UNIT II

Micro programmed Control: Control memory, Address sequencing, micro program example, design of control unit.

Central Processing Unit: General Register Organization, Instruction Formats, Addressing modes, Data Transfer and Manipulation, Program Control.

Micro programmed Control

Control Unit:

The main function of control unit is to initiate sequences of micro operations. The number of micro operations in the systems is finite.

Two major types of Control Unit:

0. Hardwired Control:

When the **control signals are generated by Hardware using conventional logic design techniques** then the control unit is said to be **hardwired.**

The control logic is implemented with gates, F/Fs, decoders, and other digital circuits

The key characteristics are

- High speed of operation
- Expensive
- Relatively complex
- No flexibility of adding new instructions (Wiring change-if the design has to be modified)

Examples of CPU with hardwired control unit are Intel 8085, Motorola 6802, Zilog 80, and any RISC CPUs.

1. **Microprogrammed Control**:

The control information is stored in a **control memory**, and

The control memory is programmed to initiate the required sequence of micro operations

Any required change can be done by updating the micro program in control memory, - Slow operation

The key characteristics are

- Speed of operation is low when compared with hardwired
- Less complex
- Less expensive
- Flexibility to add new instructions

Examples of CPU with micro programmed control unit are Intel 8080, Motorola 68000 and any CISC CPUs.

The control function that **specifies a micro operation is a binary variable.**

When it is in one state the corresponding micro operation is executed. The opposite state does not change the state of registers.

Control signal (that specify microoperations) in a bus-organized system are: **A groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units**

Control unit initiates a series of micro operations. During any time certain micro operations are initiated while others are idle.

Control Word:

The control variables (specifying a micro operation) at any given time can be represented **by a string of 1's and 0's** is called **"control word".**

Microprogrammed Control Unit :

A control unit whose **binary control variables are stored in memory** (control memory)

Microinstruction

The microinstruction specifies one or more microoperations for the system.

Microprogram

A sequence of microinstruction

Control Memory:

A Memory that is a part of control unit. The control unit consists of **control memory used to store the micro program.** Control memory is a permanent i.e., read only memory (ROM).

A computer having a Micro programmed Control Unit has 2 separate Memories :

1. **Main Memory:** For storing user program (Machine instructions/data) . The contents of the main memory may alter.

2. **Control Memory**: For storing microprogram that can not be altered.

The **Microprogram c**onsists of microinstructions.

Microinstruction specifies various control signals for execution of register micro operations.

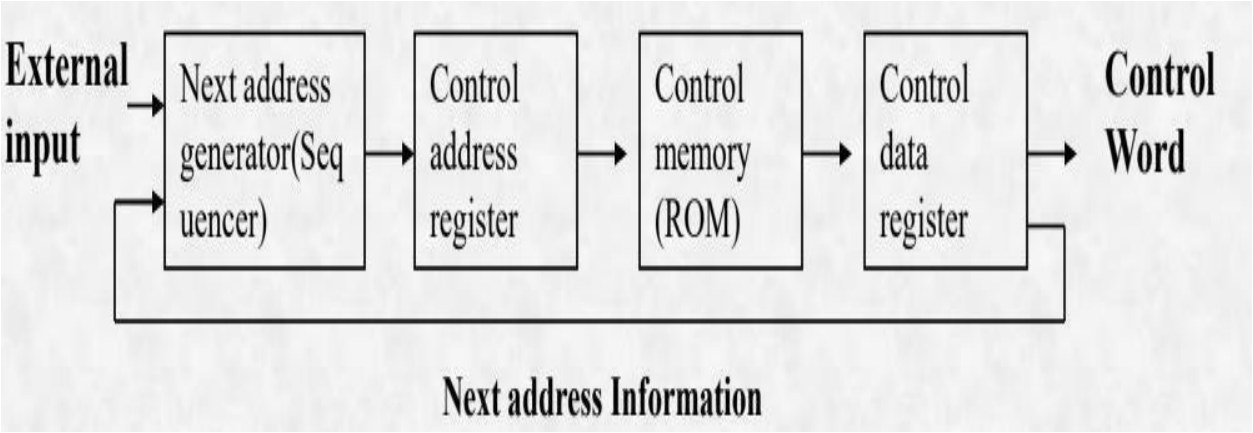
Each machine Instruction initiates a series of Microinstructions in control Memory.

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,
- To return control to the fetch phase in order to repeat the cycle for the next instruction.

Micro programmed Control Organization

The general configuration of a micro programmed control unit is shown below



Control Memory (ROM):

- A memory is part of a control unit.
- All the control Information is permanently stored.

2. Control Address Register

- Specify the address of the microinstruction

3. Control Data Register (Pipeline Register)

- Hold the present microinstruction(specifies one or moremicrooperations) read from control memory

  To generate the address of the next microinstruction, some bits of present micro instruction can be used.
- Thus a microinstruction contains bits for initiating the microoperations and bits that determine the address sequence for control memory.

4. Next Address Generator (Sequencer)

- Determine the address sequence that is read from control memory
- Next address of the next microinstruction can be specified several way depending on the sequencer input

Address Sequencing:

Microinstructions are stored in control memory in groups, with each group specifies a **Routine.**

The hardware that controls the address sequencing of the control memory must be **able of sequencing the microinstruction within a routine and be able to branch from one routine to another.**

Fetching:

An initial address is loaded into the control address register when power is turned on. This address is the address of the first microinstruction that activates the fetch routine.

After the end of fetch routine, the instruction is in the instruction register of the computer (Decoding).

The control memory next must go through the routine that **determines the effective address of the operand.** After computing the effective address, 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 upon the operation code part of instruction.

Each instruction has its own microprogram routine stored in a given location of the control memory.

The transformation from the instruction code bits to an address in the control memory where the routine is located is called as **Mapping.**

After the execution of the instruction control must return to the fetch routine.

Address Sequencing Capabilities:

1) Incrementing of the control address register
2) Unconditional branch or conditional branch, depending on status bit conditions
3) Mapping process (bits of the instruction address for control memory)
4) A facility for subroutine return

The below figure shows a block diagram of a control memory and the associated hardware needed for selecting the next Micro instruction address

