Unit - II

CPU and Micro Programmed Control

Objective:

- To familiarize the concept of Addressing Modes, Control memory, Hard wired control, Micro programmed control.

Syllabus:

Central Processing unit: Introduction, instruction formats, addressing modes. Control Memory, address sequencing, design of control unit - hard wired control, micro programmed control.

Learning Outcomes:

At the end of the unit student will be able to:

1. Differentiate micro-programmed and hard-wired control units

Learning Material

INSTRUCTION FORMATS

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into 3 fields with each field consisting of a group of bits.
    a. **Operation code** field that specifies the operation to be performed
    b. **Address** field that designates a memory address or processor register where the operand is present
    c. **Mode** field that specifies the way the operand or the effective address is to be determined.
- Other special fields are sometimes employed under certain circumstances.
- Instruction formats are concerned with the address fields in an instruction.
- Operations specified by computer instructions are executed on some data stored in memory or registers.
- Operands residing in memory are specified by their memory address.
- Operands residing in registers are specified with a register address.
- A CPU with 16 processor registers labeled R_0 through R_{15} will have a register address field of 4 bits.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

i. **Single accumulator organization**

- All operations are performed in combination with accumulator. It requires only one address field.
- ADD X, where X is the address of the operand. The ADD instruction results in the operation
- AC←AC+M[X]

ii. **General register organization**

- It requires two or three register address fields

  ADD R1,R2,R3 resulting in the operation R1←R2+R3.
- The number of address fields in the instruction can be reduced from three to two if the destination register is one of the source registers.

  ADD R1,R2 resulting in operation R1←R1+R2. These instructions need two address fields to specify the source and the destination.

iii. **Stack organization**

- It has PUSH and POP instructions which require no address field.
- Consider the instruction ADD, this operation has the effect of popping the top two elements from the stack, adding the numbers, and pushing the sum into the stack.

Let us see the assembly language programs in different instruction formats that evaluates X=(A+B)*(C+D) as follows.

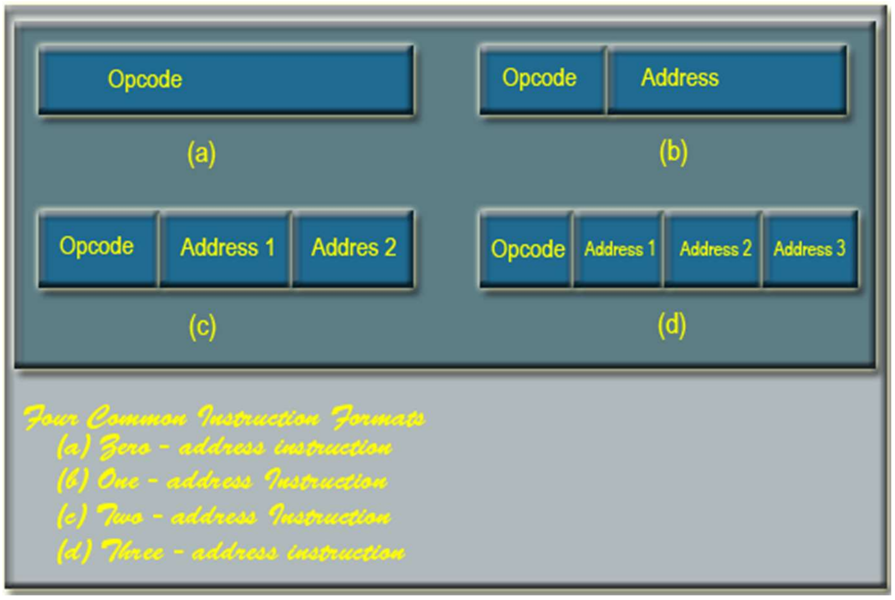

Opcode
Opcode
Address
(a)
(b)
Opcode
Address 1
Addres 2
Opcode
Address 1
Address 2
Address 3
(c)
(d)
Four Common Instruction Formats
(a) Zero - address instruction
(b) One - address Instruction
(c) Two - address Instruction
(d) Three - address instruction

Figure 1: Four common Instruction formats

THREE ADDRESS INSTRUCTIONS

- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

```
ADD   R1,A,B          R1←M[A]+M[B]
ADD   R2,C,D          R2←M[C]+M[D]
MUL   X,R1,R2         M[X]←R1*R2
```

Merit: it results in short programs when evaluating arithmetic expressions.

Demerit: The binary-coded instructions require too many bits to specify three addresses.

TWO ADDRESS INSTRUCTIONS

- Each address field can specify either a processor register or a memory word.

```
MOV   R1,A            R1←M[A]
ADD   R1,B            R1←R1+M[B]
MOV   R2,C            R2←M[C]
ADD   R2,D            R2←R2+M[D]
MUL   R1,R2           R1←R1*R2
MOV   X,R1            M[X]←R1
```

The MOV instruction moves or transfers the operands to and from memory and processor registers.

ONE ADDRESS INSTRUCTIONS

- One address instructions use an implied accumulator (AC) register for all data manipulations.

```
LOAD  A          AC←M[A]
ADD   B          AC←AC+M[B]
STORE T          M[T]←AC
LOAD  C          AC←M[C]
ADD   D          AC←AC+M[D]
MUL   T          AC←AC*M[T]
STORE X          M[X]←AC
```

- All operations are done between accumulator register and a memory operand. T is the address of the temporary memory location required for storing the intermediate result.

ZERO ADDRESS INSTRUCTIONS

- A stack organized computer does not use an address field in the instructions. The PUSH and POP instructions, however need an address field to specify the operand that communicates with the stack.

```
PUSH  A     TOS←A
PUSH  B     TOS←B
```

```
ADD         TOS←(A+B)
PUSH  C     TOS←C
PUSH  D     TOS←D
ADD         TOS←(C+D)
MUL         TOS←(A+B)* (C+D)
POP   X     M[X]←TOS
```

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish (Postfix) notation.

ADDRESSING MODES

- In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction.
- Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.
- An example of an instruction format with a distinct addressing mode field is shown in the below figure 2.

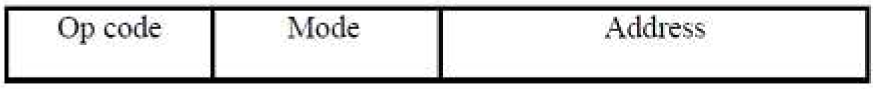


Figure 2: Instruction format with mode field

- The operation code specifies the operation to be performed.
- The mode field is used to locate the operands needed for the operation.
- There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register.
- The instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.
- There are two modes that need no address field at all. These are the implied and immediate modes.

1. Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction.

- **For example**, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

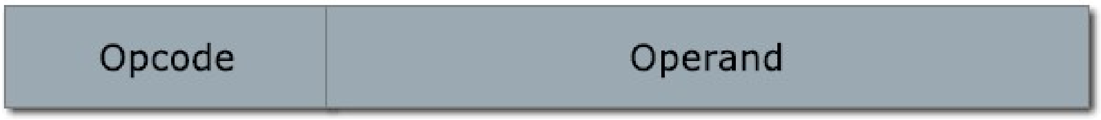
- In fact, all register reference instructions that use an accumulator are implied-mode instructions.

Example: Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

CMA, CME, CLE, CLA

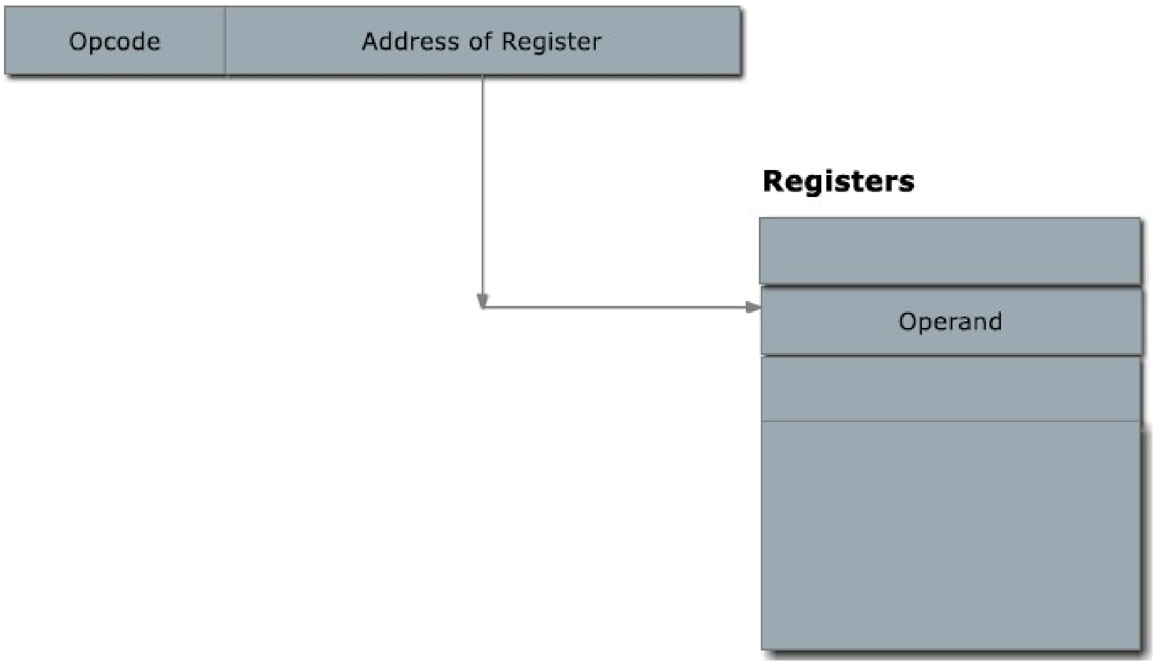
2. Immediate Mode: In this mode the operand is specified in the address part of the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. Immediate-mode instructions are useful for initializing registers to a constant value.

LD #20



3. Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

LD R

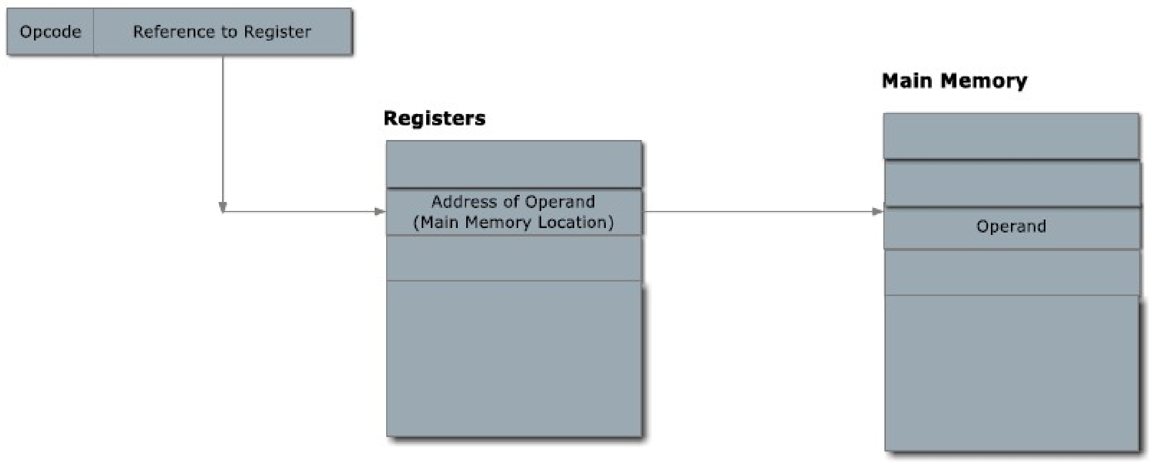


4. Register Indirect Mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.

- Before using a register indirect mode instruction, the programmer must ensure that the memory address for the operand is placed in the processor register with a previous instruction.
- A reference to the register is then equivalent to specifying a memory address.
- The advantage of a register indirect mode instruction is that the address field of the instruction sues fewer bits to select a register than would have been required to specify a memory address directly.

