC). PC ← PC + 1
   (value stored in the PC is incremented by one)
   MDR ← MEM [MAD] (interrogate memory, resulting in the instruction being placed in the MDR).
   IR ← MDR (load the instruction from MDR to the instruction register).

   For now, we will say that each of these steps proceeds in one *machine cycle.* Note that the instruction to be executed is now stored in IR and the address of the *next* instruction to be executed is stored in PC

2. **DECODE -** In this phase the instruction stored in PC is examined in order to decide what portion of the micro architecture needs to be involved in the execution of the instruction. For example, for a 4-bit opcode, this can be implemented as a 4-to-16 decoder. This decoder will examine bits 12-15 stored in the IR and will activate the appropriate circuitry necessary to carry out the instruction

3. **EVALUATE ADDRESS -** This phase Compute the address of the memory location that is needed to process the instruction. Some instructions do not need this phase, e.g., instructions that work directly with the registers and do not require any operands to be loaded or stored form memory.



4. **FETCH OPERANDS -** In this phase, the source operands needed to carry out the instruction are obtained from memory. For some instructions, this phase equals to loading values form the register file. For others, this phase involves loading operands from memory

5. **EXECUTE -** In this phase instruction is carried out. Some instructions may not require this phase, e.g., data movement instructions for which all the work is actually done in the FETCH OPERANDS phase 6. **STORE RESULTS -** In this phase the result is written to its designated destination.

After the 6 phases of the instruction cycle are done, the control unit begins the next instruction cycle, starting with the new FETCH (instruction) phase. Since the PC was previous incremented by one, it contains the pointer to the next instruction to be fetched and executed

An instruction cycle basic involves three sub cycles.

1. **Fetch**
2. **Decode**
3. **Execute**

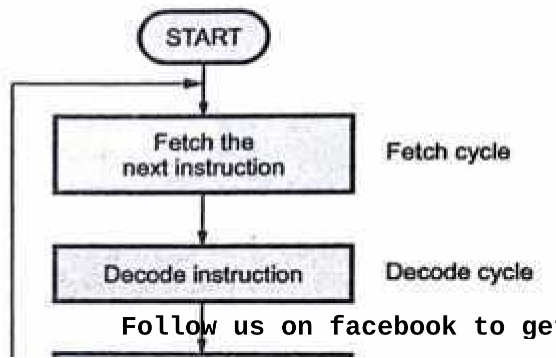


Figure 4. shows the basic instruction cycle.

The fetch phase reads the next instruction from memory into the CPU. The decode phase interprets the opcode by decoding it. The execute phase performs the indicated operation.

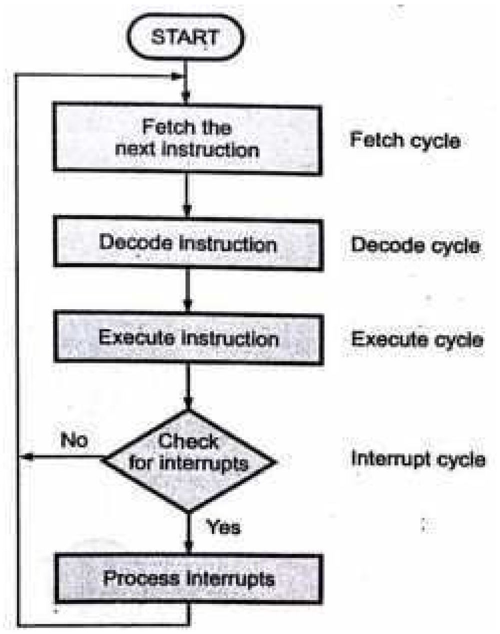


Figure 5. Basic instruction cycle with interrupt cycle

Actually, CPU checks for valid interrupt request after each instruction cycle. If any valid interrupt request is present, CPU saves the current process state, and services the interrupt. Servicing the interrupt means executing interrupt service routine. After completing it, CPU starts the new Instruction cycle from where it has been interrupted. Fig. 5 shows this instruction cycle with interrupt cycle.

The indirect cycle: If the operands on Which the instruction works are present Within the CPU-registers, a memory access is not required. But if the execution of an instruction involves one or more operands in memory, each requires a memory access. If indirect addressing is used then additional memory accesses are required. For fetching the indirect addresses, one or more instruction sub cycles are required. After fetching the instruction, it is decoded and if any indirect addressing is involved, the required operands are fetched using indirect addressing. Also, after performing the operation on the operands according to the opcode, a similar process may be needed to store the result in memory. Following execution, interrupt processing may be required before fetching the next instruction. The same process can be viewed as shown in Fig 6.

Fetch cycle - Initially, the program counter PC is loaded with the address of the first instruction in the program. To provide decoded timing signal To, the Sequence Counter (SC) is cleared to 0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2 and so on.

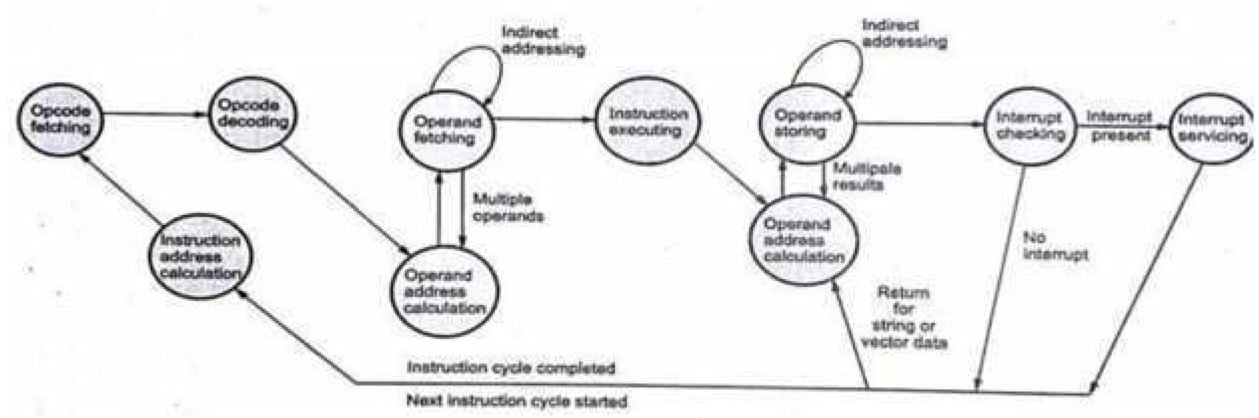


Figure 6. Instruction cycle state diagram

Register transfer statements for fetch cycle are:

T0 : AR⍰PC
T1: IR⍰M[AR], PC⍰PC+1

At To, the address from PC is transferred from PC to AR. The instruction read from the memory is then placed in the Instruction Register (IR) during T1. At the same time, PC is incremented by one.

The Fig. 6 shows the implementation of the first two register transfer statements in the common bus system.

In T0: 1. The contents of PC are placed onto the common bus by enabling its EN input by setting S2 S1 S0 =010
2. The contents of the common bus are transferred to AR by enabling its LD input.

In T1: 1. Read input of the memory is enabled by setting S2 S1 S0 = 1 1 1. This places the contents of memory onto the bus.
2. The contents of common bus are transferred to IR by enabling its LD input.
3. PC is incremented by enabling the INR input of PC.

Decode cycle - At T2, decoding s instruction is done. Register transfer statement for decode cycle is:

T2 : D0……D7 ⍰ Decode IR (12 - 14), AR ⍰ IR (0 - 11), I ⍰ IR (15)

In T2 : 1. Bits 12 - 14 from IR are decoded using 3 : 8 decoder.
2. Bits 0-11 from IR are loaded into the AR.
3. Bit 15 of IR is loaded into the addressing mode (I).