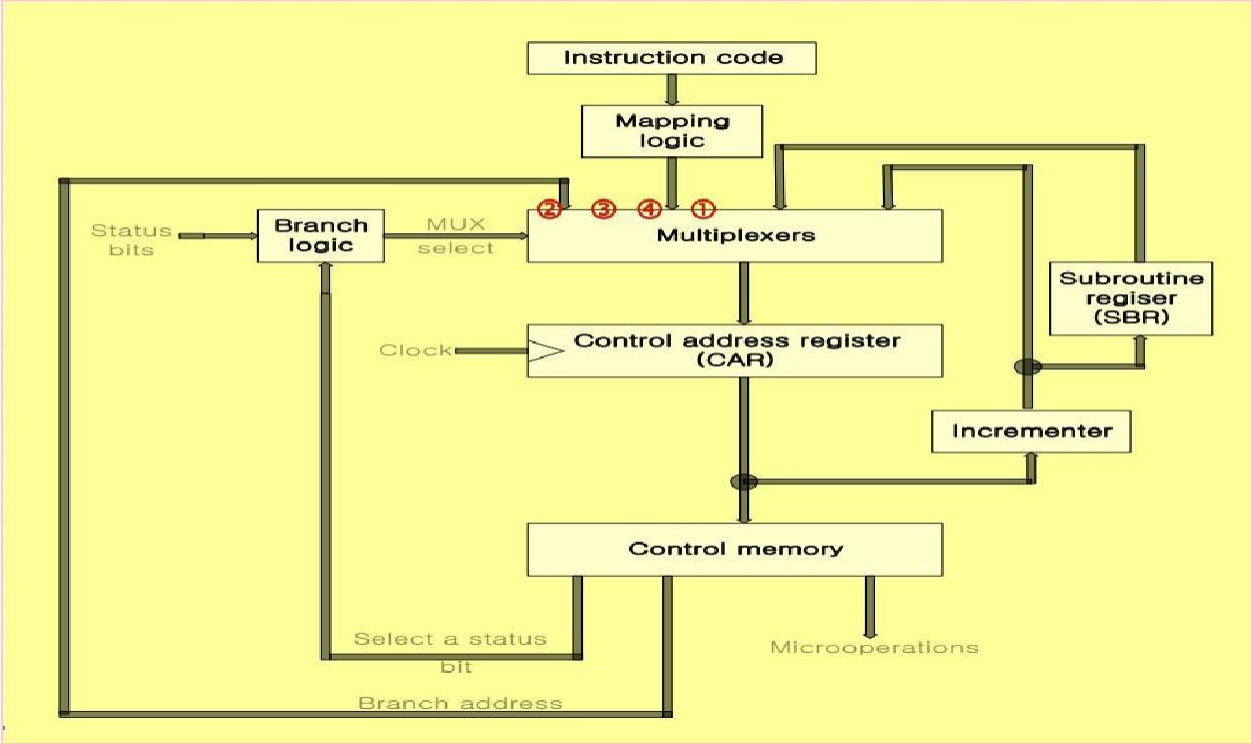


Fig: Selection of Address for Control Memory

Selection of Address for Control Memory

The Micro instruction in control memory contains **set of bits to initiate micro operations in computer registers** and **other bits to specify the method by which the next address is obtained.**

The diagram shows **four different paths** from which the **Control Address Register receives the information.**

1) Incrementer (Increment CAR by 1)

2) Branch address from control memory

3) Mapping Logic (External Address from main memoryto control memory)

4) SBR: Subroutine Register

- Return Address cannot be stored in ROM
- Return Address for a subroutine is stored in SBR

Conditional Branching

Status conditions are special bits in the system that provide parameter information such as carry-out of an adder, sign bit of number, mode bits of an instruction, input /output status condition.

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.

Branch Logic: It can be implemented in different ways.

The simple 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 the help of **Multiplexer.**

Example: Let there are eight status bit conditions in the system. Three bits in the microinstruction are used to specify one of eight status bits.

If the selected status bit is in the 1 state the output of the multiplexer is 1, otherwise 0.

The 1 output in the multiplexer generates a control signal **to transfer the branch address from the microinstruction into the control address register** otherwise **address register to be incremented.**

The unconditional branch microinstruction can be implemented by loading the branch address from control memory into control address register.

MAPPING OF INSTRUCTIONS TO MICROROUTINES:

Mapping from the OP-code of an instruction to the address of the microinstruction which is the starting microinstruction of its execution microprogram

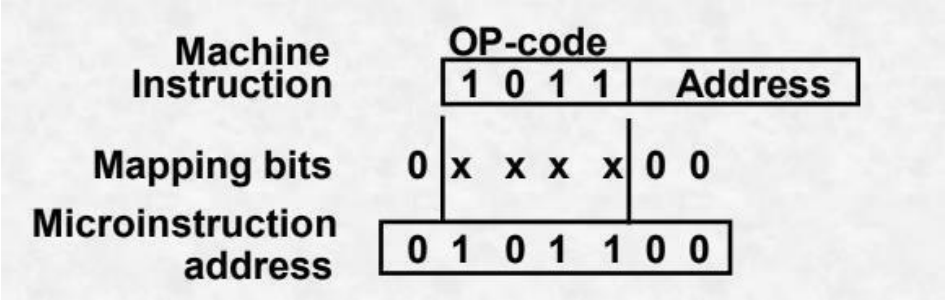


Fig Mapping from instruction code to Microinstruction address

Consider a 4 bit Opcode = specify up to 16 distinct instructions. And assume that control memory has 128 words.

Mapping Process : Converts the 4-bit Opcode to a 7-bit control memory address

1) Place a "0" in the most significant bit of the address
2) Transfer 4-bit Operation code bits
3) Clear the two least significant bits of the CAR

Mapping Function:

Mapping is implemented by ROM or PLD. A PLD (an Integrated Circuit) is similar to ROM except that it uses AND and OR gates with internal electronic fuses.

The interconnection between AND, OR and outputs can be programmed as in ROM

Subroutine:

Subroutines are program 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 micro program.

Frequently many microprogram contain identical section of code. Microinstruction can be saved by employing subroutines that used common section of micro-code.

Ex. The sequence of micro operations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions.

This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Microprogram that uses Subroutines must have for storing return address during a subroutine call and restoring the address during a subroutine return.

Microprogram Example:

The process of code generation for the control memory is called *microprogramming.*

The block diagram of the computer configuration is shown in below figure.

Two memory units:

1. Main memory – stores instructions and data
2. Control memory – stores microprogram

Four processor registers :

1. Program counter – PC
2. Address register – AR
3. Data register – DR
4. Accumulator register - AC

Two control unit registers

1. Control address register – CAR
2. Subroutine register – SBR

Transfer of information among registers in the processor is through MUXs rather than a bus.