LD (R) OR **LD** @R



5. Auto increment or Auto decrement Mode:

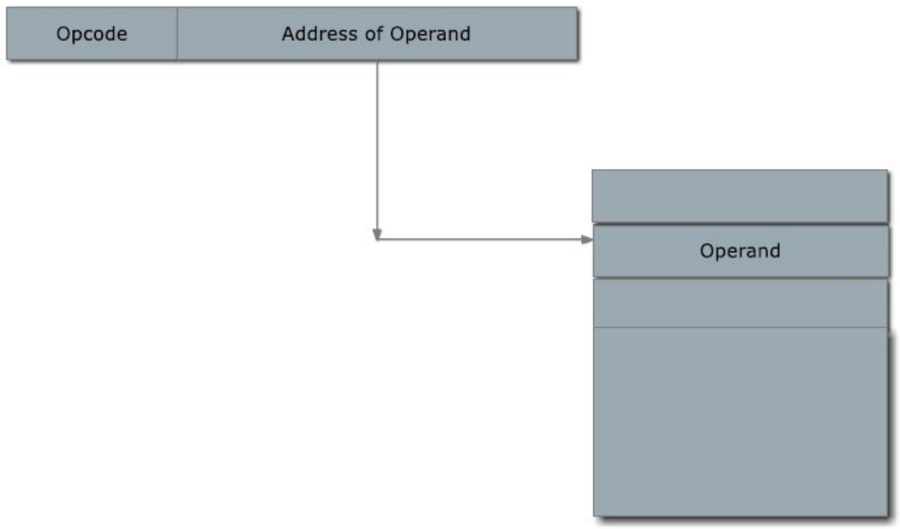
- This is similar to the register indirect mode except that the register value is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

LD (R) +	Auto increment
LD (R) -	Auto decrement

6. Direct Address Mode:

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory.
- In a branch-type instruction the address field specifies the actual branch address.

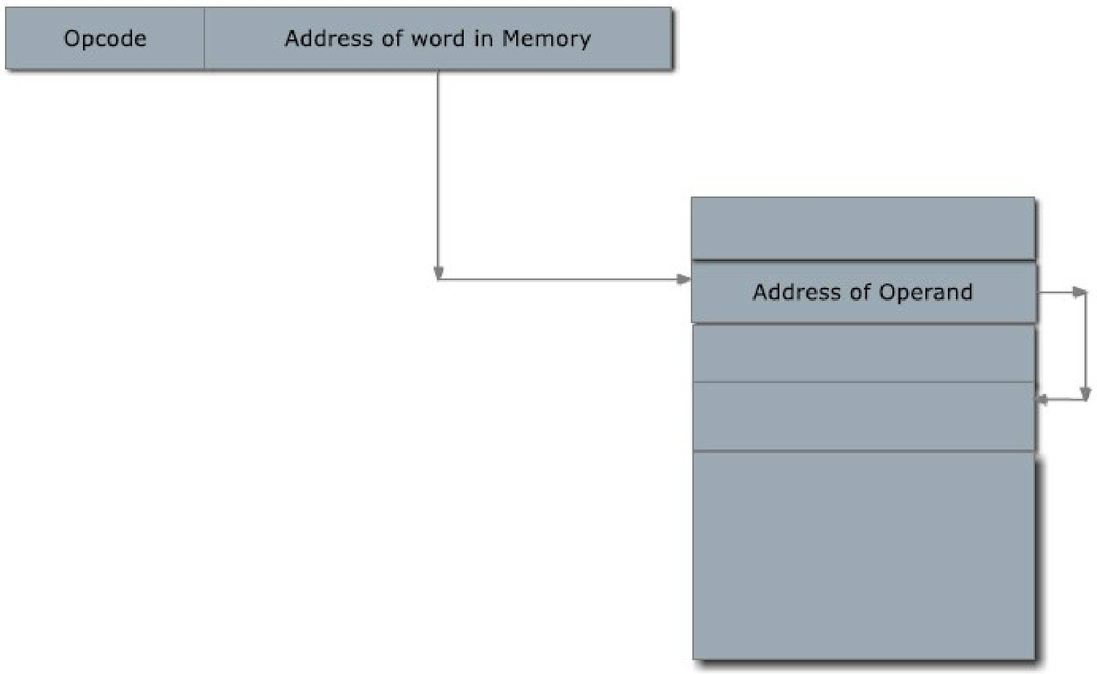
LD 10



7. Indirect Address Mode:

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

  LD (10)



8. Relative Address Mode:

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

- The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

  LD \$5

9. Indexed Addressing Mode:

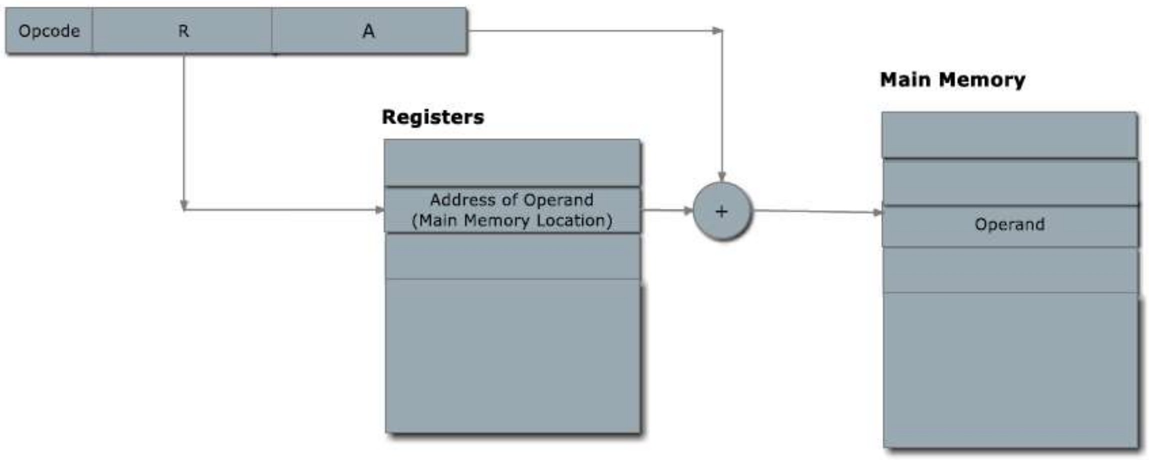
- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.

