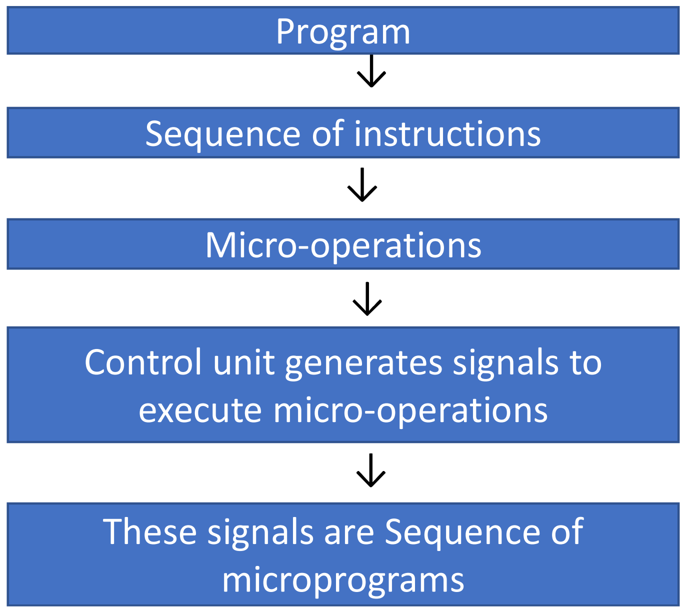
UNIT-2
Microprogrammed Control

- Control memory
- Address Sequencing
- Microprogram Example
- Design of Control Unit


Program
Sequence of instructions
Micro-operations
Control unit generates signals to execute micro-operations
These signals are Sequence of microprograms

- What is a microprogram?
- A sequence of micro instructions is a microprogram.

- What is a control unit?
- The function of the control unit in a digital computer is to initiate sequence of microoperations.
- Control unit can be implemented in two ways

✓ Hardwired control

✓ Microprogrammed control

- Hardwired Control:
- When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.
- The key characteristics are
- High speed of operation
- Expensive
- Relatively complex
- No flexibility of adding new instructions
- Microprogrammed Control:
- Control information is stored in control memory.
- Control memory is programmed to initiate the required sequence of micro-operations.
- The key characteristics are
- Speed of operation is low when compared with hardwired
- Less complex
- Less expensive
- Flexibility to add new instructions

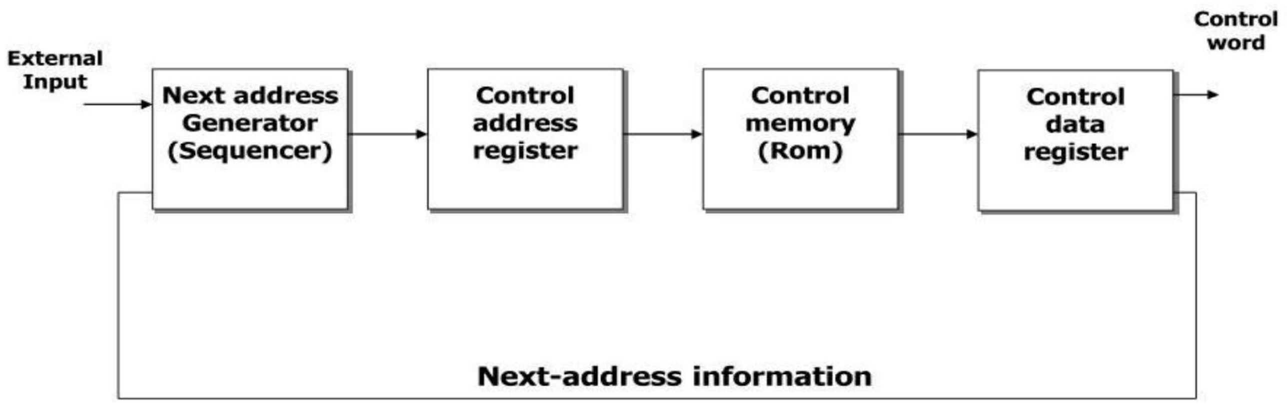
Control Memory

- The control function that specifies a microoperation is called as control variable.
- When control variable is in one binary state, the corresponding microoperation is executed.
- For the other binary state the state of registers does not change.
- The active state of a control variable may be either 1 state or the 0 state, depending on the application.
- Example;
- For bus-organized systems the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.