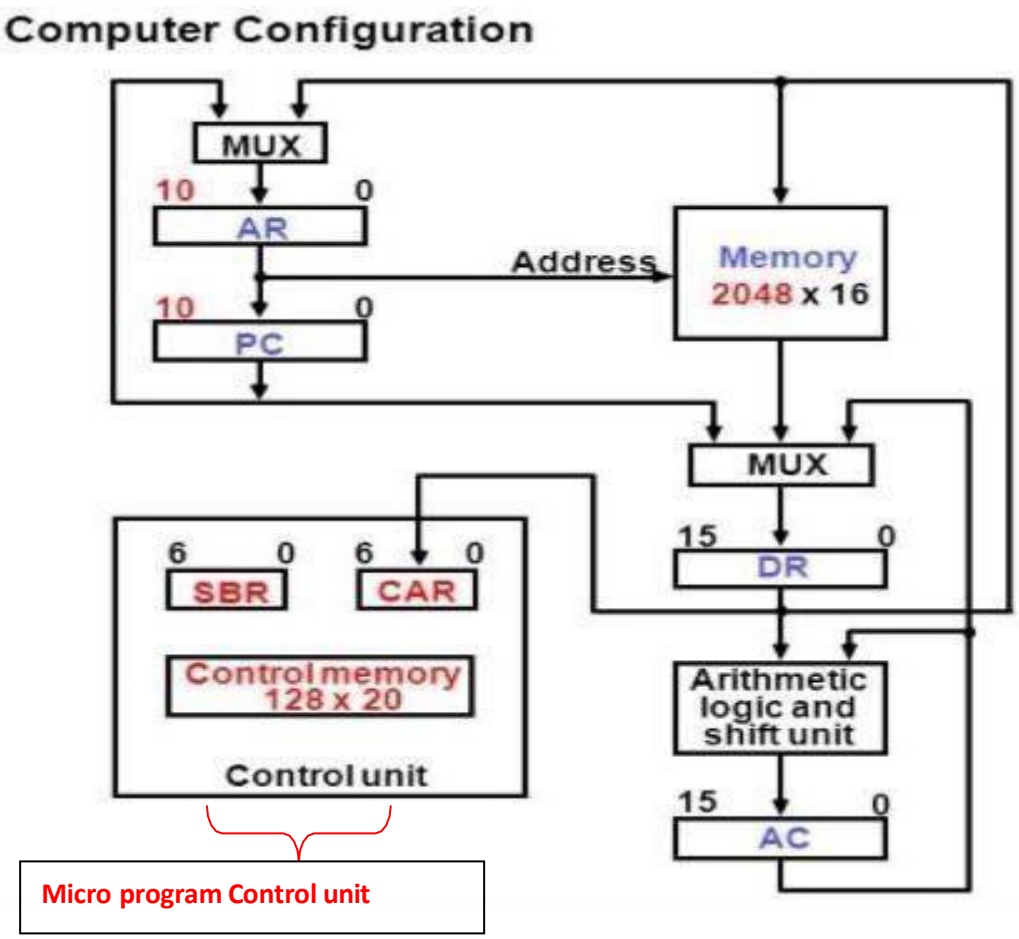


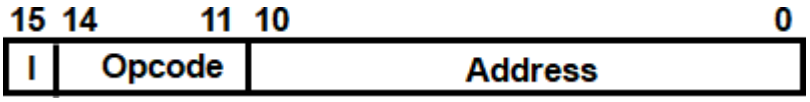
Fig: Computer Hardware configuration

MACHINE INSTRUCTION FORMAT

It consists of 3 fields:

1. I bit to denote Direct or Indirect Addressing
2. 4 bit opcode
3. 11 bit Address Field

Machine instruction format=16 BIT **Address size=11bits**



Sample machine instructions

Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

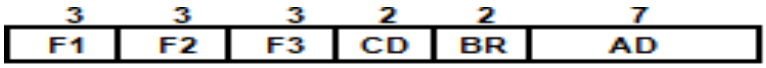
MICRO INSTRUCTION FORMAT*:*

The Microinstruction format for the control memory is shown In figure. The 20 bits of the microinstruction are divided into 4 functional parts.

Control Memory

128*20

Microinstruction Format=20 BIT==6 FIELDS



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

The microinstruction format is composed of 20 bits with four parts to it

1. **Three fields F1, F2, and F3 specify micro operations for the computer [3 bits each].**
2. **The CD field selects status bit conditions [2 bits]**
3. **The BR field specifies the type of branch to be used [2 bits]**
4. **The AD field contains a branch address [7 bits]**

Each of the three micro operation fields can specify one of seven possibilities. This gives a total of 21 Instructions.

Not more than three micro operations can be chosen for a microinstruction.

If fewer than three micro operations are used, the next 1 or more fields will use the binary code 000 = NOP.

Each Micro operation in Table is defined with a register transfer statement and is assigned a symbol for symbolic notation.

The three bits in each field are encoded to specify seven distinct microoperations listed in below table.

All transfer type micro operations symbols use five letters. The first 2 letters indicate source register and the third letter is always T, and last 2 letters designate destination register.

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Table: Symbols and Binary code for MicroInstruction Fields

MICROINSTRUCTION FIELD DESCRIPTIONS - CD, BR

The condition field (CD) is two bits to specify four status bit conditions shown below

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 branch field (BR) consists of two bits and is used with the address field to choose the. address of the next microinstruction.

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

SYMBOLIC MICROINSTRUCTIONS

Symbols are used in microinstructions as in assembly language

A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.

Each symbolic microinstruction is divided into 5 fields:

Label, Micro operations, CD, BR, and AD.

Sample Format

Five fields: label; micro-ops; CD; BR; AD

Label: May be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z}, where
- U: Unconditional Branch
- I: Indirect address bit
- S: Sign of AC
- Z: Zero value in AC