  LD 5(IR)

10. Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

  LD 5(BR)



Displacement addressing mode (Relative Addressing, Base Register Addressing, Indexed Addressing)

Numerical Example of Addressing Modes:

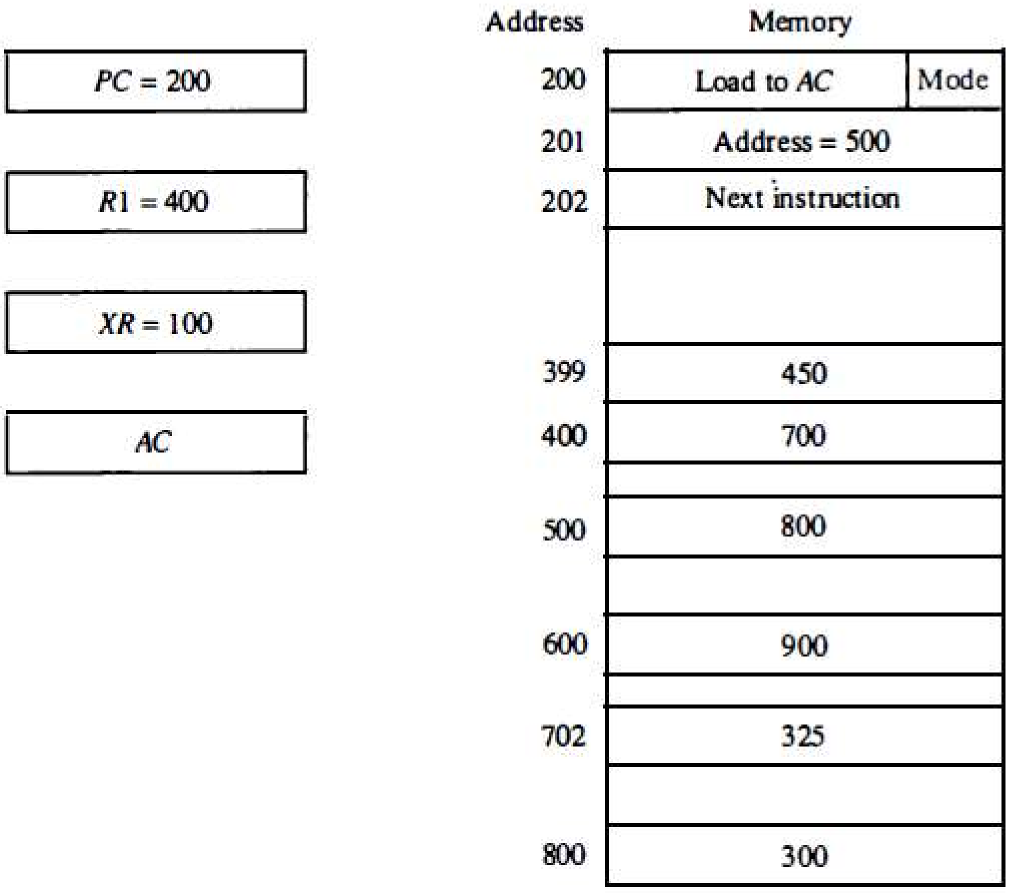


Figure 3: Numerical example for addressing modes

Table 1: Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Description on Addressing Modes

Addressing Mode	**Example**	**Description**
Implied / Implicit	CMA, CME, CLA, CLE	Operands are Specified Implicitly in the instruction
Immediate	LDAC #20	Here operands is specified in the instruction itself Here data value 20 is loaded into AC
Direct	LDAC 5	Here 5 refer address of operand. Here instruction reads operand from address location 5 and load into AC
Indirect	LDAC @10 (OR) LDAC (10)	Here first retrieves the content of memory location 10 say 5, then CPU goes to address location 5, read data and load into AC.
Register	LDAC R	Here operands are in Registers. If the register has operand 20, this instruction Loads operand 20 into AC
Register Indirect	LDAC @R LDAC (R)	If the register has address location 5, then CPU goes to address location 5, read data from address location 5, and then load into AC
Relative	LADC $5	Assume that Next instruction address is (PC) 3, then add it to the address part of instruction 5 i.e. 5+3→ 8 From the address location 8 the CPU read data and load into AC.
Indexed	LDAC 5(XR)	If the index register XR contains value 10, then this instruction read data from address location 5+10. i.e. address location 15 and load into AC

Control Memory

- The function of the control unit in a digital computer is to initiate sequences of microoperations.
- Two major types of Control Unit.
   i. *Hardwired Control Unit:* The control logic is implemented with gates, Flip-flops, decoders, and other digital circuits, then that control unit is said to be Hardwired Control Unit.
   ii. *Microprogrammed Control Unit:* The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations is referred as Microprogrammed Control Unit.
- *Control Word:* The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- *Microinstruction:* The microinstruction specifies one or more microoperations.
- *Microprogram:* A sequence of microinstruction constitutes a microprogram.
   i. *Static microprogramming:* Here Control Memory = ROM
      - The content of the words in ROM are fixed and cannot be altered by simple programming, since no writing capability is available in the ROM.
      - Control words in ROM are made permanent during the hardware production of the unit.
   ii. *Dynamic microprogramming:* Here Control Memory = RAM
      - Microprogram is loaded initially from an auxiliary memory such as a magnetic disk.
      - Control units that use dynamic microprogramming employ a writable control memory. This type of memory can be used for writing (to change the microprogram) but is used mostly for reading.
- A memory that is part of a control unit is referred to as a *control memory*.
- A computer that employs a microprogrammed control unit will have two separate memories: a main memory and a control memory.
- The *main memory* is available to the user for storing the programs. The contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data.
- The *control memory* holds a fixed microprogram that cannot be altered by the occasional user.
- The general configuration of a microprogrammed control unit is demonstrated in the block diagram of the following figure 4.

- The control memory is assumed to be a ROM, within which all control information is permanently stored.

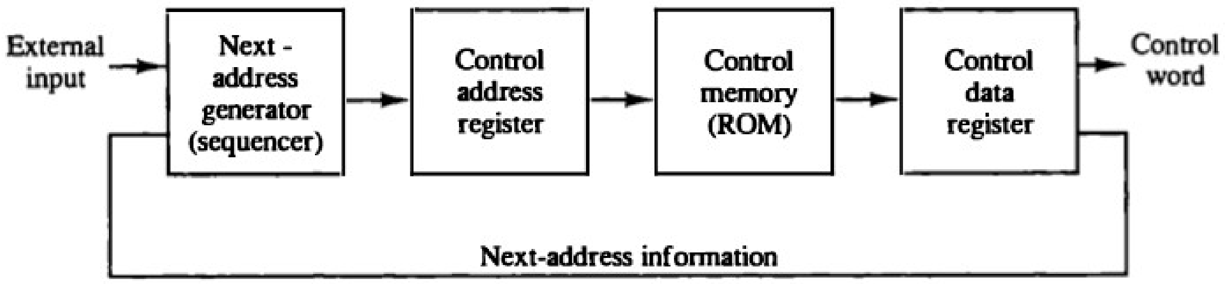


Figure 4: Microprogrammed control organization

- *Control address register:* The control address register specifies the address of the microinstruction.
- *Control data register:* The control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations 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 one next in sequence, or it may be located somewhere else in the control memory. 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.
- The next address may also be a function of external input conditions. While the microoperations 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.
- The next address generator is sometimes called a microprogram *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 microprogram 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 microoperations 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.