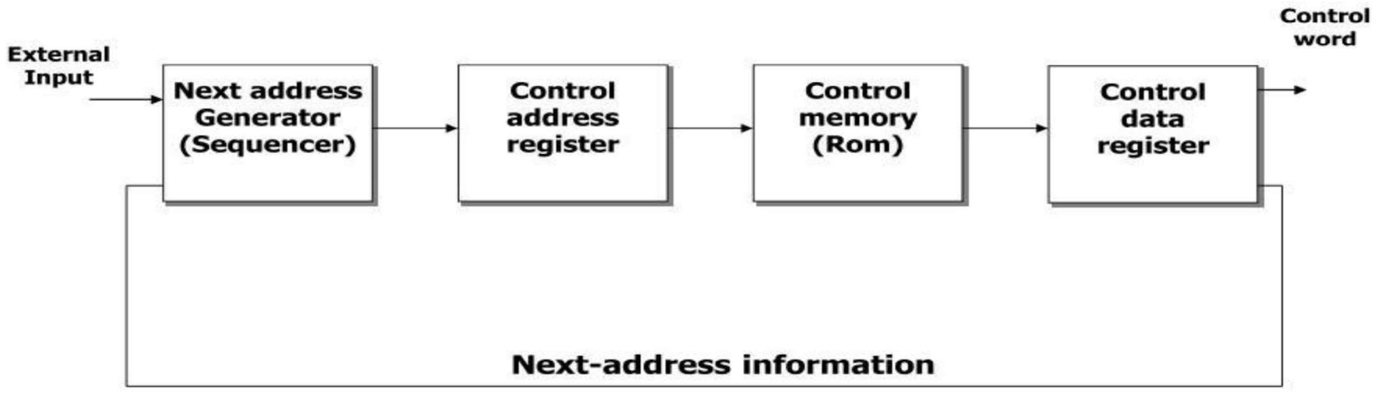
Control Word:

- The control variables at any given time can be represented by a string of 1’s and 0's called a control word.
- All control words can be programmed to perform various operations on the components of the system.

- Microprogram control unit:
- A control unit whose binary control variables are stored in memory is called a microprogram control unit.
- The control word in control memory contains within it a microinstruction.
- The microinstruction specifies one or more micro-operations for the system.
- A sequence of microinstructions constitutes a microprogram.
- The control unit consists of control memory used to store the microprogram.
- Control memory is a permanent i.e., read only memory (ROM).
- The general configuration of a micro-programmed control unit organization is shown as block diagram below.



- The control memory is ROM so all control information is permanently stored.
- The control memory address register (CAR) specifies the address of the microinstruction and the control data register (CDR) holds the microinstruction read from memory.
- The next address generator is sometimes called a microprogram sequencer. It is used to generate the next micro instruction 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.
- So it is necessary to use some bits of the present microinstruction to control the generation of the address of the microinstruction.
- Sometimes the next address may also be a function of external input conditions.
- The control data register holds the present microinstruction while next address is computed and read from memory.
- The data register is sometimes called a pipeline register.



- A computer with a microprogrammed control unit will have two separate memories:
- a main memory (RAM)
- control memory (ROM)
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:
- ▪ fetch the instruction from main memory
- ▪ evaluate the effective address
- ▪ execute the operation
- ▪ return control to the fetch phase for the next instruction

Addressing sequence

- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- Each computer instruction has its own microprogram routine to generate the microoperations.
- 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

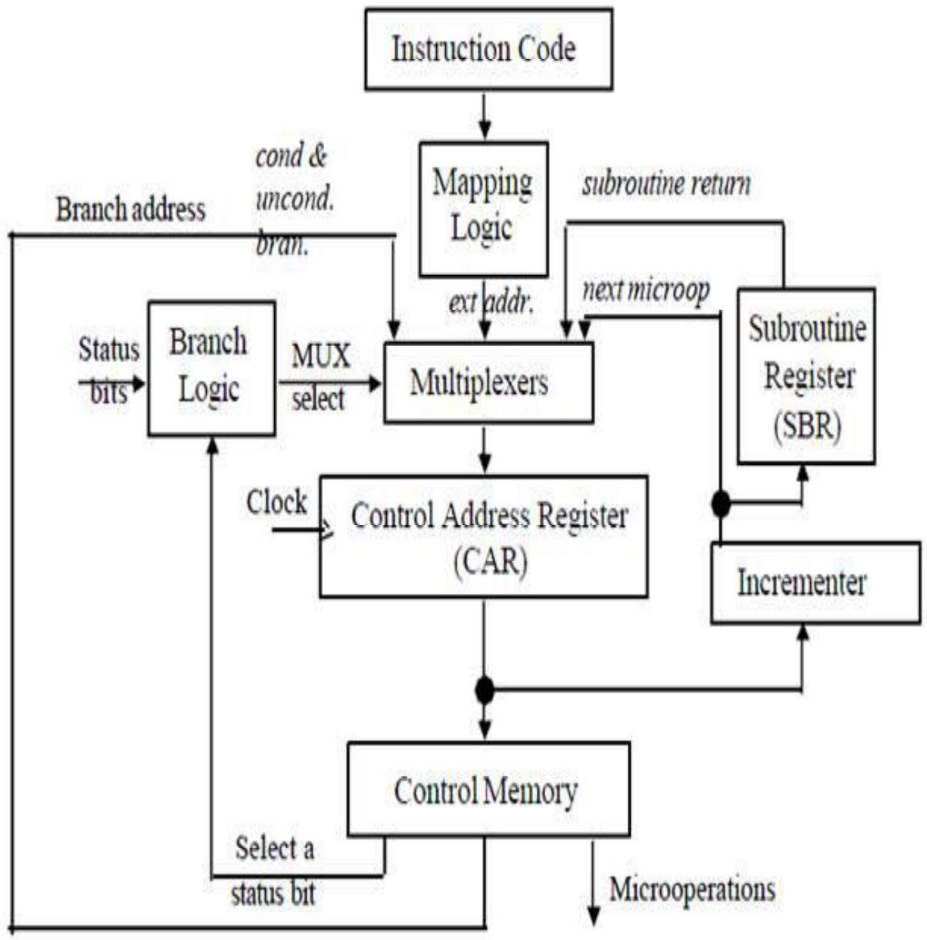
➢ Steps the control must undergo during the execution of a single computer instruction are as follows