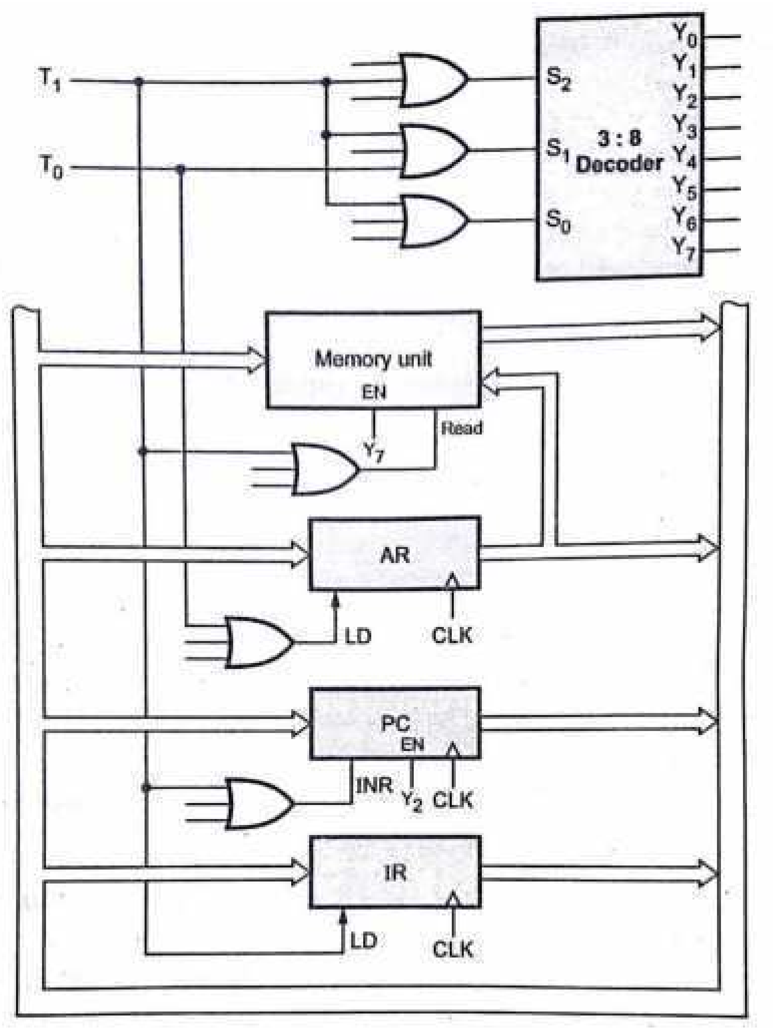
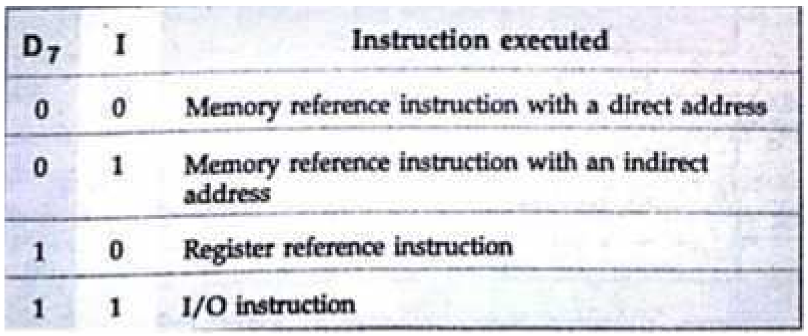


Figure 7. implementation of register transfer instruction for fetch cycle

Determination of type of Instruction

The Fig. 8 shows how the control circuitry determines the type of instruction after the decoding. As shown in the Fig. 8, if decoder output D7 = 0, it is memory reference instruction; otherwise, it is as register reference or I/O instruction. According to D7 and I bits, different instructions are executed listed in Table 1.

Table 1



D_7	I	Instruction executed
0	0	Memory reference instruction with a direct address
0	1	Memory reference instruction with an indirect address
1	0	Register reference instruction
1	1	I/O instruction

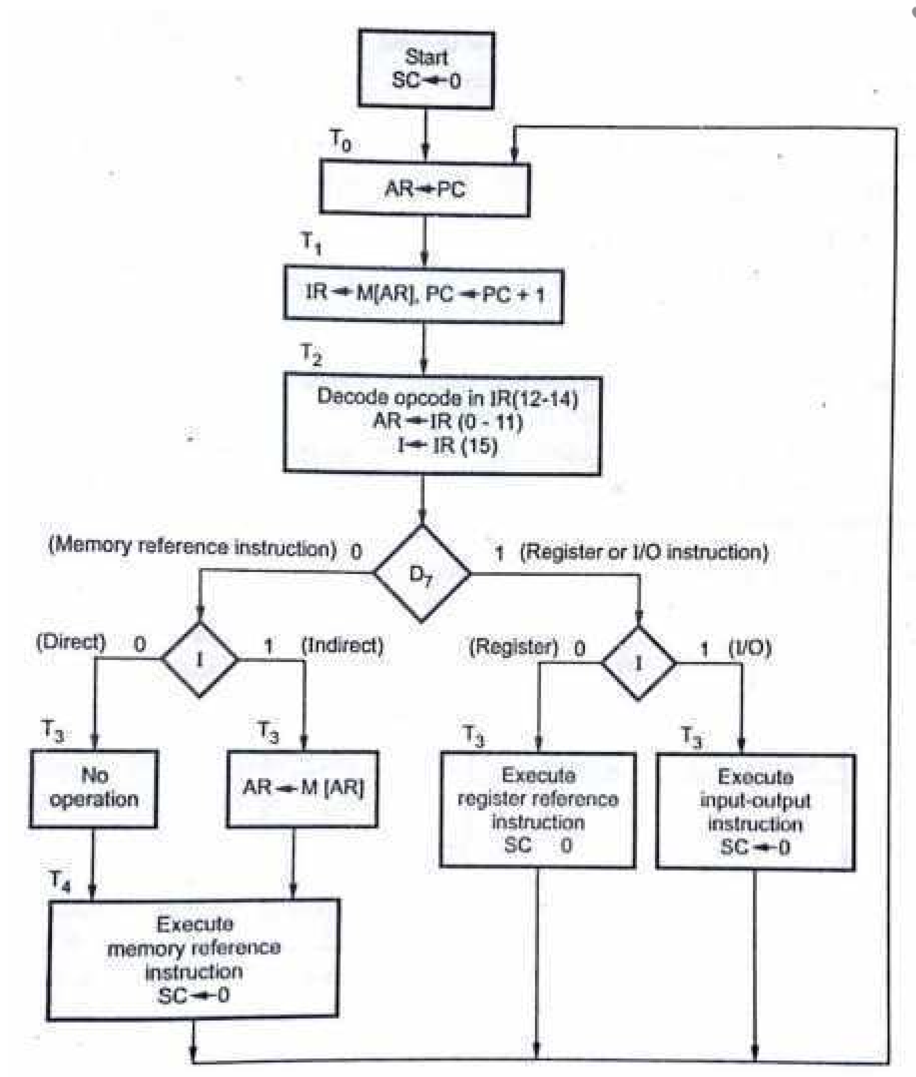

Start
SC←0
T0
AR←PC
T1
IR←M[AR], PC←PC+1
T2
Decode opcode in IR(12-14)
AR←IR (0 - 11)
I←IR (15)
(Memory reference instruction) 0
D7
1 (Register or I/O instruction)
(Direct) 0
I
1 (Indirect)
(Register) 0
I
1 (I/O)
T3
No operation
T3
AR←M [AR]
T3
Execute register reference instruction SC 0
T3
Execute input-output instruction SC←0
T4
Execute memory reference instruction SC←0

Figure 8. Flowchart for Instruction cycle

HARDWIRED CONTROL UNIT

The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro-operations for the accumulator. There are **two major types** of control organization: **Hardwired Control** and **Micro-programmed Control**. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the **advantage** that it can be optimized to produce a **fast mode of operation**. In the micro-programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro-operations. A hardwired control, as the name implies, re- quires changes in the wiring among the various components if the design has to be modified or changed. In the micro-programmed control, any required changes or modifications can be done by updating the micro-program in control memory. A hardwired control for the basic computer is presented in this section.

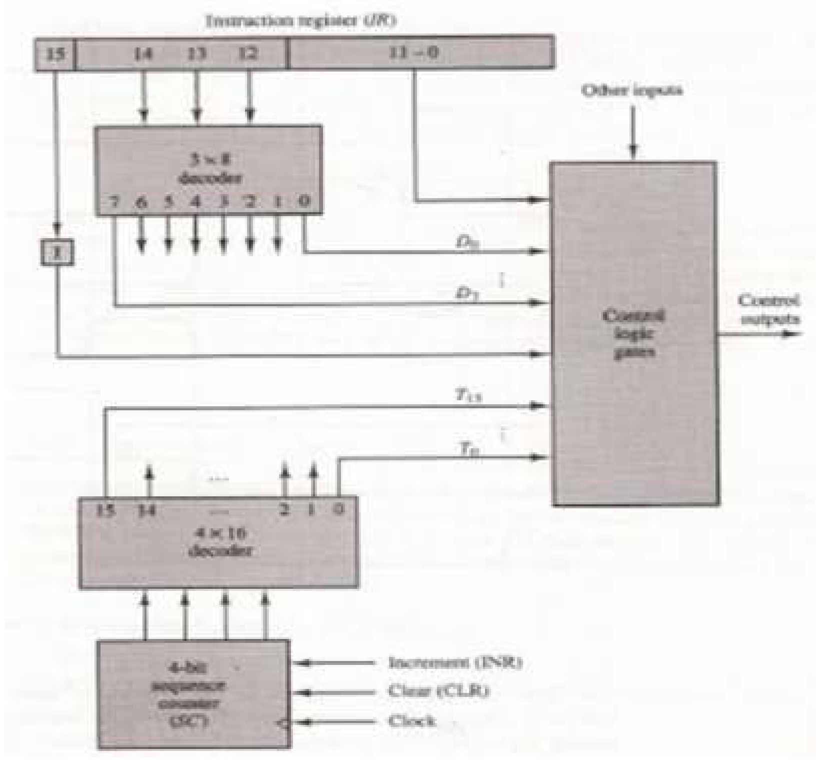


Figure 9. Hardwired Control Unit of Basic Computer

In the hardwired control, the **control units use fixed logic circuit** to interpret instructions and generate control signals from them. The fixed logic circuits use contents of the control step counter, contents of the instruction register, contents of the condition code flag and the external input signals such as MFC and interrupt requests to generate control Signals.