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty}

SYMBOLIC MICROPROGRAM - FETCH ROUTINE

During FETCH, Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the fetch cycle:

```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

Symbolic microprogram for the fetch cycle:

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

Binary equivalents translated by an assembler

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

- Control Storage: 128 20-bit words
- The first 64 words: Routines for the 16 machine instructions
- The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)
- Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, …, 60

Partial Symbolic Microprogram

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

SBR= CAR+1=0000001
CAR= 0000001
DR(15)=1

BINARY MICROPROGRAM

Micro Routine	Address Decimal	Address Binary	Binary Microinstruction F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANG	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

This microprogram can be implemented using ROM

Design of Control Unit

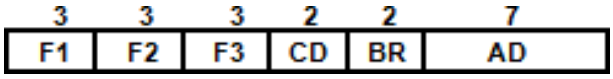
The bits of microinstruction are usually divided into fields, with each field defining a distinct, separate function.

The various fields available in the instruction format provide control bits to initiate the microoperation.

Special bits(status bits) are used to specify the way that the next address is to be evaluated and an address field for branching.

Decoding of Microinstruction Fields :

- F1, F2, and F3 of Microinstruction are decoded with a 3 x 8 decoder
- Output of decoder must be connected to the proper circuit to initiate the corresponding microoperation .



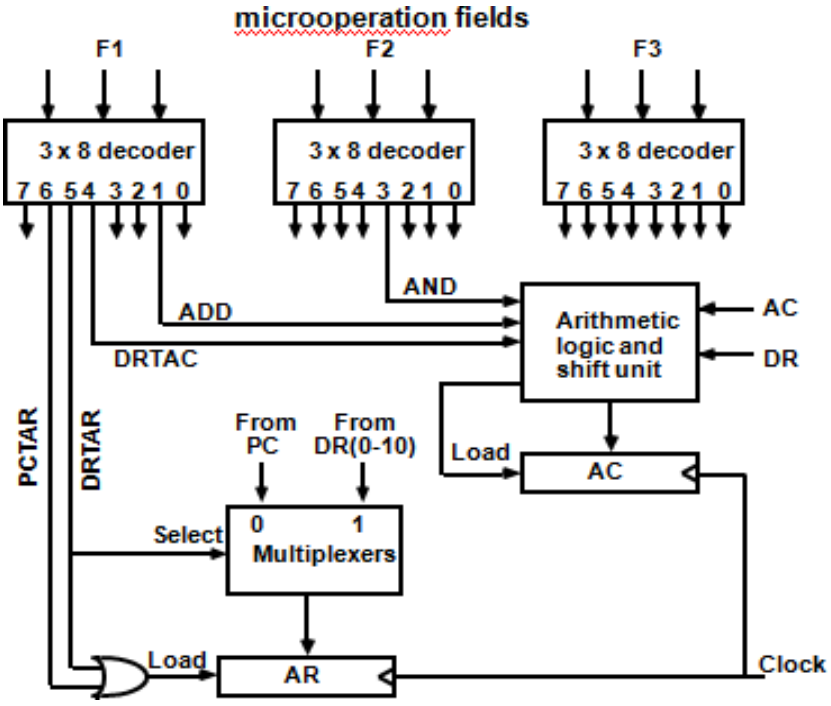


Fig Decoding of Micro operation fields

When F1 = 101 (binary 5), the next pulse transition transfers the content of DR (0-10) to AR.

Similarly, when F1= 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR).

As shown in figure, 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 transition only when output 5 or output 6

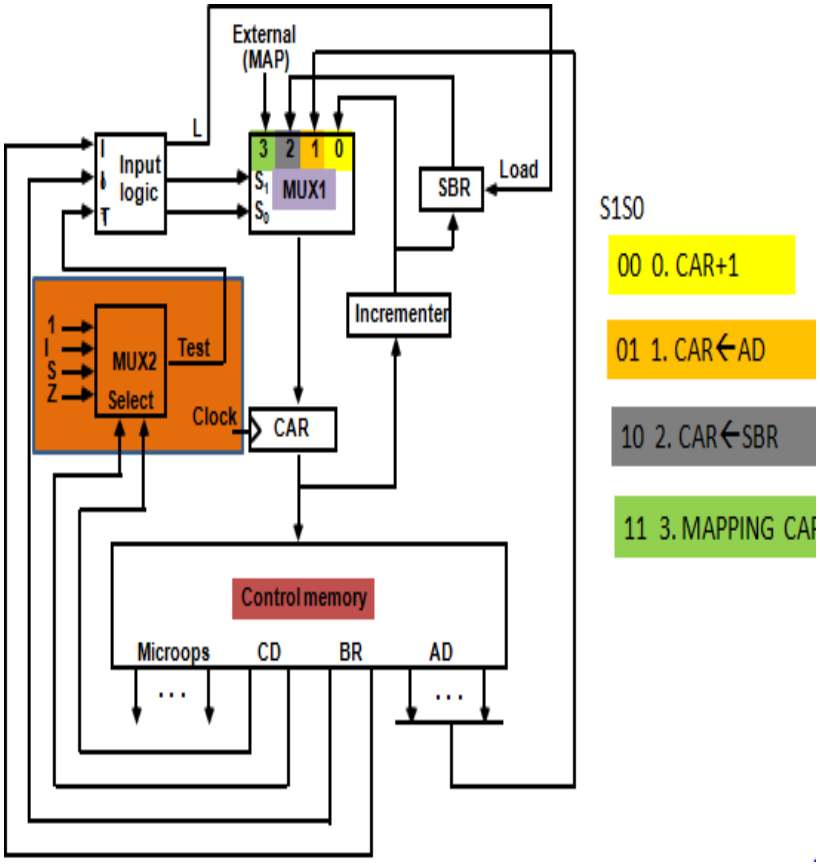
of the decoder is active.

For the arithmetic logic shift unit the control signals are instead of coming from the logical gates, now these inputs will now come from the outputs of AND, ADD and DRTAC respectively.

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.**
- 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 block diagram of the micro program sequencer is shown in below figure.

Microprogram sequencer for Control Memory



There are two multiplexers in the circuit.

1The first multiplexer selects an address from one of four sources and routes it into control address register *CAR.*

1. 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 registers *SBR.*

The other three inputs to multiplexer come from

1. The address field of the present microinstruction
2. from the out of SBR
3. 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 variable is equal to 1; otherwise, it is equal to 0.

The *T value together with two bits from the BR (branch) field goes to an input logic circuit.*

The input logic in a particular sequencer will determine the type of operations that are available in the unit.

MICROINSTRUCTION FIELD DESCRIPTIONS - CD, BR

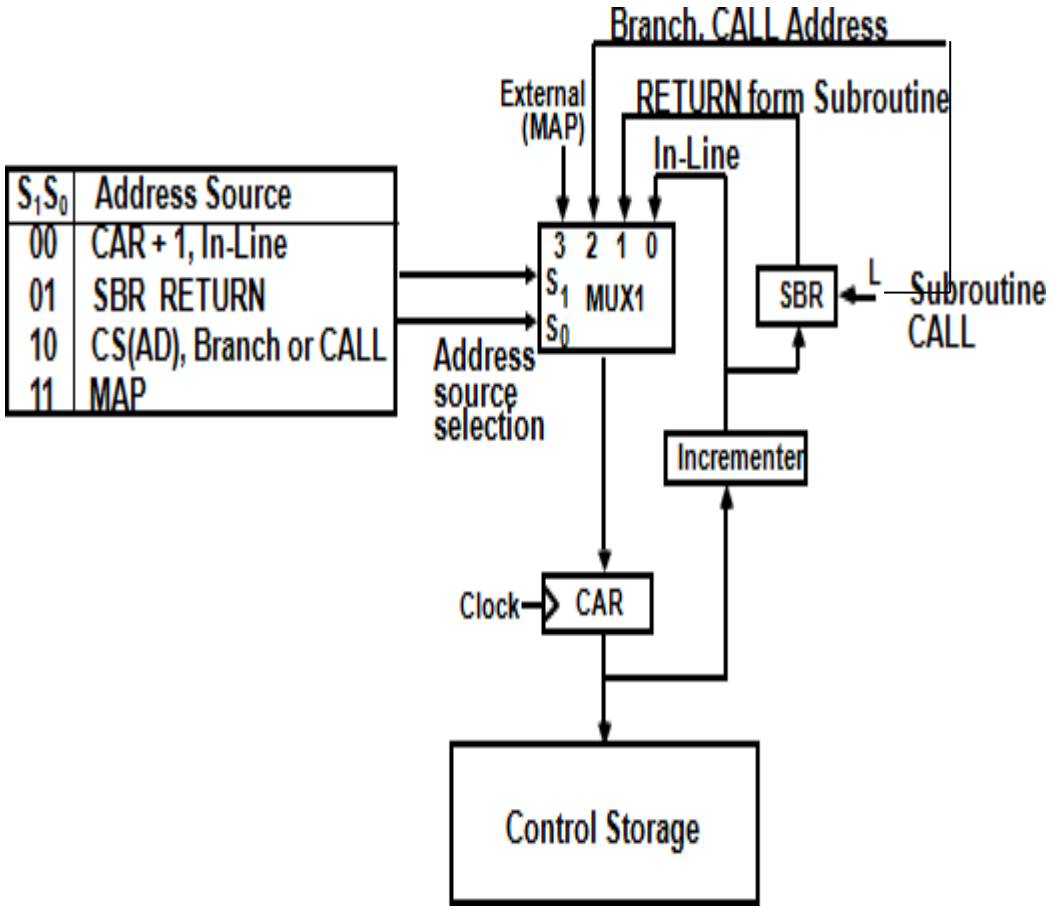
CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

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

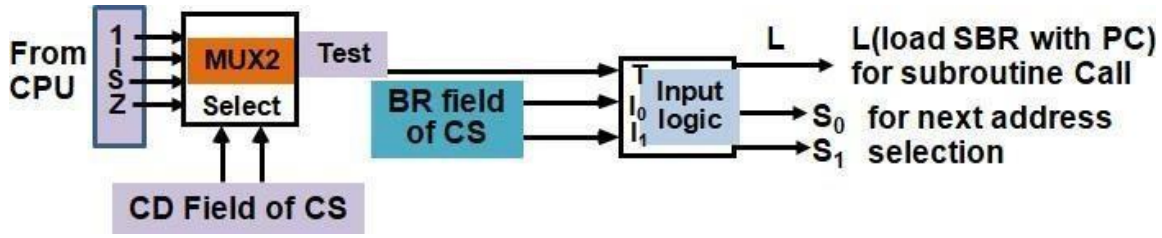
MICROPROGRAM SEQUENCER-NEXT MICROINSTRUCTION ADDRESS LOGIC

MUX-1 selects an address from one of four sources and routes it into a CAR

- In-Line Sequencing CAR + 1
- Branch, Subroutine Call CS(AD)
- Return from Subroutine Output of SBR
- New Machine instruction MAP



MICROPROGRAMSEQUENCER-CONDITION AND BRANCH CONTROL



Input Logic

I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CAR←(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$

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

General Register Organisation:

In computer architecture, general-purpose registers are a set of fast storage locations within the CPU used to hold data and instructions during program execution. Unlike architectures with a single accumulator, general-purpose registers