- Initial address is loaded into the control address register(CAR) when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- At the end of the fetch routine instruction is placed in the instruction register- IR
- The control memory then goes through the routine to determine the effective address of the operand with the help of mode bits and branch micro instructions
- At the end of this routine Address register AR holds operand address
- The next step is to generate the microoperations that execute the instruction fetched from memory by considering the opcode and applying a mapping process.
- The transformation of the instruction code bits to an address in control memory where the routine of instruction located is referred to as mapping process.
- After execution, control must return to the fetch routine by executing an unconditional branch

- In brief 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.

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.

In the figure four different paths form which the control address register (CAR) receives the address.

✓The incrementer increments the content of the control register 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.

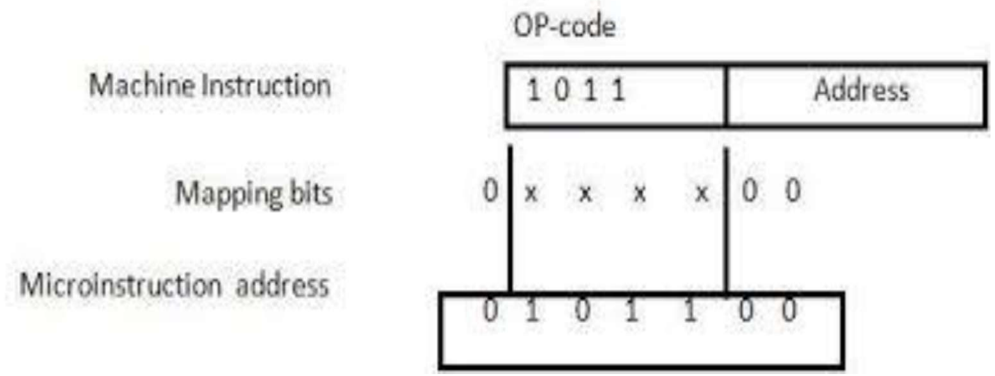
✓The return address for a subroutine is stored in a special register, that value is used when the micoprogram wishes to return from the subroutine.

Conditional branching

- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- 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 i/o status conditions.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the branch logic.
- The branch logic tests the condition, if met then branches, otherwise, increments the CAR.
- Conditional branching can be implemented with a multiplexer. If there are 8 status bit conditions, then 3 bits in the microinstruction are used to specify any one of the condition and they provide the selection variables for the multiplexer.
- If the selected status bit is in 1 state, the output of multiplexer is 1, otherwise it is 0.
- A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.
- A 0 output in the multiplexer causes the address register to be incremented.
- For unconditional branching, fix the value of one status bit to be 1.
- Reference to this bit causes the branch address to be loaded into the control address register unconditionally.