- The other two components: the *sequence*r and the *control memory* are combinational circuits and do not need a clock.
- The main advantage of the microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.
- The hardware configuration should not be changed for different operations, the only thing that must be changed is the microprogram residing in control memory.

Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a *routine*.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- 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 control memory next must go through the routine that determines the *effective address* of the operand.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- The next step is to generate the microoperations that execute the instruction fetched from 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.
- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- When the execution of the instruction is completed, control must return to the fetch routine. This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.
- The address sequencing capabilities required in a control memory are:
    1. Incrementing of the control address 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.

- The following figure 5 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

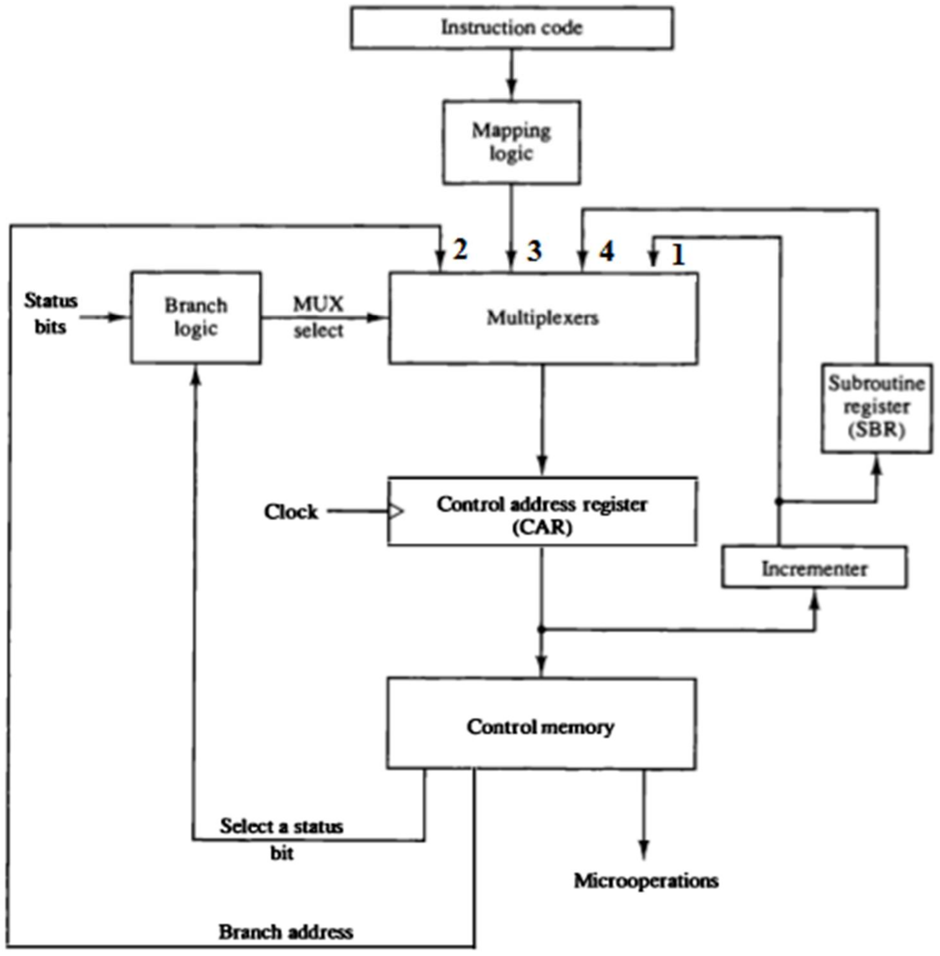


Figure 5: Selection of address for control memory

- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- The above diagram shows four different paths from which the control address register (CAR) receives the address.
  1. Incrementer
  2. Branch address from control memory
  3. Mapping Logic
  4. SBR : Subroutine Register
- In the above diagram Multiplexer receives 4 inputs as follows.
  1. CAR Increment

2. JMP/CALL
3. Mapping
4. Subroutine Return

- Subroutine Register (SBR): Return Address cannot be stored in ROM. So return Address for a subroutine is stored in SBR.

Conditional Branching

- The branch logic of figure 5 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 Control the conditional branch decisions generated in the Branch Logic.
- The Branch LogicTest the specified condition and Branch (jump) to the indicated address if the condition is met; otherwise, the control address register is just incremented.
- 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.

Mapping of Instruction

- A special type of branch exists when a microinstruction specifies a branch to the *first word* in control memory where a microprogram 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 figure 6 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 microprogram 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 figure 6.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

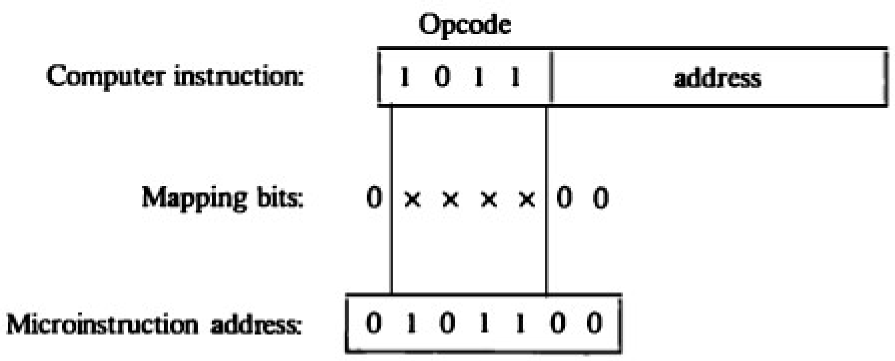


Figure 6: Mapping from instruction code to microinstruction address

Subroutines

- Subroutines are programs that are used by other routines to accomplish a particular task. A subroutine can be called from any point within the main body of the microprogram.
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return. This may be accomplished by placing the incremented output from the control address register into a subroutine register and branching to the beginning of the subroutine.
- The subroutine register can then become the source for transferring the address for the return to the main routine.

Microinstruction Format

- The microinstruction format for the control memory is shown in following figure 8. The 20 bits of the microinstruction are divided into four functional parts.
- The three fields F1, F2, and F3 specify microoperations for the computer.
- The CD field selects status bit conditions.
- The BR field specifies the type of branch to be used.
- The AD field contains a branch address.
- The address field is seven bits wide, since the control memory has 128 = 2^7 words.