The block diagram of the control unit is shown in Figure 9. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register **(IR).** The position of this register in the common bus system is indicated in Fig. The instruction register is shown again in above Fig. where it is divided into three parts: the 1 bit, the operation code, and bits **0** through **11**. The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols **D0** through **D7**. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol **1**. Bits **0** through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals **To** through **T15**. The internal logic of the control gates can be derived when we consider the design of the computer in detail. The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be **To**.

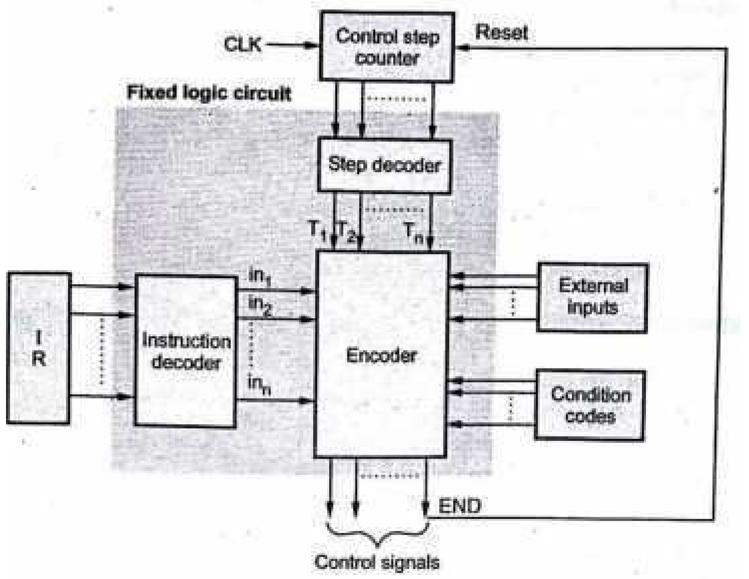


Figure 10. Detail block diagram for hardwired control unit

Figure 10 shows the typical hardwired, control unit. Here, the fixed logic circuit block includes combinational circuit (decoder and encoder) that generates the required control outputs, depending on the state of all its inputs. By separating the decoding and encoding functions, we have shown more detail block diagram for hardwired control unit as shown in the Fig. 10. The instruction decoder decodes the instruction loaded in the IR. If IR is an 8-bit register then instruction decoder generates 2^8 i.e.
256 lines; one for each. According to code in file IR, only one line amongst output lines of decoder goes high i.e. set to l and all other lines are set to 0. The Step decoder provides a separate signal line for each step, or time slot, in a control sequence. The encoder gets in the input from instruction decoder, step decoder, external inputs and condition codes. It uses all these inputs to generate the individual control signals. After execution of each instruction end signal is generated this resets control step counter and make it ready for generation of control step for next instruction.

Advantages of Hardwired Control Unit

1. Hardwired control unit is fast because control signals are generated by combinational circuits.
2. The delay in generation of control signals depends upon the number of gates.
3. It has greater chip area efficiency since its uses less area on-chip.

Disadvantages of Hardwired Control Unit

1. More the control signals required by CPU; more complex will be the design of control unit.
2. Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit:
3. It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

Design Methods of Hardwired Control Unit - There are four simplified and systematic methods for the design 0f hardwired controllers.

1. **State-table Method**: It is the standard algorithmic approach to sequential circuit design.
2. **Delay-element Method**: It is a heuristic method based on the use of clocked delay elements for control signal timing.
3. **Sequence-counter Method**: It uses counters for timing purposes.

4. **PLA Method**: It uses programmable logic array.

CONTROL MEMORY

Every instruction in a processor is implemented by a sequence of one or more, sets of, concurrent micro-operations. Each micro-operation is associated with a specific set of control lines which, when activated, causes that micro-operation to take place. In the hardwired control, the control unit uses fixed logic circuits to interpret instructions and generate control signals from them. Micro-programming is an elegant and systematic method for controlling the .micro—operation sequences. Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high Thus, it is costly and difficult to design. Furthermore, the hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

An advance development known as **Dynamic Micro-programming** permits a micro- program to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic micro-programming use a writable control memory. This type of memory can be used for writing (to change the micro-program) but is used mostly for reading. A memory that is part of a control unit is called a **Control Memory.**

Micro-programming is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM or RAM **called a control memory CM**. The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM in much similar way an instruction is fetched from main memory. Each micro-instruction also explicitly or implicitly Specifies, the next microinstruction to be used, thereby providing the necessary information for sequencing. A sequence of one or more micro-operations designed to control specific operation, such as addition, multiplication is **called a micro program.** The micro-programs for all instructions are stored in the control memory.

A **control variable is a binary digit or bit (0 or 1)** controls the function that specifies a micro-operation. When it is binary 1 state, the corresponding micro- operation is executed while in the opposite binary state, the state of the registers in the system remains unchanged. In a bus-organized system, the control signals that specify the Moro-operations are groups of bits that select the paths in multiplexers, decoders and ALUs. The address where the microinstructions are stored in control memory is generated by micro-program sequencer/micro-program controller in the **micro-programmed control unit.** Thus, the control unit initiates a series of sequential steps of micro-operations. As per the operation, certain micro-operations are to be initiated, while others remain idle at any given time. Grouping the control variables at any given time form a 'string of 1's and 0's, **called a control word**. The control words are stored in the control memory to perform various operations on the components of the system. The control unit whose binary control variables are stored in memory **is called a micro-programmed control unit**. Each word in control memory contains within it a microinstruction. Each microinstruction specifies one or more micro-operations for the system. A sequence of microinstructions constitutes a micro-program. If the operations that are to be perforated by a control unit are fixed, a Read-Only Memory (ROM) can be used as a control memory.

MICRO-PROGRAMMED CONTROL UNIT

A computer that uses a micro-programmed control unit usually has two separate memories - **a main memory and a control memory**. The **main memory** is available to the user for storing their programs. The contents of main memory may change when the data are manipulated and every time the program is changed. The user's program in main memory consists of machine instructions and data, whereas, the **control memory** holds a fixed micro-program that cannot be altered by the occasional user. The micro-program consists of micro-instructions that specify various internal control signals for execution of register micro-operations. Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the micro-operations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

External
Control Word

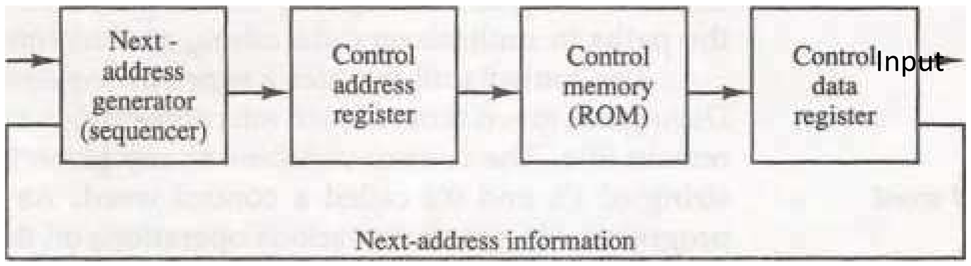


Figure 11. Micro-programmed Control Organization

The general configuration of a micro-programmed control unit is demonstrated in the block diagram above. The **control memory** is assumed to be a ROM, within which all control information is permanently stored. The **control memory address register** specifies the address of the microinstruction, and the **control data register** holds the microinstruction read from memory. The microinstruction contains a control word that specifies one or more micro-operations for the data processor. Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the following place:

1. One next in sequence, or
2. It may be located somewhere else in the control memory
3. The next address may also be a function of external input conditions.