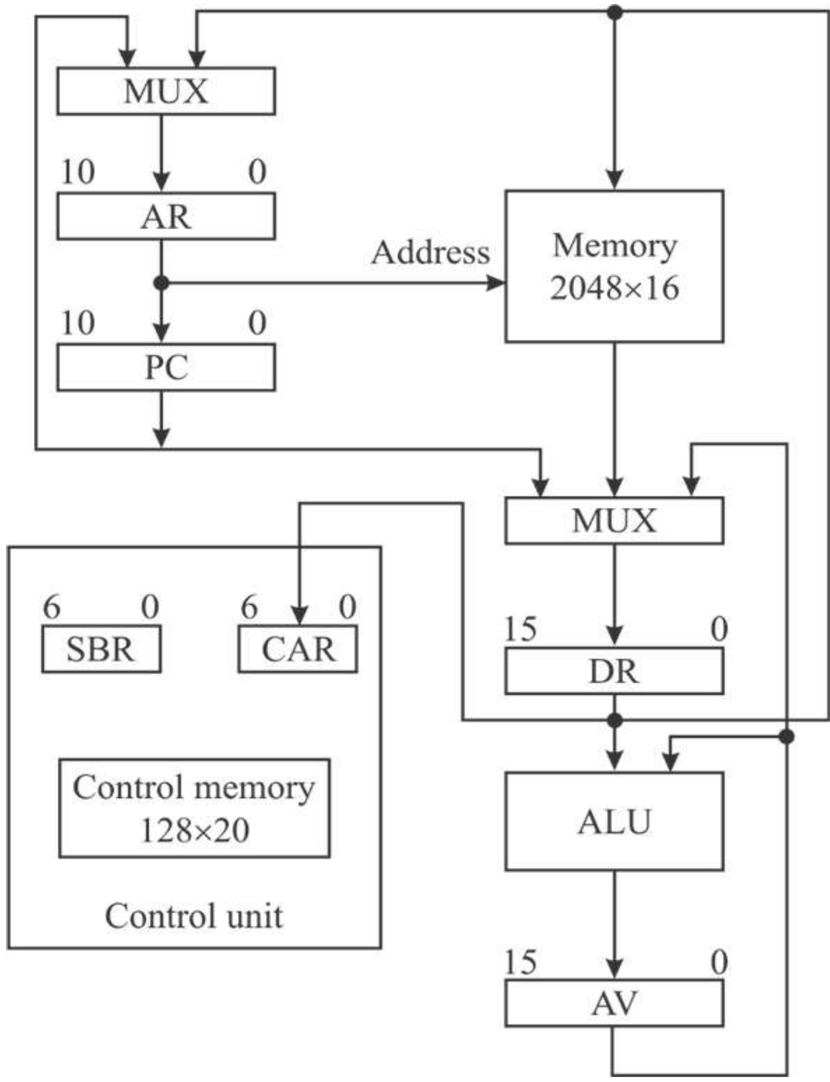
Mapping of instructions

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located.
- The status bits for this type of branch are the bits in the opcode.
- Assume an opcode of four bits and a control memory of 128 locations. The mapping process converts the 4-bit opcode to a 7-bit address for control memory shown in below figure.
- 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.
- The mapping function is implemented by integrated circuit called programmable logic device

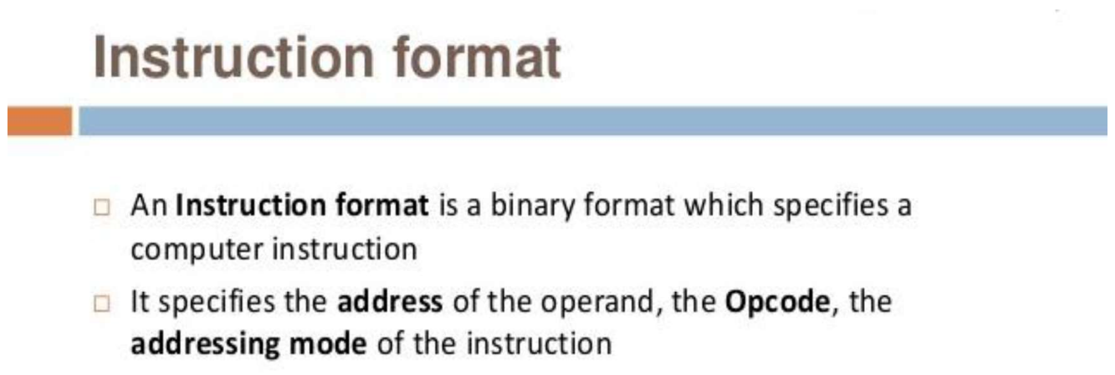


Microprogram example

- The process of code generation for the control memory is called microprogramming.
- The block diagram of the computer configuration is shown in the 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.
- DR receives information from AC,PC or memory.
- AR can receive information from PC or DR