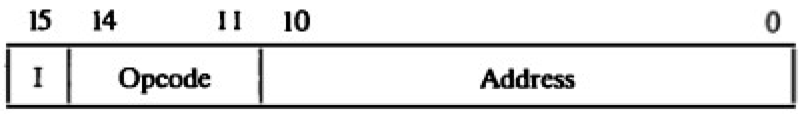
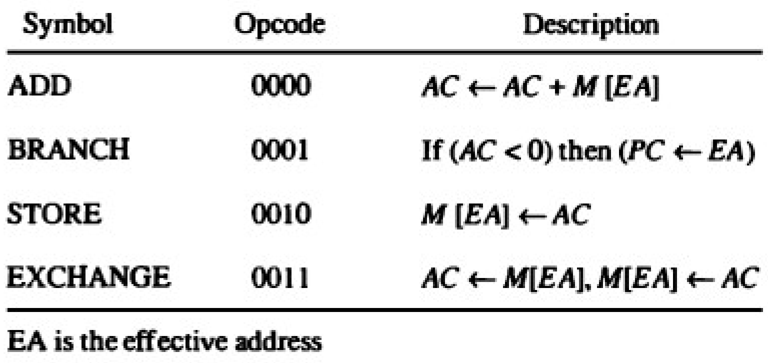


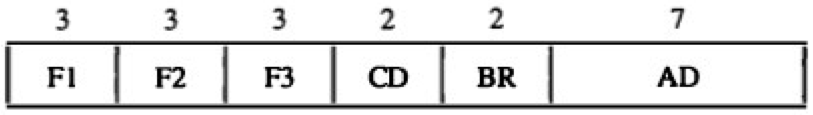
Figure 7: Instruction format

Table 2: Four computer instructions



Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address



F1, F2, F3: Microoperation fields

CD: Condition for branching

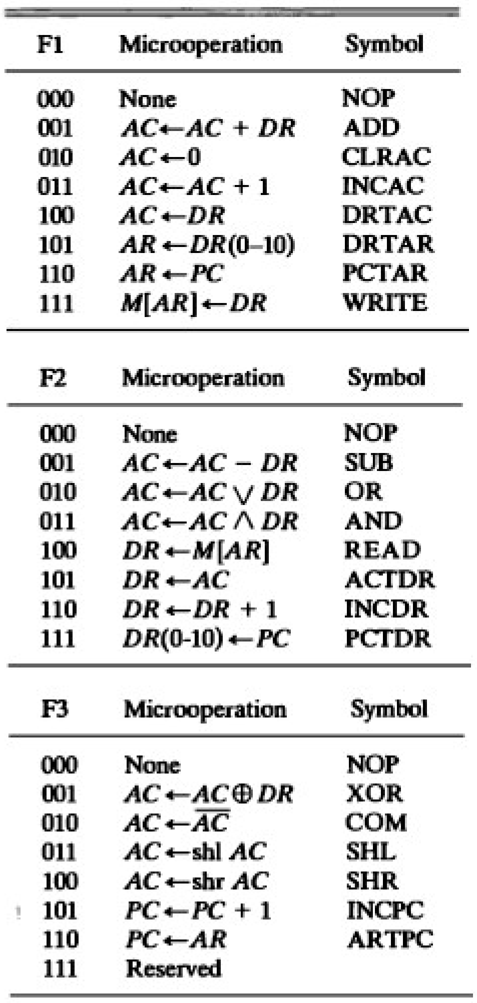
BR: Branch field

AD: Address field

Figure 8: Microinstruction code format (20 bits)

- The microoperations are subdivided into three fields of three bits each.
- The three bits in each field are encoded to specify seven distinct microoperations as listed in Table 2. This gives a total of 21 microoperations.
- No more than three microoperations can be chosen for a microinstruction, one from each field. If fewer than three microoperations are used, one or more of the fields will use the binary code 000 for no operation.
- Each microoperation in Table 3 is defined with a register transfer statement and is assigned a symbol for use in a symbolic microprogram.
- All transfer-type microoperations symbols use five letters. The first two letters designate the source register, the third letter is always a *T*, and the last two letters designate the destination register.

Table 3: Symbols and Binary Code for Microinstruction Fields



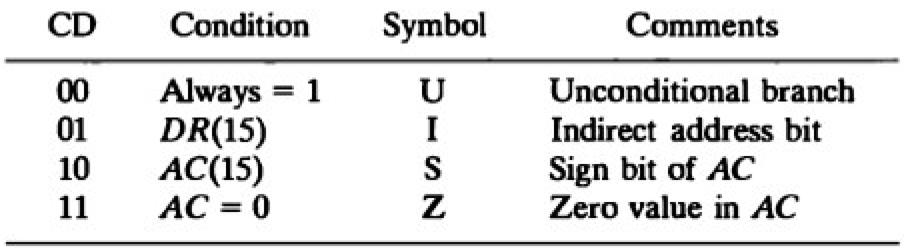
F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- For example, the microoperation that specifies the transfer AC ←DR (F1 = 100) has the symbol DRTAC, which stands for a transfer from DR to AC.
- The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 4.

Table 4: Symbols and Binary Code for CD field



CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

- The first condition is always a 1, so that a reference to CD = 00 (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation.
- The indirect bit *I* is available from bit 15 of DR after an instruction is read from memory.
- The sign bit of AC provides the next status bit.
- The zero value, symbolized by *Z*, is a binary variable whose value is equal to 1 if all the bits in AC are equal to zero.
- We will use the symbols U, I, S, and Z for the four status bits when we write microprograms in symbolic form.

Table 5: Symbols and Binary Code for BR field

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD, SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14), CAR(0,1,6) \leftarrow 0$

- The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction.
- As shown in Table 5, when BR = 00, the control performs a jump (JMP) operation (which is similar to a branch), and when BR = 01, it performs a call to subroutine (CALL) operation. The two operations are identical except that a call microinstruction stores the return address in the subroutine register SBR.
- The jump and call operations depend on the value of the CD field. If the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to the control address register CAR. Otherwise, CAR is incremented by 1.
- The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR.
- The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11. This mapping is as depicted in figure 6.
- The bits of the operation code are in DR(ll-14) after an instruction is read from memory. Note that the last two conditions in the BR field are independent of the values in the CD and AD fields.

Design of Control Unit

- The various fields encountered in instruction formats provide control bits to initiate microoperations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching.
- Each field requires a decoder to produce the corresponding control signals.
- The control memory output of each subfield must be decoded to provide the distinct microoperations. The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- The following figure 9 shows the three decoders and some of the connections that must be made from their outputs.