For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction. While the micro- operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction. Thus a microinstruction contains bits for initiating micro- operations in the data processor part and bits that determine the address sequence for the control memory.
The next address generator is sometimes called a **Micro-program *Sequencer*,** as it determines the address sequence that is read from control memory. The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations. The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is sometimes called a ***Pipeline Register***. It allows the execution of the micro-operations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

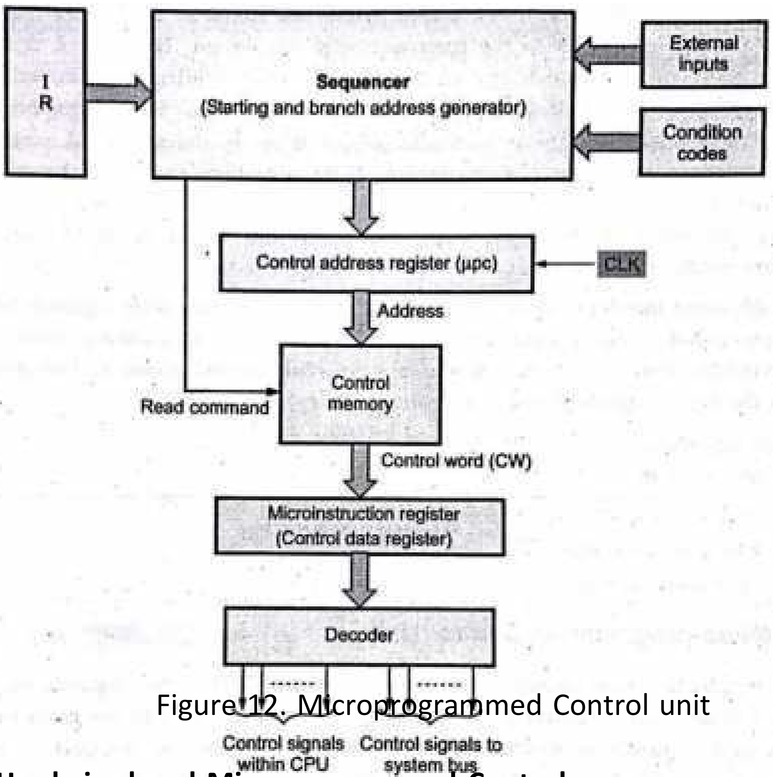
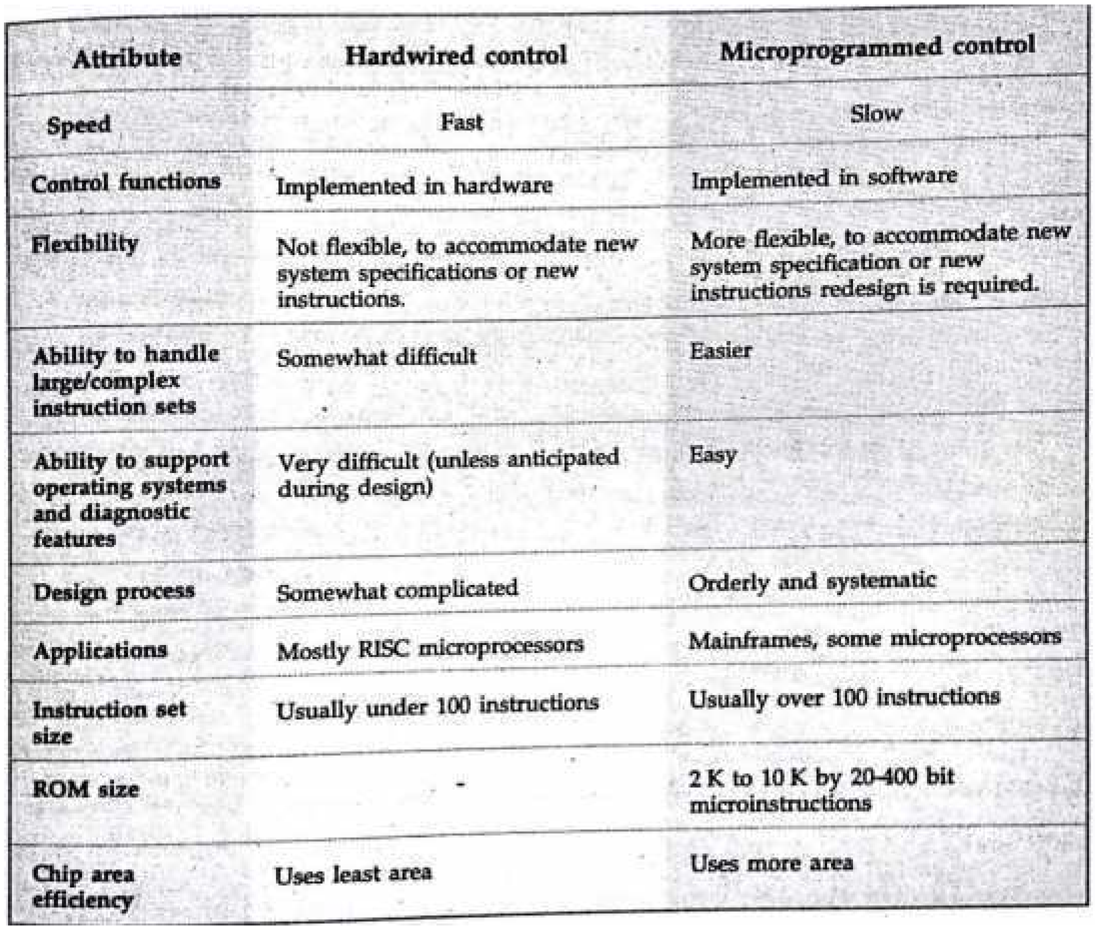

I R
Sequencer
(Starting and branch address generator)
External inputs
Condition codes
Control address register (µpc)
CLK
Address
Read command
Control memory
Control word (CW)
Microinstruction register
(Control data register)
Decoder
Control signals within CPU
Control signals to system bus

Figure 12. Microprogrammed Control unit

Comparison between Hardwired and Microprogrammed Control



Attribute	Hardwired control	Microprogrammed control
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible, to accommodate new system specifications or new instructions.	More flexible, to accommodate new system specification or new instructions redesign is required.
Ability to handle large/complex instruction sets	Somewhat difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult (unless anticipated during design)	Easy
Design process	Somewhat complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instruction set size	Usually under 100 instructions	Usually over 100 instructions
ROM size	-	2 K to 10 K by 20-400 bit microinstructions
Chip area efficiency	Uses least area	Uses more area

Advantages of microprogrammed control

1. It simplifies the design of control unit, Thus it is both, cheaper and less error prone to implement.
2. Control functions are implemented in software rather than hardware.
3. The design process is orderly and systematic.
4. More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.
5. Complex function Such as floating point arithmetic can be realized efficiently

Disadvantages of micro programmed control

1. A micro programmed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.
2. The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Besides these disadvantages, the microprogramming is the dominant technique for implementing control units. However, the most computers based on the Reduced Instruction Set Computer (RISC) architecture concept use hardwired control.

ADDRESS SEQUENCEING

A simple way to structure microinstructions is to assign one bit position to each control signal required in the CPU. However, this scheme has one serious drawback assigning individual bits to each control signal results in long microinstructions, because the number of required signals is usually large. Moreover, only a few bits are used in any given instruction. The solution of this problem is to group the control signals. Grouping technique is used to reduce the number of bits in the microinstruction.

Microinstructions are stored in control memory in groups, with each group specifying a ***Routine.*** Each computer instruction has its own micro-program routine. It is stored in control memory to generate the micro-operations that execute the instruction. The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another. An initial address is loaded into the **control address register** when power is turned on in the computer. This address is usually the address of the first microinstruction/ that activates the instruction fetch routine. The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions. At the end of the fetch routine, the instruction is in the instruction register of the computer.

The control memory next must go through the routine that determines the effective address of the operand. A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers. The effective address computation routine in control memory can be reached through a branch microinstruction. When the effective address computation routine is completed, the address of the operand is available in the memory address register. The next step is to generate the micro-operations that execute the instruction fetched from memory. The microoperation steps to be generated in processor registers depend on the operation code part of the instruction. Each instruction has its own micro-program routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a ***mapping*** **process**.

A mapping procedure is a rule that transforms the instruction code into a control memory address. Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of micro-operations will depend on values of certain status bits in processor registers. After completion of instruction execution, control must return to the fetch routine. This is done by executing an unconditional branch microinstruction to the first address of the fetch routine. In summary, the address sequencing capabilities required in a control memory are: 1. Incrementing of the control addresses register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