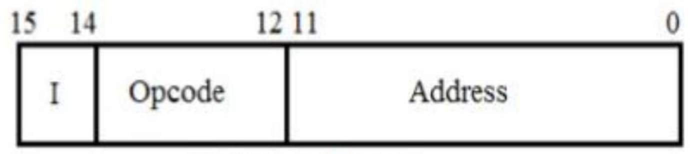


Computer instruction format



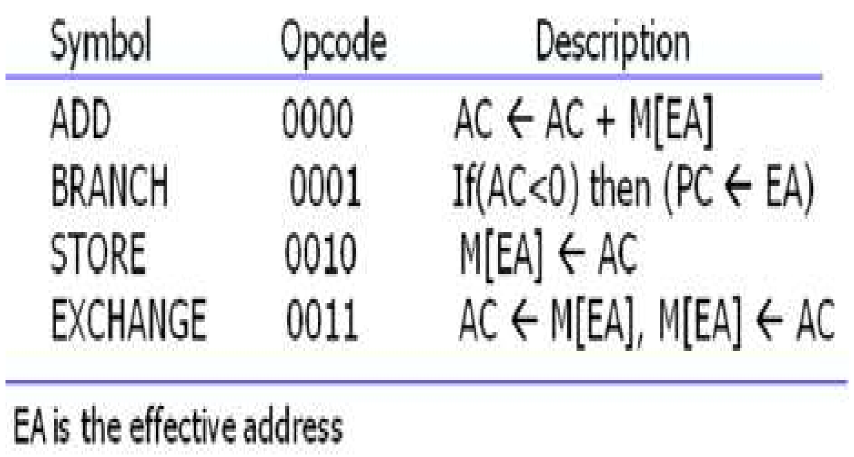
Instruction format

- An **Instruction format** is a binary format which specifies a computer instruction
- It specifies the **address** of the operand, the **Opcode**, the **addressing mode** of the instruction



Instruction Format

- Three fields for an instruction:
1. 1-bit field for direct/indirect addressing
2. 4-bit opcode
3. 11-bit address field



Symbol	Opcode	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

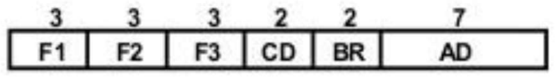
EA is the effective address

Micro instruction format

The microinstruction format is composed of 20 bits divided into four parts

- Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]
- The CD field selects status bit conditions [2 bits]
- The BR field specifies the type of branch to be used [2 bits]
- The AD field contains a branch address [7 bits] because control memory has 128 words
- Each of the three microoperation fields can specify one of seven possibilities.
- No more than three microoperations can be chosen for a microinstruction.
- If fewer than three are needed, the code 000 = NOP.
- The three bits in each field are encoded to specify seven distinct microoperations listed in below table.
- The condition field (CD) is two bits to specify four status bit conditions .
- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

Microinstruction Format



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

Symbols and Binary Code for Microinstruction Fields

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 $\vee$ DR	OR
011	AC ← AC $\wedge$ 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 $\oplus$ 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	

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

Symbolic microinstructions

- Different symbols can be used to construct the micro instructions in symbolic form.
- Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts

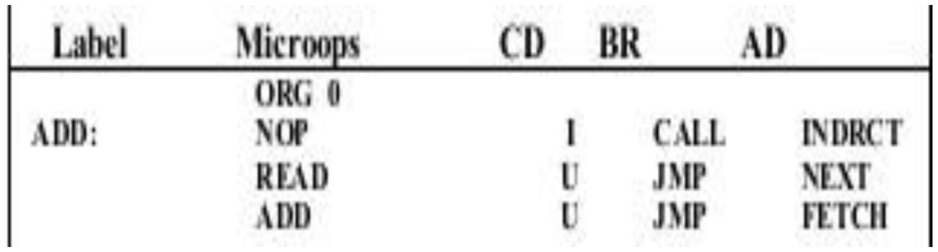
➢ Lable

➢ Microoperations

➢ CD

➢ BR

➢ AD



Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH

1. The label field may be empty or it may specify a symbolic address. Terminate with a colon (:).

2. The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros

3. The condition field specifies one of the four conditions U,I,S,Z.

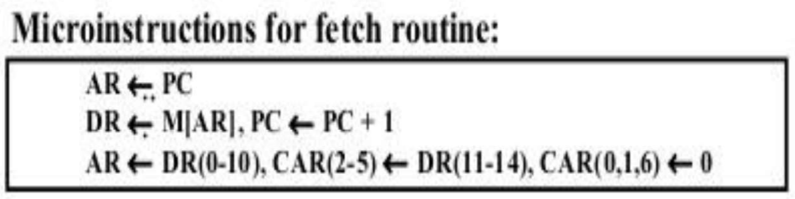
4. The branch field has one of the four branch symbols JMP,CALL,RET,MAP

5. The address field has three formats

- a. A symbolic address – must also be a label
- b. The symbol NEXT to designate the next address in sequence
- c. Empty if the branch field is RET or MAP and is converted to 7 zeros
- The symbol ORG defines the origin i;e the first address of a microprogram routine.
- Eg; ORG 64 – places first microinstruction at control memory 1000000 which is equivalent to decimal number 64.