allow for more flexible and efficient data manipulation by providing multiple storage locations for various data types.

Here's a more detailed explanation:

Key characteristics of general-purpose registers:

Temporary storage:

They hold data and instructions temporarily while the CPU is actively processing them.

Multiple registers:

Instead of a single accumulator, a set of registers is used, allowing for more complex operations and faster data access.

Various data types:

They can store different types of data, such as integers, floating-point numbers, memory addresses, and control information.

Faster access:

Accessing data stored in registers is significantly faster than accessing data from main memory.

How they work:

The CPU uses these registers to perform arithmetic and logical operations, store intermediate results, and manage program execution.

Instructions are designed to operate directly on data stored in registers, which speeds up processing.

The registers are part of the CPU's internal structure and are closely linked to the Arithmetic Logic Unit (ALU).

Examples of General Purpose Registers:

x86 architecture:

Includes registers like EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI, according to Learning Malware Analysis.

8086 microprocessor:

Includes registers like AX, BX, CX, DX, SP, BP, SI, and DI,

Other related terms:

Program Counter (PC): Keeps track of the memory address of the next instruction to be executed.

Stack Pointer (SP): Points to the top of the stack, used for managing function calls and local variables.

Status Flags: Store information about the results of operations, such as whether a result is zero or negative.

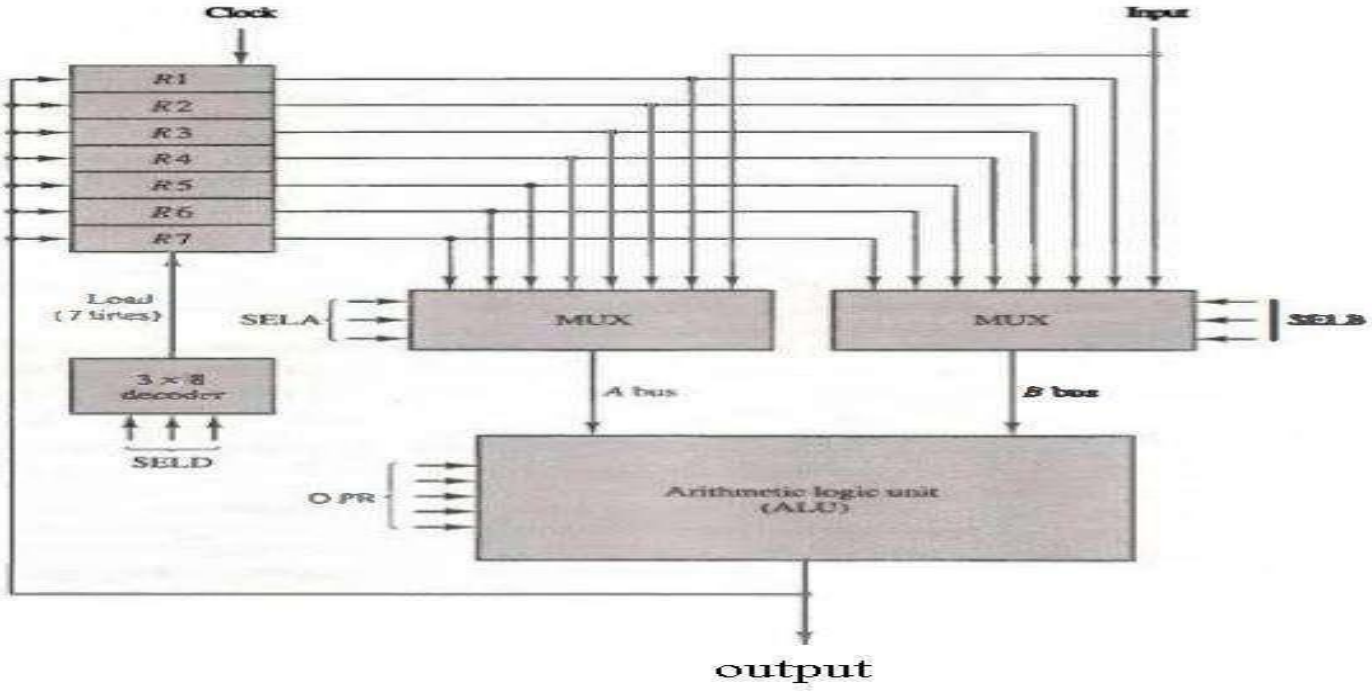
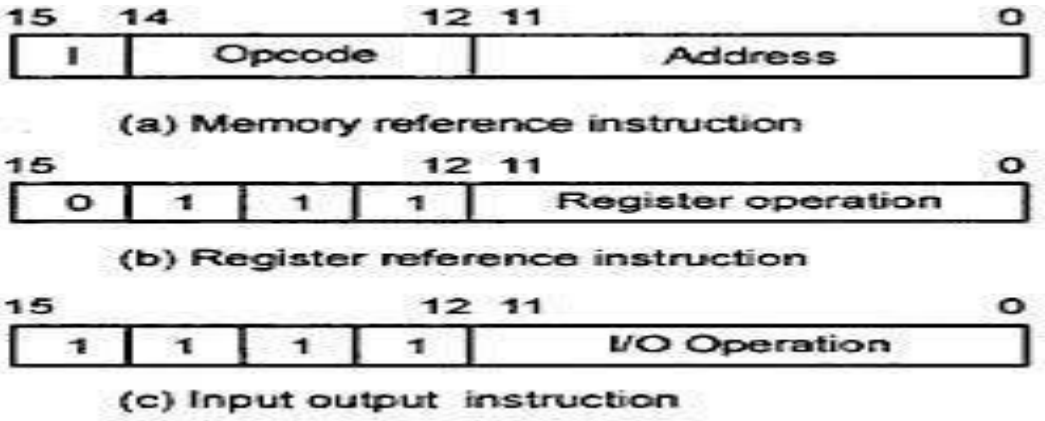


Figure 2 Register set with common ALU.

The basic computer has three instructions code formats, as shown in the following figures.



Each format has 16 bits. the operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address. The register-reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit of the instruction. A register-reference instruction specifies an operation on AC register. An input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction.

2.1.1. Instruction Set Completeness

A computer should have a set of instructions so that the use can construct machine language programs to evaluate any function that is known to be computable. The set of instructions are said to be complete if the computer includes a sufficient

number of instructions in each of the following categories:

1. Arithmetic, logical, and shift instructions
2. Instructions for moving information to and from memory and processor registers
3. Program control instructions together with instructions that check status conditions
4. Input and output instructions

2.1.2. INSTRUCTION CYCLE

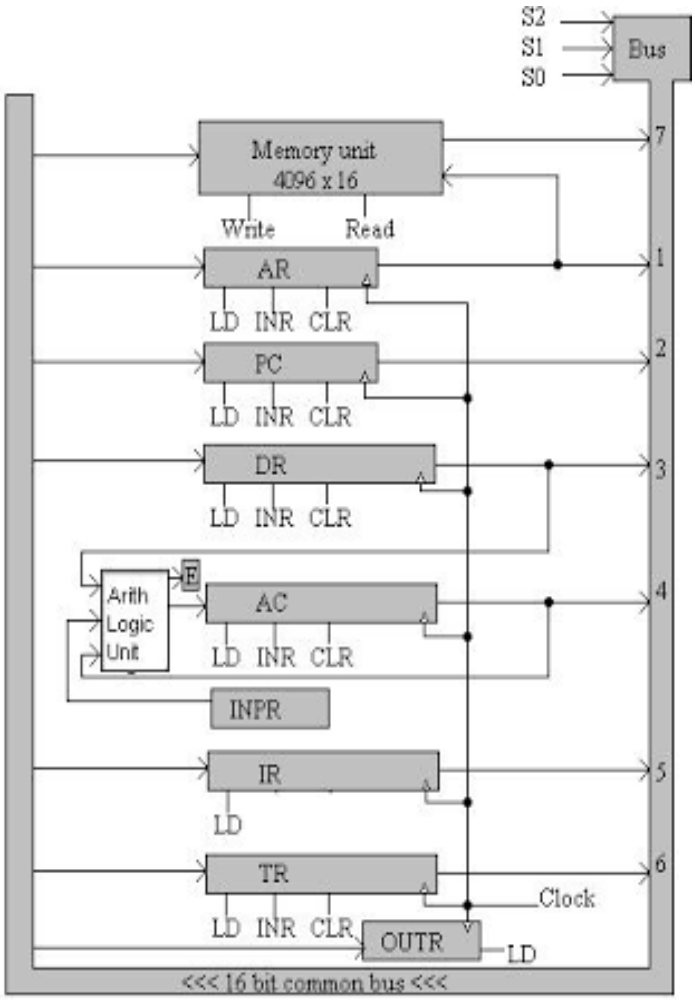
A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.

4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.

2.1.3. Fetch and Decode



Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2 and so on. The micro operations for fetch and decode phases can be specified by the following register transfer statements.

T_0 : AR← PC

T_1 : IR ← M[AR], PC ← PC + 1

T_2 : $D_0, D_1, \ldots\ldots, D_7$ ← Decode IR(12-14), AR ← IR(0-11), I← IR(15)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing

signal T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. SC is incremented after each clock pulse to produce the sequence T0,T1 and T2. The above figure shows the implementation of the first two register transfer statements.

To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.
2. Transfer the content of the bust to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since T_0 =1. In order to implement the second statement it is necessary to use timing signal T1 to provide the following connections in the bus system.

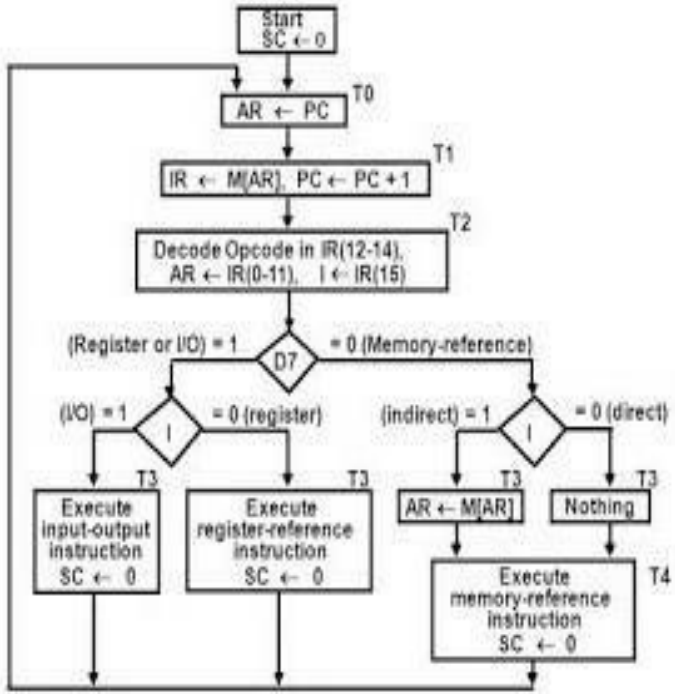
1. Enable the read input of memory.
2. Place the content of memory onto the bus by making S2S1S0 = 111.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since T1=1.