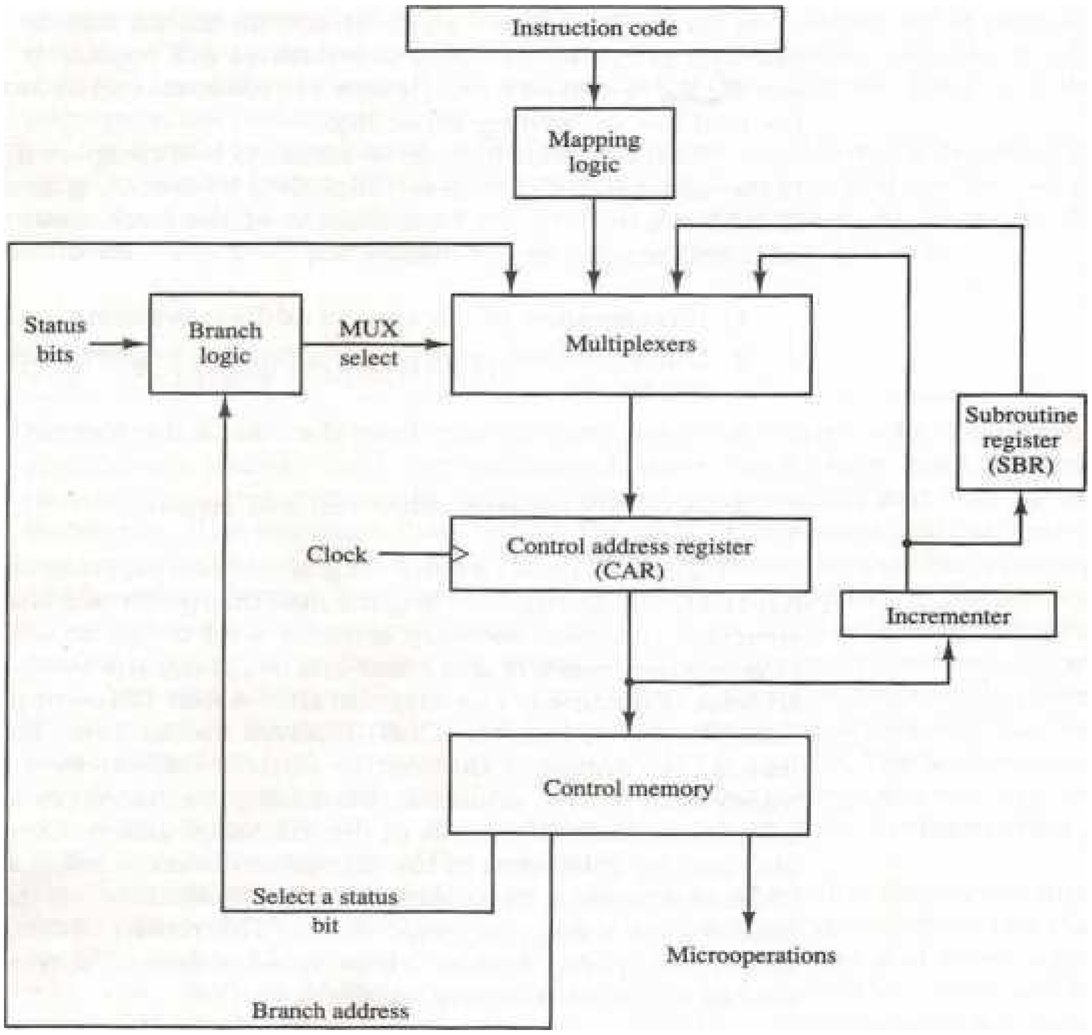


Figure 13. Selection of address for control memory

Figure above shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address. The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence. Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from the subroutine. **Conditional Branching**

The branch logic of above Figure provides **decision-making capabilities** in the control unit.

The **status conditions are special bits** in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions. Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0. The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented. This can be implemented with a multiplexer. An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register. This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1. A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register unconditionally.

Special bits - Conditional branching is achieved by using part of the microinstruction to select a specific status bit in order to determine its condition. **Special bits are used to check Conditions** such as the sign bit of a number, carry- out of an adder; the mode bits in an instruction and input or output status conditions The branch logic checks the status of these bits **(1 or 0)** together with the field in the microinstruction that specifies a branch address and control the conditional branch decisions.

Branch logic - The branch logic hardware checks the status of bits reserved in the microinstruction to take branching decision on the occurrences of specified conditions. One way to implement branch logic hardware is given below. Suppose 8 different parameters are to be checked in a system; It requires 8 status bits. A multiplexer can be used to implement branch logic hardware. In this case, three bits in the microinstruction are used to specify any one of eight status bit conditions. These three bits are used as selection input variables for the multiplexer. If the selected status bit is in the 1 state, the output of the multiplexer goes 1; otherwise it is 0. The 1 output of the multiplexer generates a control signal that transfers the branch address from the microinstruction into the control address register. A 0 output of the multiplexer increments the address register.

When conditional branch microinstruction is executed, the microprogram follows one of two possible paths. The value of status bit decides the selection of path. When an unconditional branch microinstruction is executed, the branch address is loaded from control memory into the control address register. This can be implemented by fixing the value of one status bit at the input of the multiplexer, so it. is always equal to 1.

Mapping of Instruction

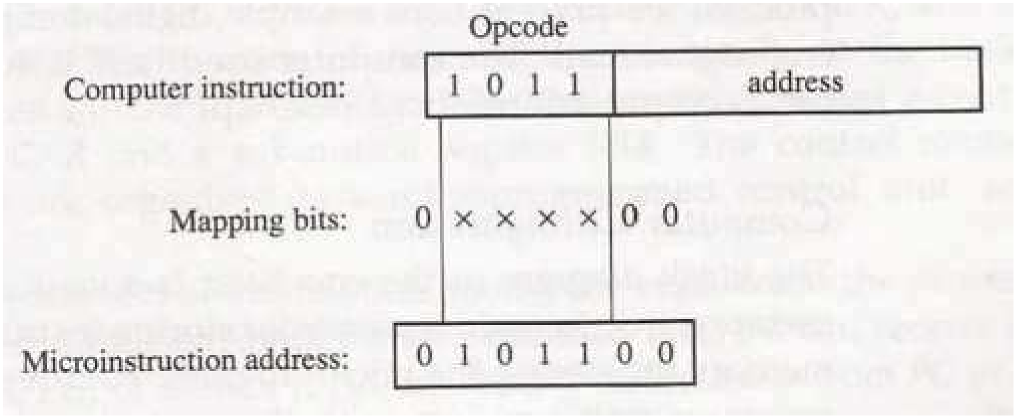


Figure 14. Mapping from instruction code to microinstruction address.

We have seen that the transformation from the instruction code bits to an address in control memory where the microprogram routine is located is **referred to as a mapping process**. A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a micro-program routine for an instruction is located. The status bits for this type of branch are the bits in the operation code part of the instruction.

For example, a computer with a simple instruction format as shown in above has an operation code of four bits which can specify up to 16 distinct instructions. Assume further that the control memory has 128 words, requiring an address of seven bits. For each operation code there exists a micro-program routine in control memory that executes the instruction. One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig. This mapping consists of

1. Placing a 0 in the most significant bit of the address.
2. Transferring the four operation code bits.
3. Clearing the two least significant bits of the control address register.

If the routine needs more than four microinstructions, it can use **addresses** **1000000** through **1111111**. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function. In this configuration, the bits of the instruction specify the address of a mapping ROM. The contents of the mapping ROM give the bits for the control address register. In this way the micro-program routine that executes the instruction can be placed in any desired location in control memory. The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Subroutines

When a certain group of instructions is repeatedly required in a program, then instead of writing it repeatedly, it can be stored separately and can be called in a main program whenever required. This group of instructions is called subroutine. This is the way to use memory efficiently.

In microprogrammed control unit, many microprograms contain identical sections of code. In this case, subroutines can be used for common sections of microcode. For example, generation of the effective address of the operand for an instruction is the task commonly required for the execution of memory reference instructions. The subroutine can be written to perform this task and can be called within many routines to execute the effective address computation.

Subroutine Register

Subroutines can be used in microprograms to use control memory efficiently. When the microprogram (main routine) needs subroutine, the address of subroutine is to be loaded into the control address register. This transfers program control from main routine to the subroutine. However, after execution of subroutine, the control should be transferred back to the main routine. So, before transferring control from main routine to the subroutine, it is necessary to store return address. This may be accomplished by storing the incremented output from the control address register (return address) into a subroutine register and then branching to the beginning of the subroutine. Upon completion of subroutine execution, the return address is restored into the control address register from subroutine register. Thus, the subroutine register stores the return address during a subroutine call and restores it during a subroutine return. The stack (registers organized in LIFO fashion) can be used for the execution of subroutines.

SEQUENCING AND EXECUTION OF MICRO INSTRUCTION.

Microinstruction Format

Figure 15 shows the microinstruction format for the control memory. As shown in Fig 15 the microinstruction includes four fields.

1. F1, F2, F3: These are micro-operation fields. Each field is of three bits. They specify micro-operations for the Computer.
2. CD: This two-bit field selects status bit conditions for branching operation. The condition includes zero value in AC, sign bit of AC equal to 1 or 0, etc.
3. BR: This 2-bit field specifies the type of branch to be used. Branch types include unconditional branch, branch if zero, and branch if negative and so on.
4. AD: This is an address field which contains a branch address. This field is of seven bits since control memory has 128 words. (128 = 27).