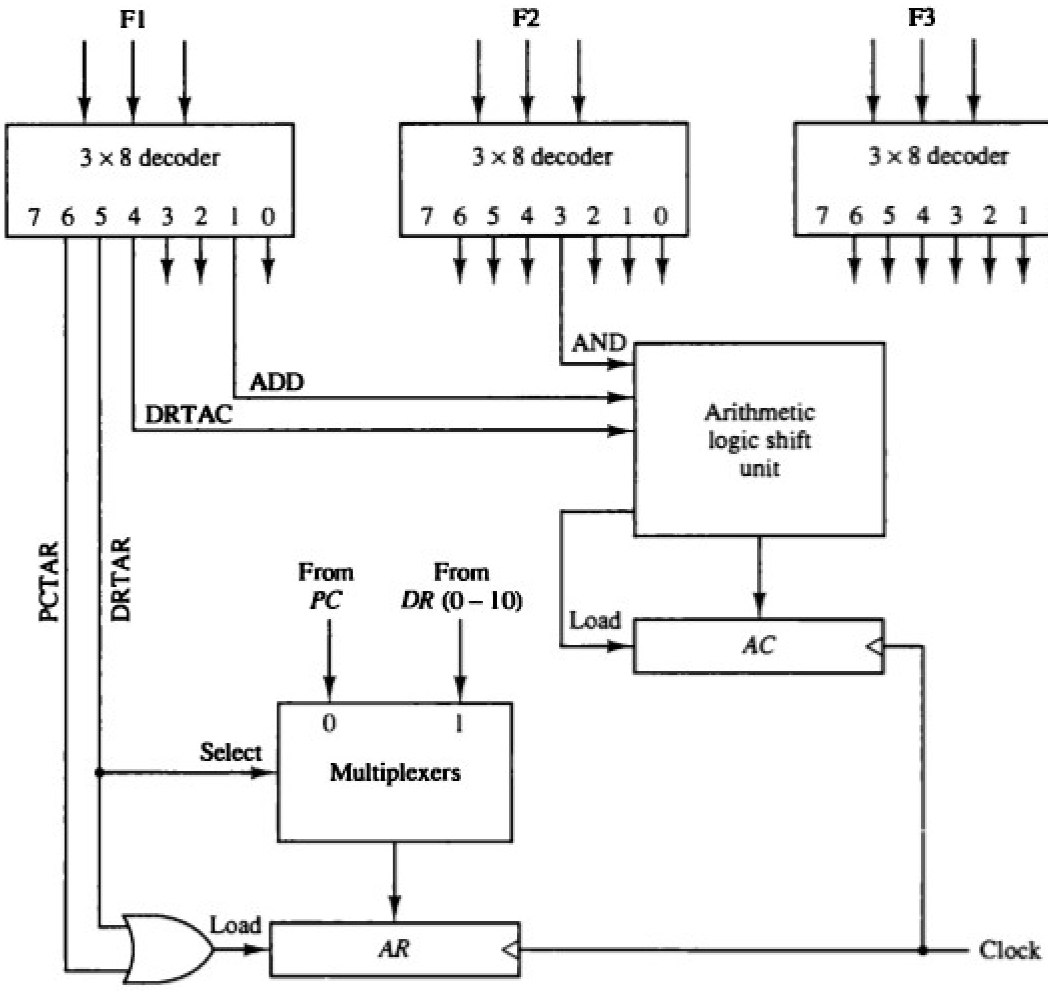


Figure 9: Decoding of microoperation fields

- Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3 x 8 decoder to provide eight outputs.

- Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation as specified in Table 3.
- For example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR(0-10) to AR (symbolized by DRTAR in Table 3). Similarly, when F1 = 110 (binary 6) there is a transfer from PC toAR (symbolized by PCTAR).
- As shown in figure 9, outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.
- The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive. The transfer into AR occurs with a clock pulse transition only when output 5 or output 6 of the decoder is active.
- The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion.
- In arithmetic logic shift unit designing, Instead of using gates to generate the control signals marked by the symbols AND, ADD, and DR, these inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively, as shown in figure 9.
- The other outputs of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

Microprogram Sequencer

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address. The address selection part is called a microprogram sequencer.
- A microprogram sequencer can be constructed with digital functions to suit a particular application. However, just as there are large ROM units available in integrated circuit packages, so are general-purpose sequencers suited for the construction of microprogram control units.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.
- The block diagram of the microprogram sequencer is shown in figure 10. The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.

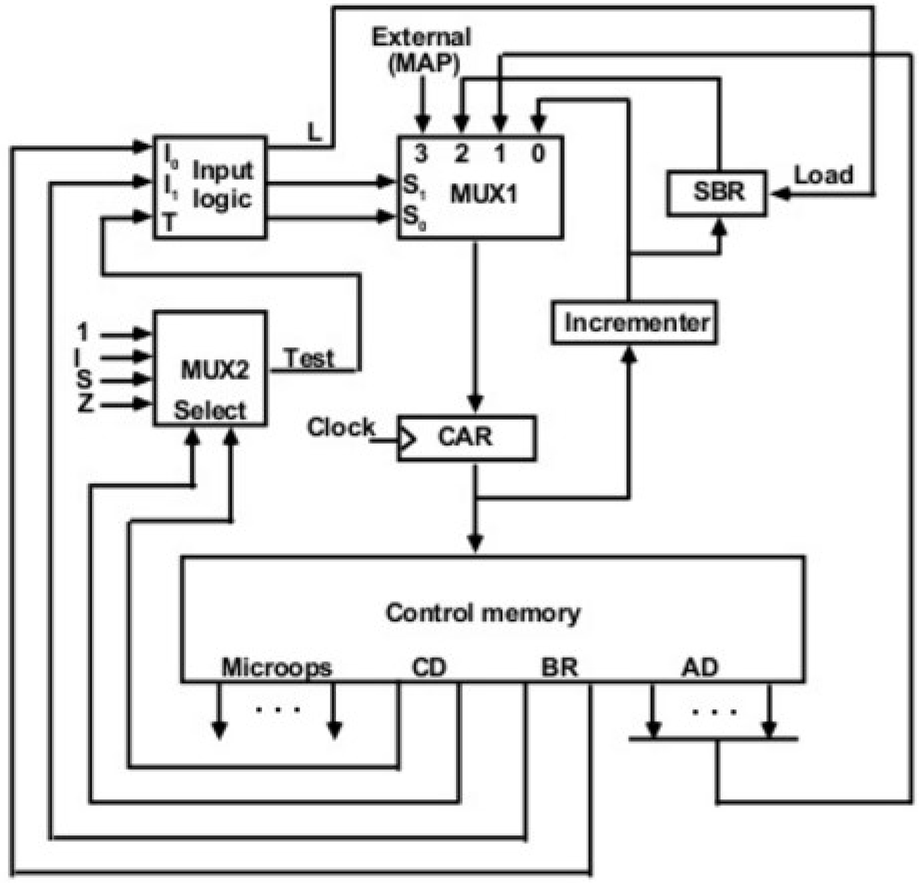


Figure 10: Microprogram sequencer for a control memory.