2.1.4. Determining the Type of Instruction

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory. The following flowchart determines the instruction type after the decoding.

Decoding output D7 is equal to 1 if the operation code is equal to binary 111. From the above figure we can determine that if D7=1, the instruction must be a register-reference or input-output type. If D7=0, the operation code must be one of the other sever values 000 through 110, specifying a memory-reference instruction. Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I. If D7=0 and I=1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory. The micro operation for the indirect address condition can be symbolized by the register transfer statement,



The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as,

D_7' I T3: AR ← M[AR]

D_7' I' T3: Nothing

D_7 I' T3: Execute a register-reference instruction

D_7 I T3: Execute an input-output instruction

2.1.5. REGISTER-REFERENCE INSTRUCTIONS

Register-reference instructions are recognized by the control when D_7 =1 and I=0. These instructions use bits 0 through 11 of the instruction code to specify one of the 12 instructions. These 12 bits are available in IR(0-11). They are transferred to AR during time T2. Let D_7I'T3= r, which is the Boolean relation for the control function. The control function is distinguished by one of the bits in IR(0-11). By assigning the symbol B_i to bit I of IR, all control functions can be simply denoted by rB_i. The control functions and micro operations for the register-reference instructions are listed in the following table.

Symbol	Control Function	Symbolic Description	Meaning
CLA	rB_{11}	AC←0	Clear AC
CLE	rB_{10}	E←0	Clear E
CMA	rB_9	AC← $\overline{AC}$	Complement AC
CME	rB_8	E← $\overline{E}$	Complement E
CIR	rB_7	AC← shr AC, AC(15)←E, E←AC(0)	Circulate right
CIL	rB_6	AC ← shl AC, AC(0)←E, E←AC(15)	Circulate left
INC	rB_5	AC ← AC +1	Increment AC
SPA	rB_4	If(AC(15)=0) then (PC←PC +1)	Skip if positive
SNA	rB_3	If(AC(15)=1) then (PC← PC+1)	Skip if negative
SZA	rB_2	If(AC=0) then PC ← PC +1	Skip if AC zero
SZE	rB_1	If(E=0) then (PC ← PC +1)	Skip if E zero
HLT	rB_0	S ←0(S is a start-stop flip-flop)	Halt computer

2.2. MEMORY-REFERENCE INSTRUCTIONS

1. AND to AC: This is an instruction that performs the AND logic operation on pairs of bits in AC and memory word specified by the effective address. The result of the operation is transferred to AC. The micro operations that execute this instruction are:

D_0T_4 : DR ← M[AR]

D_0T_5 : AC ← AC ∧ DR, SC ←0

2. ADD to AC: This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are,

D_1T_4: DR ← M[AR]

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ←0

3. LDA: Load to AC: This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

D_2T_4: DR ←
M[AR] D_2T_5:
AC ← DR,
SC ←0

4. STA: Store AC: This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one micro operation:

D_3T_4: M[AR] ← AC, SC ←0

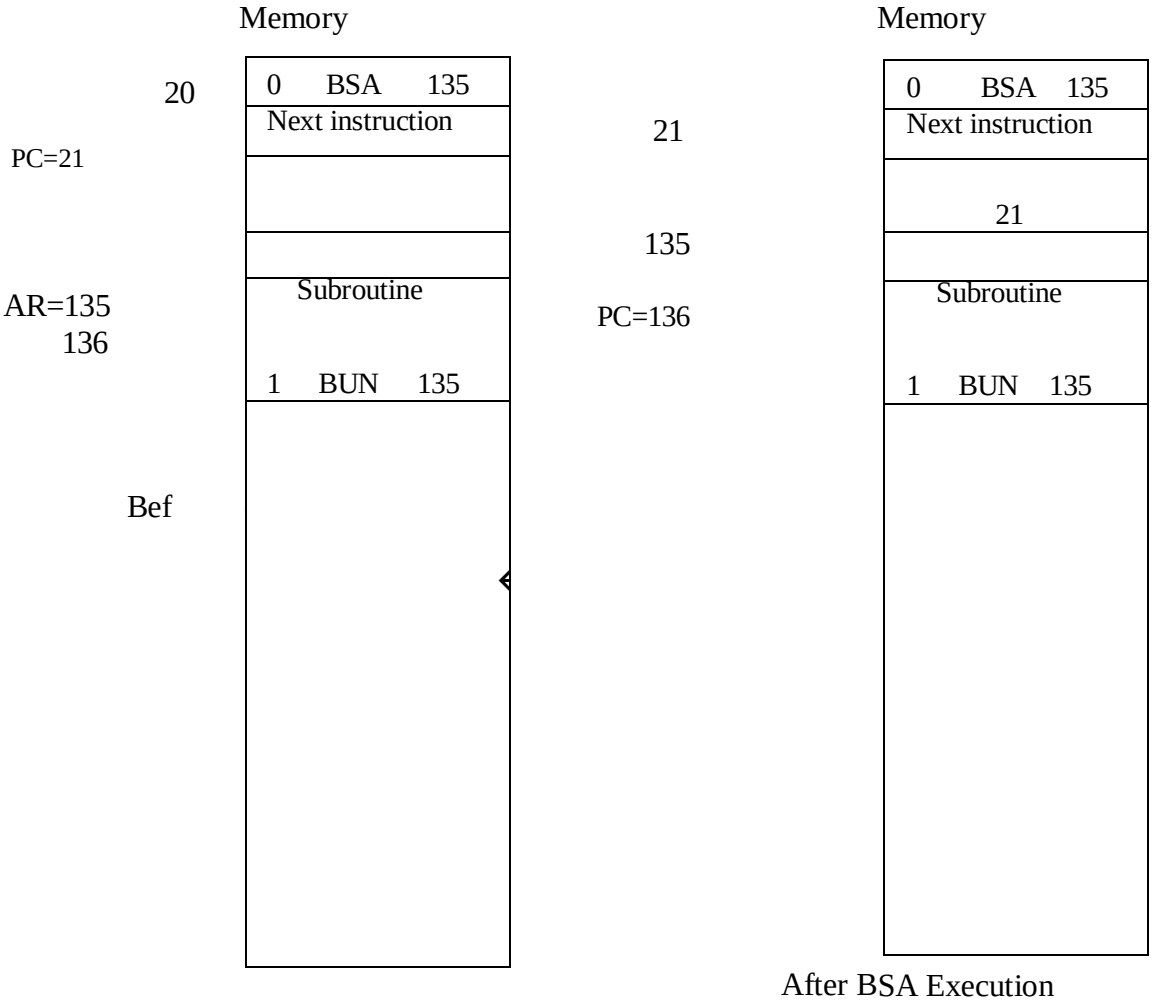
5. BUN: Branch Unconditionally: This instruction transfers the program to the instruction specified by the effective address. PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of the sequence, and the program branches (or jumps) unconditionally. The instruction is executed with one micro operation:

D_4T_4: PC ← AR, SC ← 0

6. BSA: Branch and Save Return Address: This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified with the following register transfer:

M[AR] ← PC, PC ← AR + 1

A numerical example that demonstrates how this instruction is used with a subroutine is shown in the following figure.



21, PC ← 135 + 1 = 136

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two micro operations:

$$D_5 T_4: M[AR] \leftarrow PC, AR \leftarrow AR + 1$$
$$D_5 T_5: PC \leftarrow AR, SC \leftarrow 0$$

7. ISZ: Increment and Skip if Zero: This instruction increments the word specified by the effective address, and if the increment value is equal to 0, PC is incremented by 1. The programmer usually stores a negative numbers (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is increment by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations

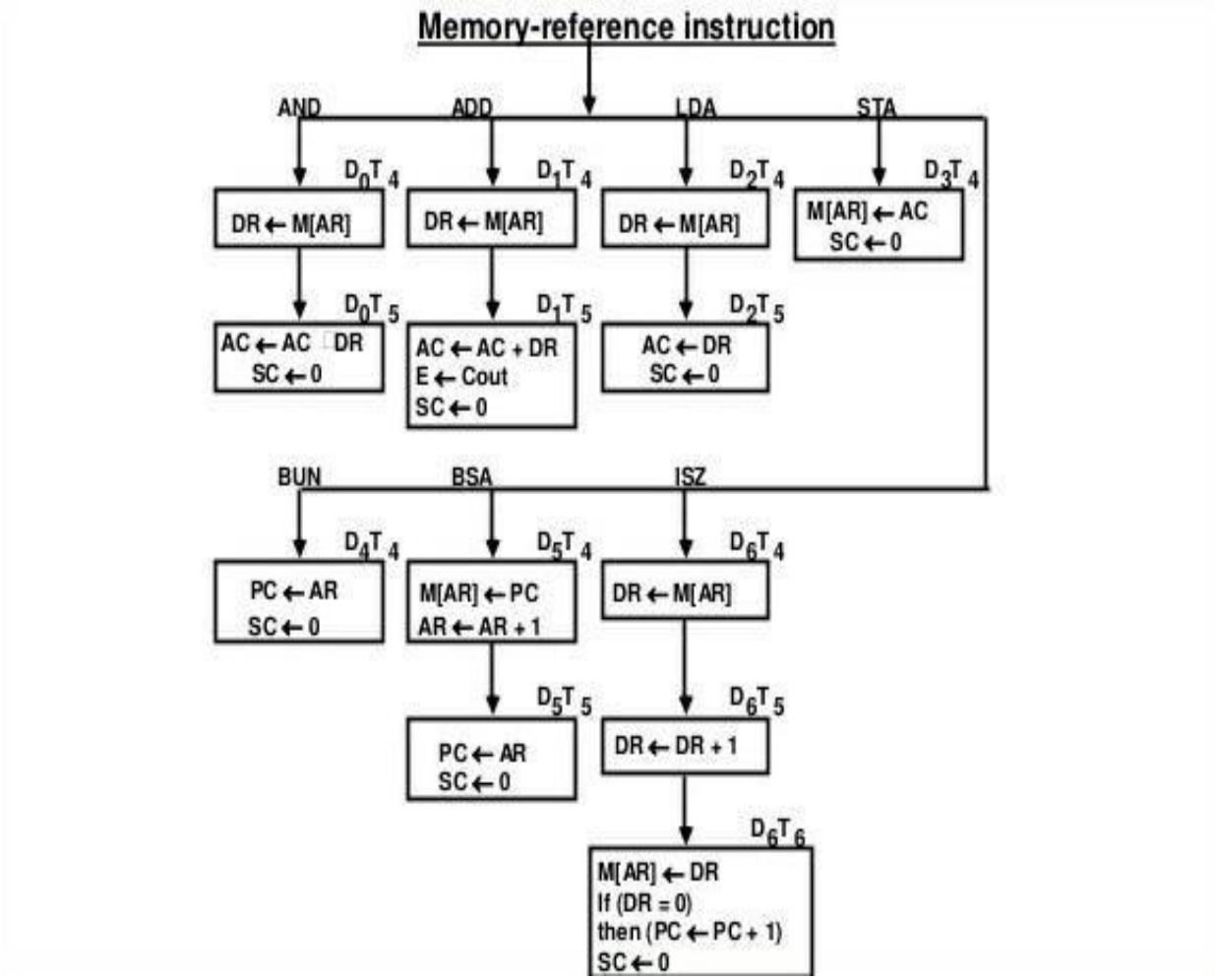
$$D_6 T_4: DR \leftarrow M[AR]$$
$$D_6 T_5: DR \leftarrow DR + 1$$

D_6T_6: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

2.2.1. Control Flowchart

The following flowchart shows all micro operations for execution of the seven memory-reference instructions.

Memory-reference instruction



2.2.1. INPUT-OUTPUT INSTRUCTIONS

Input-output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 1111 and are recognized by the control when $D_7 = 1$ and I=1. The remaining bits of the instruction specify the particular operation. Let $D_7IT_3 = p$, which is the control function for the Boolean relation. The control function is distinguished by one of the bits in IR(6-11). By assigning the symbol B_i to bit I of IR, all control functions can be denoted by pB_i, for I=6 through 11. The control functions and micro operations for the input-output instructions are listed in the following table.

Symbol	Control Function	Symbolic Description	Meaning
INP	pB_{11}	AC(0-7)←INPR, FGI←0	Input character
OUT	pB_{10}	OUTR←AC(0-7), FGO←0	Output character
SKI	pB_9	If(FGI=1) then (PC←PC +1)	Skip on input flag
SKO	pB_8	If(FGO=1) then (PC←PC +1)	Skip on output flag

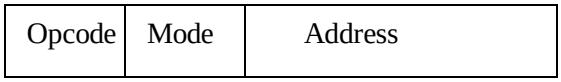
ION IOF	pB_7 pB_6	IEN←1 IEN←0	Interrupt enable on Interrupt enable off

2.3. ADDRESSING MODES

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming flexibility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

In some computers the addressing mode of the instruction is specified with a distinct binary code. Other computers use a single binary code that designates both the operation and the mode of the instruction. An example of an instruction format with a distinct addressing mode field is shown in the following figure.



The 'Mode' specifies any one of the different Addressing Modes. They are,

1. Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. For e.g., the instruction "Complement Accumulator" is an implied mode instruction, because the operand in the Accumulator register because the operand in the Accumulator register is implied in the definition of the instruction. Thus, all register-reference instructions that use an Accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions, since the operands are implied to be on top of the Stack.
2. Immediate Mode: In this mode the operand is specified in the instruction itself.
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.
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. The advantage of a Register Indirect Mode instruction is that the address field of the instruction uses fewer bits to select a register than a memory address directly.
5. Auto-increment or Auto-decrement Mode: This is similar to the register indirect mode except that the register is incremented after its value is used to access memory in case of Auto-increment and the register is decremented before its value is used to access memory in case of Auto-decrement. 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. The effective address is defined to be the memory address obtained from the computation directed by the given addressing mode
6. Direct Address Mode: In this mode the effective address is equal to the address part of the instruction.

7. Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.
8. Relative Address Mode: In this mode the content of the PC is added to the address part of the instruction in order to obtain the effective address.
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. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the Index Register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands.
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. 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 computer to facilitate the relocation of programs in memory. When programs and data are moved form one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position. With a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

2.3.1. Numerical Example

To show the differences between the various modes, let us assume the state of the memory and various processor registers as shown below.

Address	Memory
200	Load to AC | Mode
201	Address=500