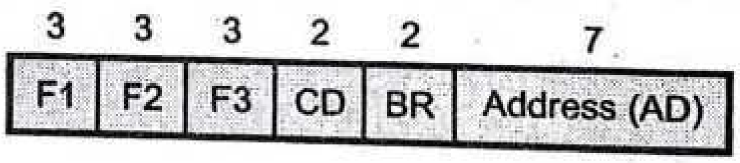
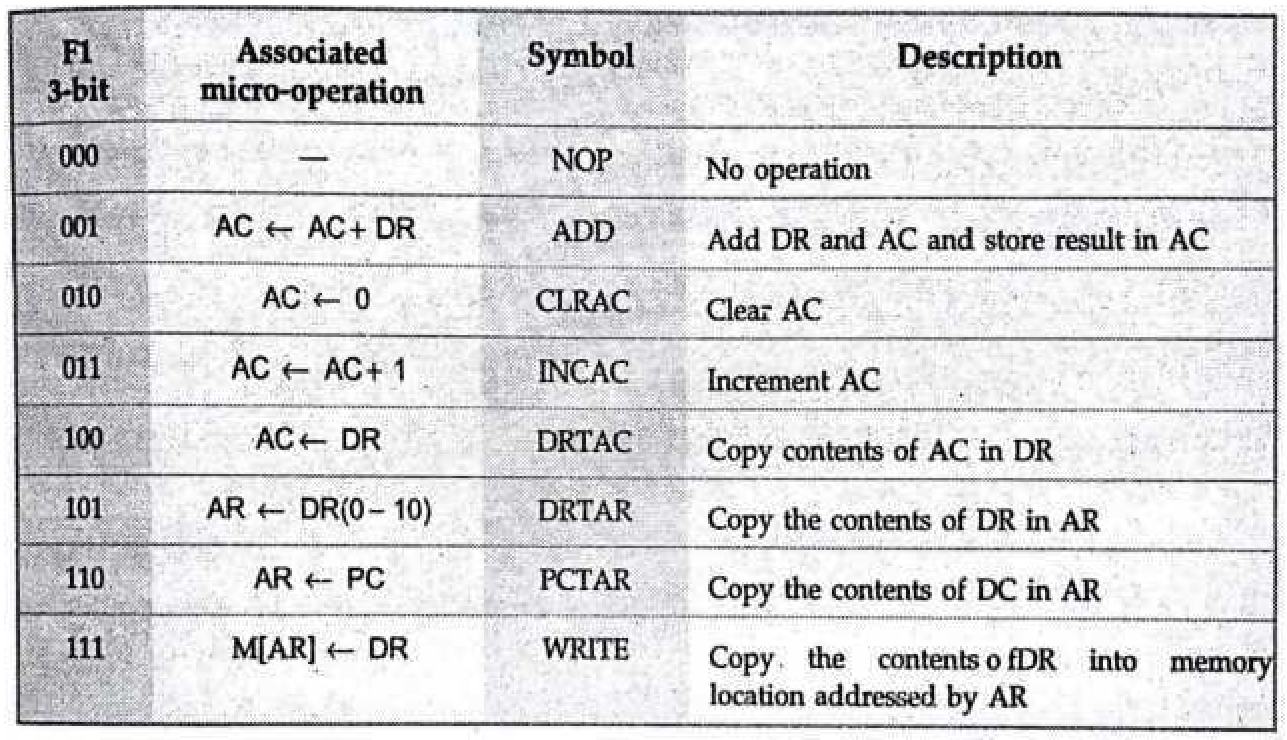


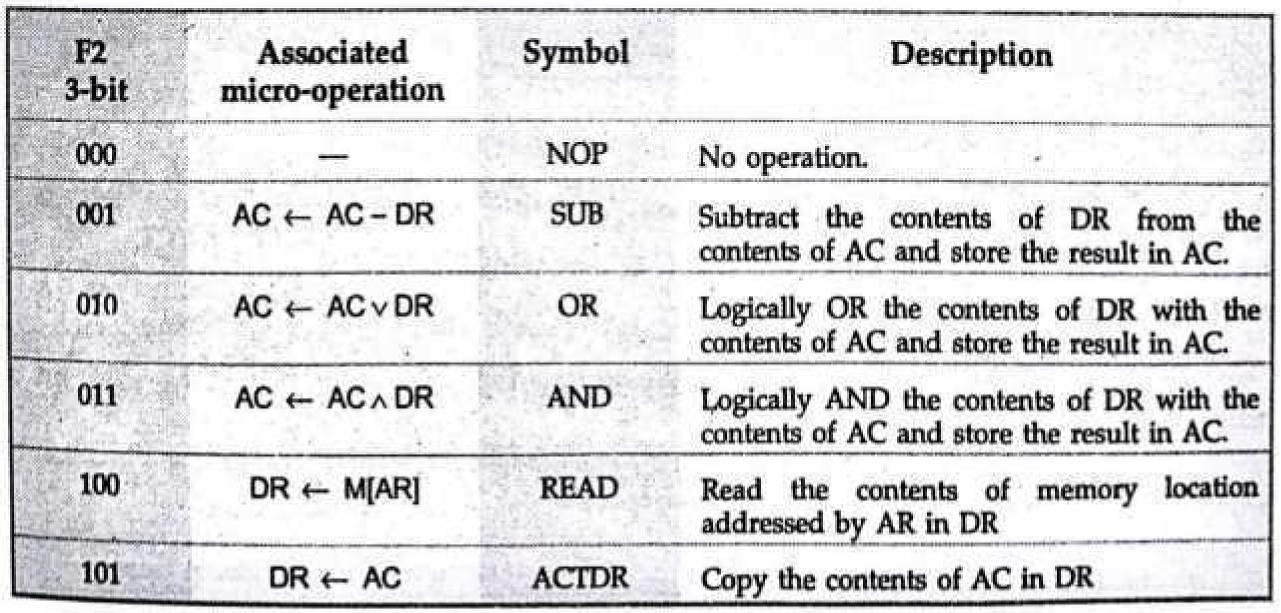
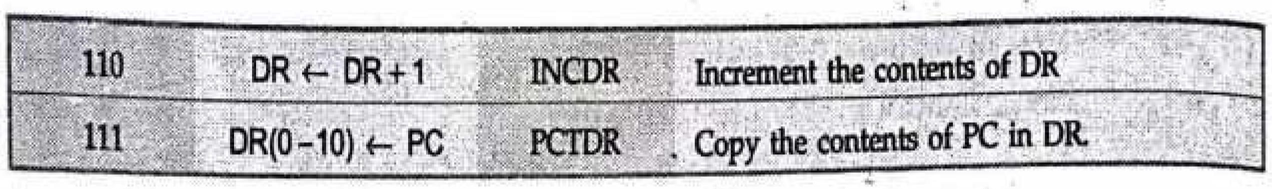
Figure 15 Microinstruction format

The micro-operations are subdivided into three fields as F1, F2, F3, each is of 3-bits. The three bits in each field are encoded to specify seven distinct micro-operations.

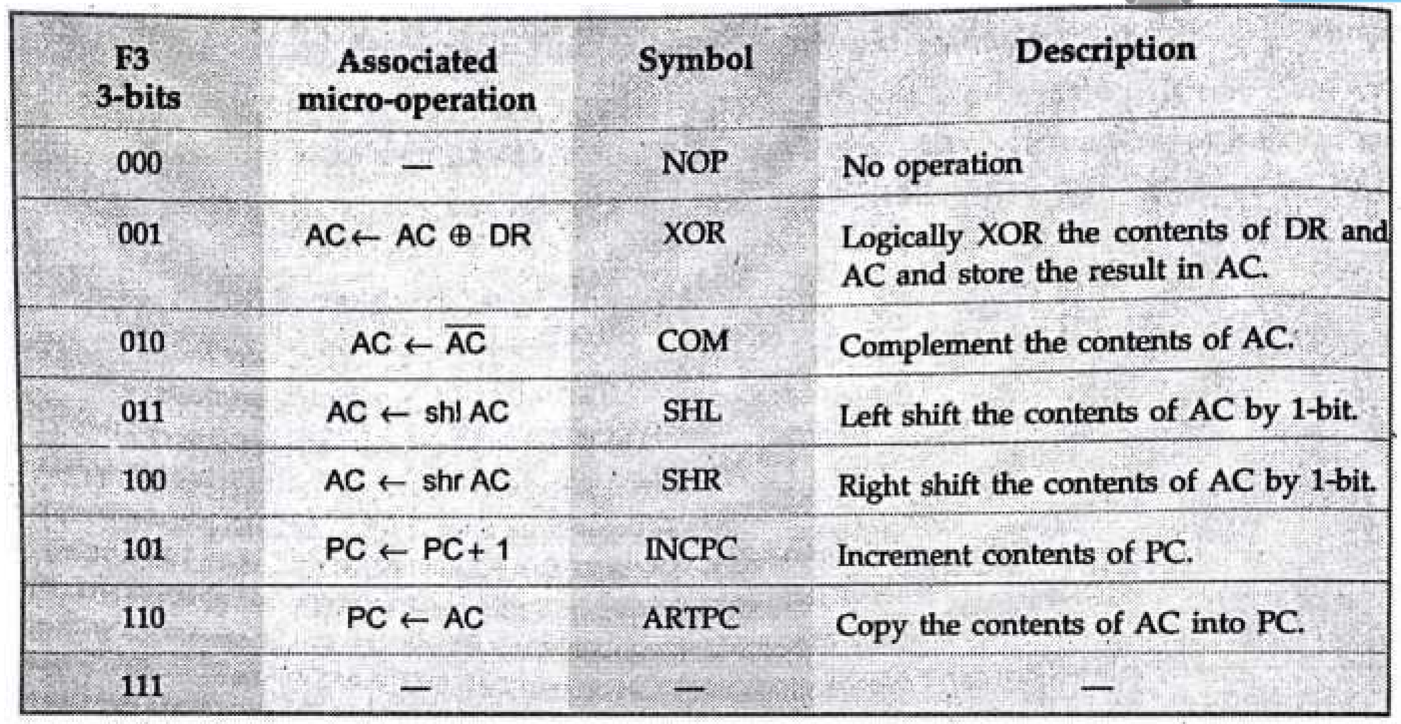
Thus there are 21 micro-operations. A microinstruction includes three micro- operations, one from each field. So, no more than three micro-operations can be selected for a microinstruction. If the microinstruction needs micro-operations less than three, one or more of the micro—operation fields will be filled by a binary code
000 for no operation.