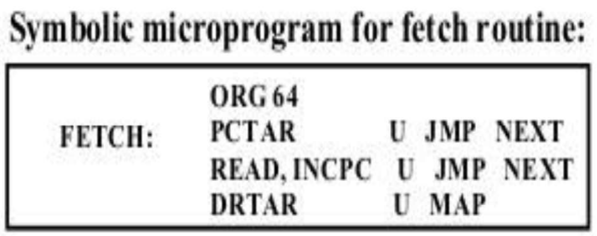
Fetch routine

- The control memory has 128 locations, each one is 20 bits.
- The first 64 locations are occupied by the routines for the 16 instructions, addresses 0-63.
- the fetch routine starts at address 64.
- The fetch routine requires the following three microinstructions at locations 64-66.
- The microinstructions needed for fetch routine are:
- The address of instruction is transferred from PC to AR and the instruction is read from memory into DR and PC is incremented.
- The address part is transferred to AR and the control is transferred to one of 16 routines by mapping the operation code part of the instruction from DR into CAR.
- Using assembly language conventions like above we can write symbolic micro programs as shown in the table.



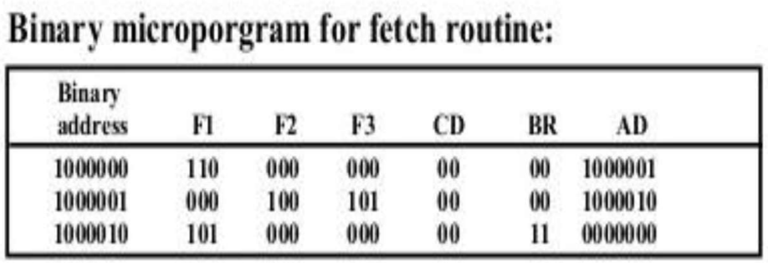
Microinstructions for fetch routine:

$AR \leftarrow PC$
$DR \leftarrow M[AR], PC \leftarrow PC + 1$
$AR \leftarrow DR(0\text{-}10), CAR(2\text{-}5) \leftarrow DR(11\text{-}14), CAR(0,1,6) \leftarrow 0$



Symbolic microprogram for fetch routine:

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



Binary microporgram for fetch routine:

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

Symbolic Microprogram

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

Partial Symbolic Microprogram

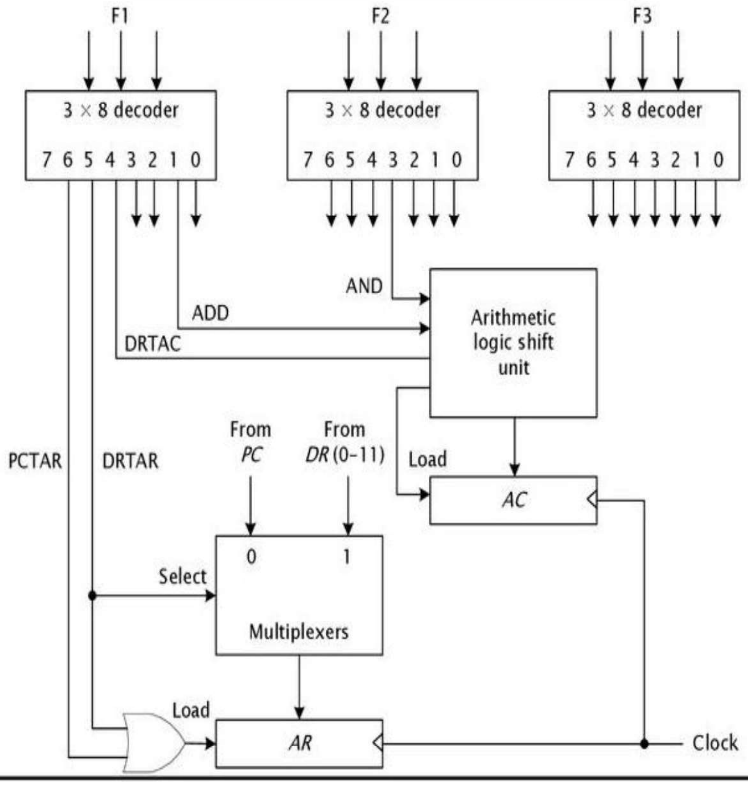
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	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	

Design of control unit

- The control memory out 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 figure shows the three decoders and some of the connections that must be made from their outputs.
- The three fields of the microinstruction in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each of the output must be connected to proper circuit to initiate the corresponding microoperation.
- 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 Fig, 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.