- There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register CAR. The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.
- The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR.
- The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction.
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T (test) variable is equal to 1, otherwise it is equal to 0.
- The T value together with the two bits from the BR (branch) field go to an input logic circuit. The input logic in a particular sequencer will determine the type of operations that are available in the unit.

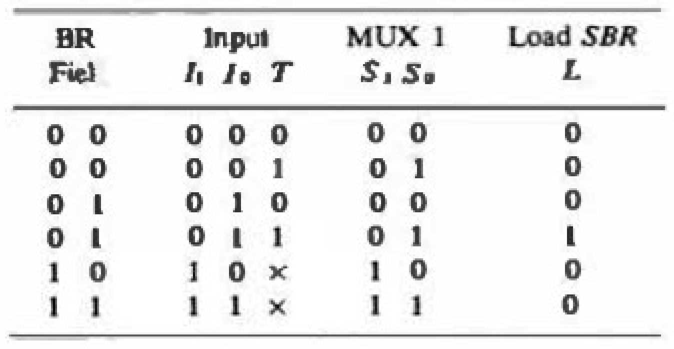
- The input logic circuit in figure 10 has three inputs, I_0, I_1, and T, and three outputs, S_0, S_1, and L.
- Variables S_0 and S_1, select one of the source addresses for CAR.
- Variable L enables the load input in SBR.
- The binary values of the two selection variables determine the path in the multiplexer. For example, with S_1 S_0 = 10, multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR.
- The truth table for the input logic circuit is shown in Table 6. Inputs I_1 and I_0 are Identical to the bit values in the BR field.
- The function listed in each entry was defined in Table 3. The bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).
- The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$

$$S_0 = I_1 I_0 + I_1' T$$

$$L = I_1' I_0 T$$

Table 6: Input logic Truth Table for MicroProgram Sequence



BR Fiel	Input I_1 I_0 T	MUX 1 S_1 S_0	Load *SBR* L
0 0	0 0 0	0 0	0
0 0	0 0 1	0 1	0
0 1	0 1 0	0 0	0
0 1	0 1 1	0 1	1
1 0	1 0 ×	1 0	0
1 1	1 1 ×	1 1	0

Hardwired control unit

- 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.
- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

- A hardwired control for the basic computer is as follows. The block diagram of the control unit is shown in following figure 11.

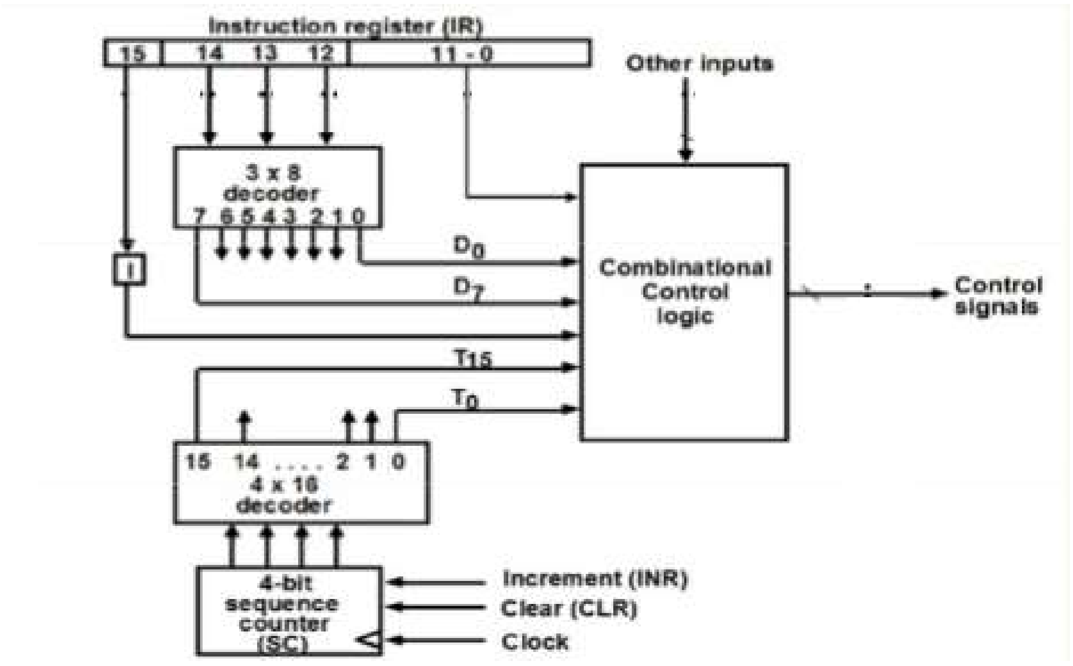


Figure 11: Control unit of basic computer

- 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 instruction register is shown in figure 11, where it is divided into three parts: the *I* 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 D_0 through D_7.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- 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 T_0 through T_{15}.
- 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 awhile, the counter is cleared to 0, causing the next active timing signal to be T_0. As an example, consider the case where SC is incremented to provide timing signals T_0, T_1 T_2, T_3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active.

Hardwired control unit	**Micro-programmed Control Unit**
1. It uses flags, decoder, logic gates and other digital circuits.	1. It uses sequence of micro-instruction in micro programming language.
2. As name implies it is a hardware control unit.	2. It is mid-way between Hardware and Software.
3. On the basis of input Signal output is generated.	3.It generates a set of control signal on the basis of control line.
4. Difficult to design, test and implement.	4. Easy to design, test and implement.
5. Inflexible to modify.	5. Flexible to modify.
6. Faster mode of operation.	6. Slower mode of operation.
7. Expensive and high error.	7. Cheaper and less error.
8. Used in RISC processor	8. Used in CISC processor.