F1 3-bit	Associated micro-operation	Symbol	Description
000	—	NOP	No operation
001	AC ← AC + DR	ADD	Add DR and AC and store result in AC
010	AC ← 0	CLRAC	Clear AC
011	AC ← AC + 1	INCAC	Increment AC
100	AC ← DR	DRTAC	Copy contents of AC in DR
101	AR ← DR(0 – 10)	DRTAR	Copy the contents of DR in AR
110	AR ← PC	PCTAR	Copy the contents of DC in AR
111	M[AR] ← DR	WRITE	Copy the contents o fDR into memory location addressed by AR

F2 3-bit	Associated micro-operation	Symbol	Description
000	—	NOP	No operation.
001	AC ← AC – DR	SUB	Subtract the contents of DR from the contents of AC and store the result in AC.
010	AC ← AC ∨ DR	OR	Logically OR the contents of DR with the contents of AC and store the result in AC.
011	AC ← AC ∧ DR	AND	Logically AND the contents of DR with the contents of AC and store the result in AC.
100	DR ← M[AR]	READ	Read the contents of memory location addressed by AR in DR
101	DR ← AC	ACTDR	Copy the contents of AC in DR
110	DR ← DR + 1	INCDR	Increment the contents of DR
111	DR(0 – 10) ← PC	PCTDR	Copy the contents of PC in DR



F3 3-bits	Associated micro-operation	Symbol	Description
000	—	NOP	No operation
001	AC ← AC ⊕ DR	XOR	Logically XOR the contents of DR and AC and store the result in AC.
010	$AC \leftarrow \overline{AC}$	COM	Complement the contents of AC.
011	AC ← shl AC	SHL	Left shift the contents of AC by 1-bit.
100	AC ← shr AC	SHR	Right shift the contents of AC by 1-bit.
101	PC ← PC + 1	INCPC	Increment contents of PC.
110	PC ← AC	ARTPC	Copy the contents of AC into PC.
111	—	—	—

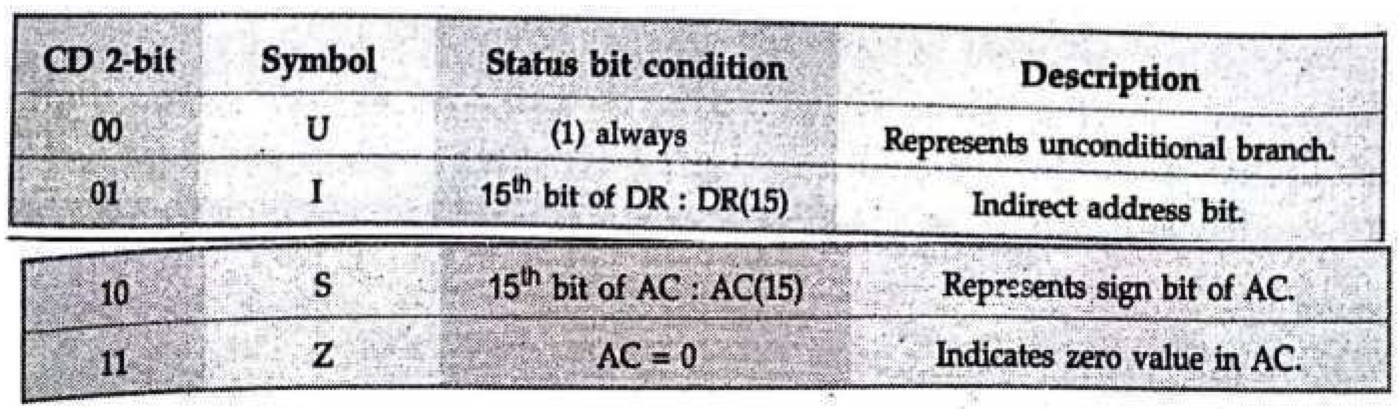
Table 1 Micro-instructions with their binary code, micro-operation and symbol

From above table, we can observe

1. Each micro-operation is defined with a register transfer statement.
2. A symbol is assigned to each macro-operation. This is useful in writing symbolic program.
3. All transfer-type micro-operations symbols use five letters. The first two letters indicate the source register, the third letter is always T and the last two letters indicate the destination register.

Condition Field

The two-bits in the condition (CD) field are encoded to specify four status bit conditions as listed m Table 2.



CD 2-bit	Symbol	Status bit condition	Description
00	U	(1) always	Represents unconditional branch.
01	I	15th bit of DR : DR(15)	Indirect address bit.
10	S	15th bit of AC : AC(15)	Represents sign bit of AC.
11	Z	AC = 0	Indicates zero value in AC.

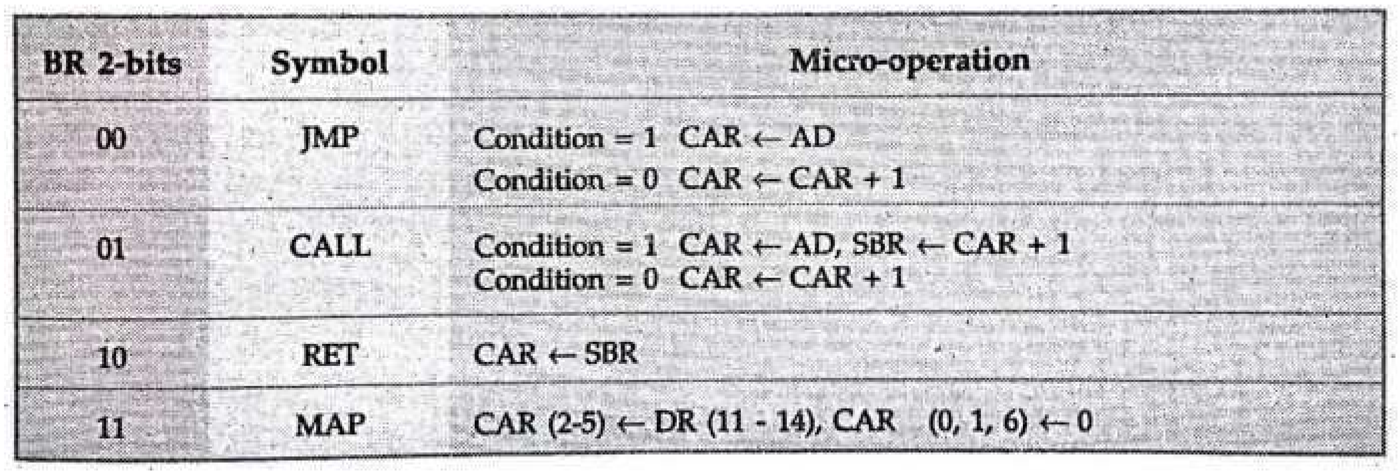
Table 2 Condition field with binary code, condition and symbol

From Table 2. we can observe,

1. The first condition is always a 1, so that a reference to 'CD = 00' (symbol U) will always find the condition to be true. When this condition is used with the branch field, it provides an unconditional branch operation.
2. The indirect bit I is available from bit 15 of DR after reading an instruction from memor y.
3. The bit 15 (sign-bit) of an accumulator provides the next status bit.
4. When the content of an accumulator becomes zero, the binary variable 2 becomes, equal to 1. The symbols U, I, S and Z are used for the four status bits while writing microprograms in symbolic form.

Branch Field and Address Field

The two bit branch field is used with 7- bit address field to obtain the address of the next microinstruction. The two-bits in the branch .field are encoded to specify four branches as shown in Table 3.