Decoding of *F* fields

Fig. 7-7 Decoding of microoperation fields



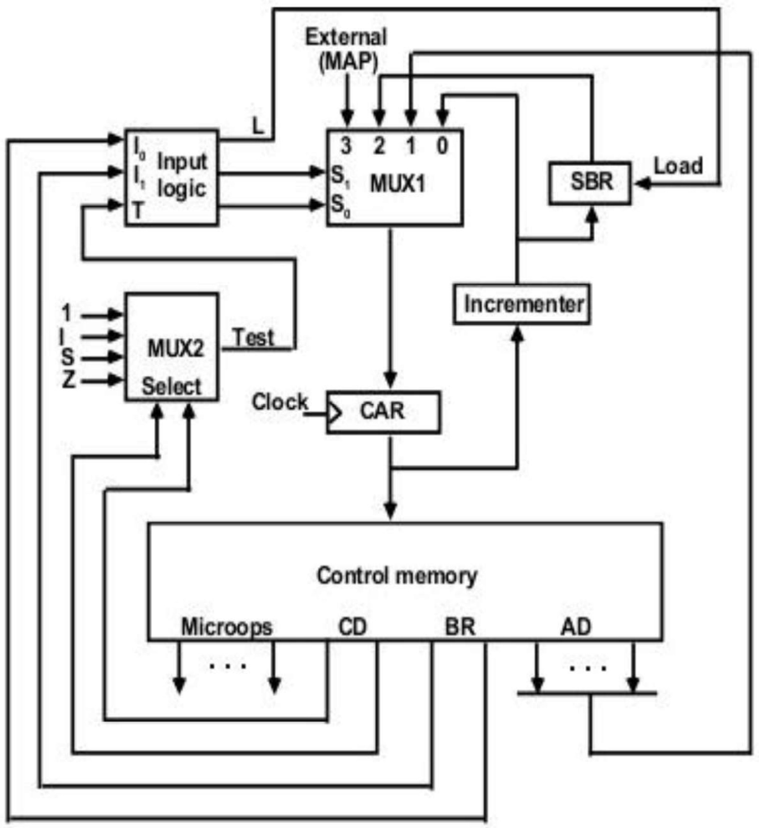
Microprogram sequencer

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection 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 to be loaded into the control address register.
- The block diagram of the microprogram sequencer is shown in the figure.
- The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.
- There are two multiplexers in the circuit.

1. The first multiplexer selects an address from one of four sources and routes it into control address register CAR.
2. 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.