202	Next Instruction
399	450
400	700
500	800
600	900
702	325
800	300

Now consider the following table, which specifies the Effective Address Obtained from the different Addressing Modes and the Contents of AC.

Addressing Mode	Effective	Content of AC

	Address	
Direct Address	500	800
Immediate Operand	200	500
Indirect Address	800	300
Relative Address	702	325
Indexed Address	600	900
Register	--	400
Register Indirect	400	700
Auto increment	400	700
Auto decrement	399	450

2.4. DATA TRANSFER INSTRUCTIONS

The Typical Data Transfer Instructions are shown in the table below.

Name	Mnemonic
Load	**LD**
Store	**ST**
Move	**MOV**
Exchange	**XCH**
Input	**IN**
Output	**OUT**
Push	**PUSH**
Pop	**POP**

The Data Transfer Instructions with Different Addressing Modes are shown in table below.

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

2.4.1. DATA MANIPULATION INSTRUCTIONS

The three Basic Types of instructions are

(1) Arithmetic instructions

(2) Logical and bit manipulation instructions

(3) Shift instructions

2.4.2. Arithmetic Instructions

The basic arithmetic operations are Add, Subtract, Increment, and Decrement etc. The following table shows the operation, its representation and description.

Operation	Representation	Description
Add	R3 ← R1 + R2	Contents of R1 and R2 are added and the result is transferred to R3
Subtract	R3 ← R1 – R2	Contents of R2 are subtracted from contents of R1 and the result is transferred to R3
1's Complement	$\bar{R}1$	Complement the content of R1
2's Complement	$\bar{R}1$ +1	Complement the contents of R1 and add 1 in it.
2's Complement subtraction	R3 ← R1 + $\bar{R}2$ + 1	Add R1 and the 2's Complement of R2
Increment	R1 ← R1 +1	Increment the contents of R1 by one
Decrement	R1 ← R1 –1	Decrement the contents of R1 by one

PROGRAM CONTROL

Program control in computer organization refers to how a computer sequences and directs the execution of instructions. This involves fetching instructions, decoding them, and then executing them in a specific order. Key aspects include the role of the control unit, the use of program counters, and various instructions that alter the flow of execution, like jumps and conditional branches.

Here's a more detailed breakdown:

1. Control Unit:

- The control unit is a crucial component of the CPU that manages the computer's operations.
- It fetches instructions from memory and translates them into control signals that direct other parts of the computer.
- It acts as the "conductor" of the computer, orchestrating the flow of data and operations.

2. Program Counter (PC):

- The PC is a special register that holds the memory address of the next instruction to be executed.
- After fetching an instruction, the PC is usually incremented to point to the subsequent instruction in sequence.

3. Program Control Instructions:

- These instructions alter the normal sequential flow of program execution.
- They allow the program to jump to different parts of the code, repeat sections (loops), or execute code conditionally based on certain criteria.

Examples of Program Control Instructions:

- **Branch Instructions:**

  These instructions change the value of the PC, causing the program to jump to a different memory address and execute instructions from that point.

- **Unconditional Branch:** Jumps to a specific address regardless of any conditions.
- **Conditional Branch:** Jumps to a specific address only if a certain condition is met (e.g., if a value is zero).
- **Loop Instructions:**

  These instructions are used to repeat a block of code multiple times, often with a counter to track the number of repetitions.

- **Subroutine Calls (and Returns):**

  These instructions allow a program to temporarily jump to a separate block of code (a subroutine or function), execute it, and then return to the original location.

- **Interrupts:**

  These are external events that can cause the program to temporarily halt and execute a special interrupt handler routine.

  In essence, program control is the mechanism that allows a computer to execute instructions in a non-linear fashion, enabling it to perform complex tasks by making decisions and repeating sections of code as needed.