BR 2-bits	Symbol	Micro-operation

00	JMP	Condition = 1 CAR ← AD
		Condition = 0 CAR ← CAR + 1

01	CALL	Condition = 1 CAR ← AD, SBR ← CAR + 1
		Condition = 0 CAR ← CAR + 1

10	RET	CAR ← SBR
11	MAP	CAR (2-5) ← DR (11 - 14), CAR (0, 1, 6) ← 0

Table 3 Branch field with binary code. symbol and function

From Table 3, we can observe,

1. When BR = 00, the control performs a jump operation.
2. When BR = 01, the control performs a call to subroutine operation.

MICRO-PROGRAM SEQUENCER

The subunit of the microprogrammed control unit which presents an address to the control memory is **called microprogram sequencer**. The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.

The Fig. 16 shows the block diagram of commercial microprogram sequencer. It consists of a multiplexer that selects an address from four sources and routes it into a control address register. The output from CAR provides the address for the control memory. The contents of CAR are incremented and applied to the multiplexer and to the stack register file. The register selected in the stack is determined by the stack pointer. Inputs 12, 11, I0 and T derived from the CD and BR fields of microinstruction specify the operation for the sequencer. They specify the input source to the multiplexer also generate push and pop signals required for stack operation.

The stack pointer is a , three-bit register and it gives the address of stack register file consists of (2^3 = 8) eight registers Initially, the stack pointer is cleared and is said to point at address 0 in the stack. Using push operation it is possible to write data into the stack at the address specified by the stack pointer. After data is written, stack pointer is incremented by one to get ready for the next push operations.

In pop operation stack pointer is decremented by one and then the contents of the register specified by the new value' of stack pointer are read. With this mechanism it is possible to implement subroutine calls. During subroutine call the incremented address (the address of the next instruction) is stored in the stack. This address also called return address is transferred back into CAR with a subroutine return operation.

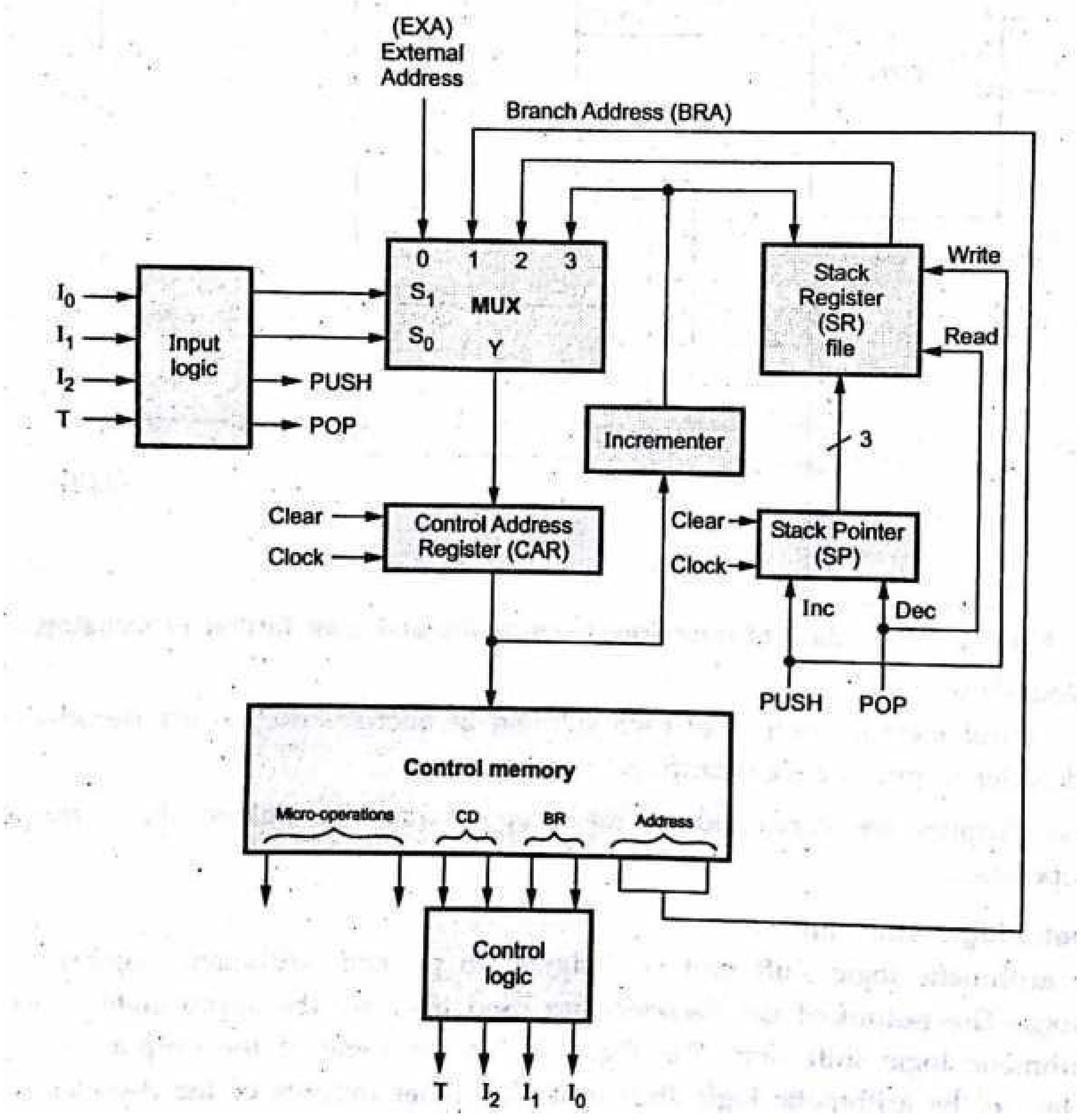
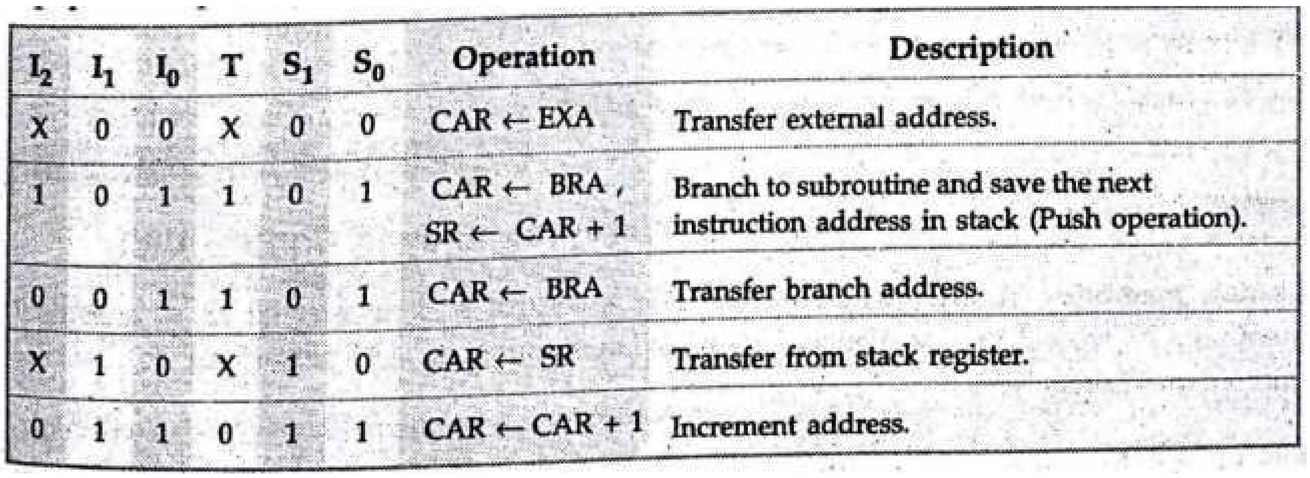


Figure 16 Typical microprogram sequencer organization

The Table 4 gives the function table for microprogram sequencer. When **S1S0 = 00**, an external address is transferred to CAR. The transfer from address field of microinstruction occurs when **S1S0 = 01** and T = 1. When **S1S0 = 10**, stack register contents are transferred to CAR and when **S1S0 = 11**, incremented contents of CAR are transferred to the CAR.

A call to subroutine is executed by activating push signal when **S1S0 = 01**. This causes a push stack operation and a branch to the address specified by microinstruction. The return from subroutine is executed by activating pop signal when **S1S0 = 10.** This cause a pop-stack operation and a branch to the address stored on top of the stack



I_2	I_1	I_0	T	S_1	S_0	Operation	Description
X	0	0	X	0	0	$CAR \leftarrow EXA$	Transfer external address.
1	0	1	1	0	1	$CAR \leftarrow BRA$, $SR \leftarrow CAR + 1$	Branch to subroutine and save the next instruction address in stack (Push operation).
0	0	1	1	0	1	$CAR \leftarrow BRA$	Transfer branch address.
X	1	0	X	1	0	$CAR \leftarrow SR$	Transfer from stack register.
0	1	1	0	1	1	$CAR \leftarrow CAR + 1$	Increment address.

Table 4 Function table of micro program sequencer