Microprogram Sequencer

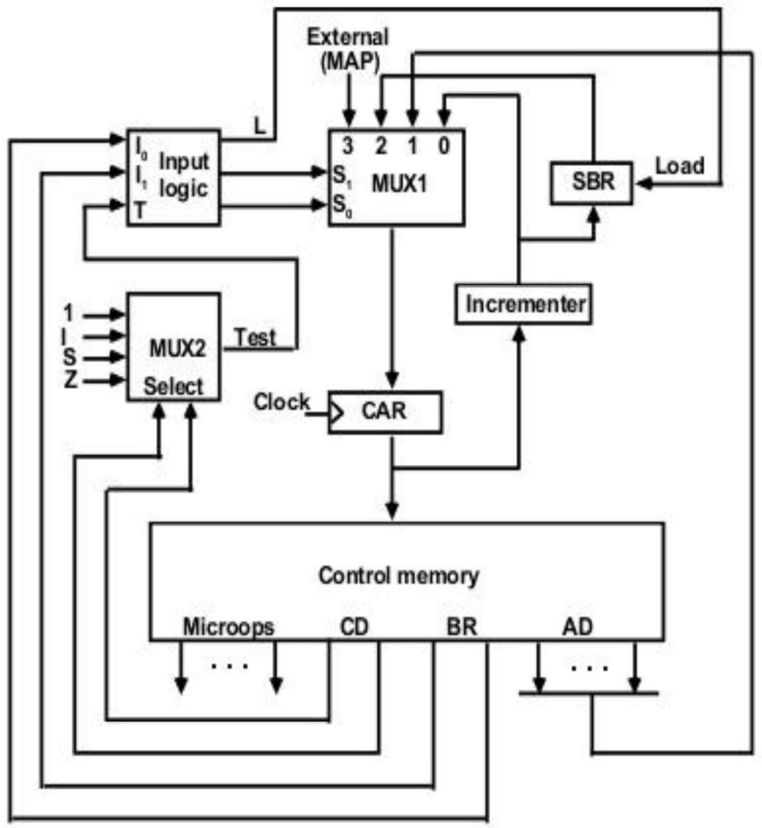


Microprogram Sequencer

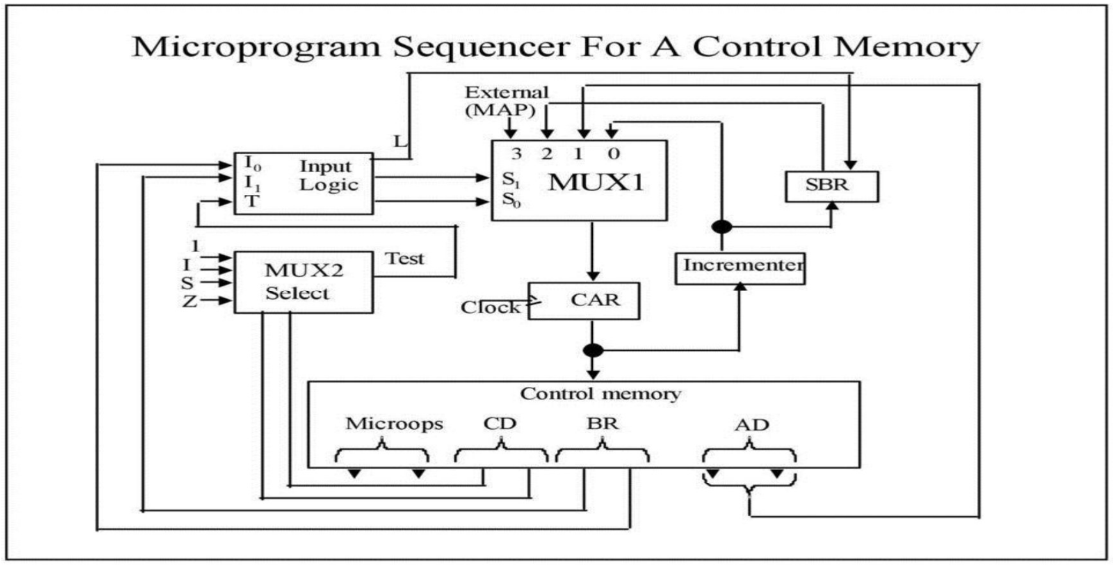
- The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR.
- The other three inputs to multiplexer come from

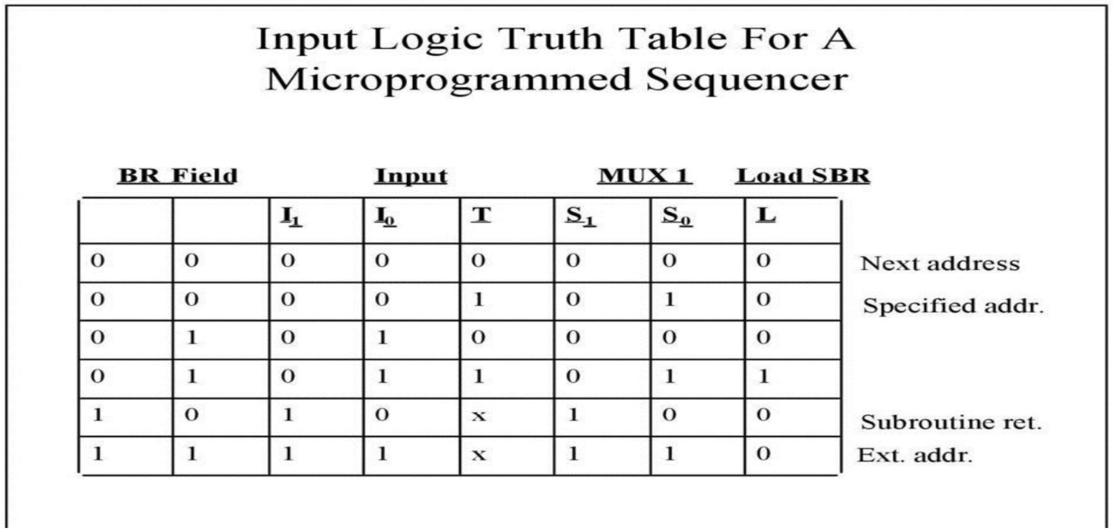
1. The address field of the present microinstruction
2. From the out of SBR
3. From an external source that maps the instruction

- 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.
- The input logic circuit in above figure has three inputs I0, I1, and T, and three outputs, S0, S1, and L.
- Variables S0 and S1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the selection variables determine the path in the multiplexer.
- For example, with S1,S0 = 10, multiplexer input number 2 is selected and establishes transfer path from SBR to CAR.



- Inputs I1 and I0 are identical to the bit values in the BR field.
- The bit values for S1 and S0 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:





Input Logic Truth Table For A Microprogrammed Sequencer

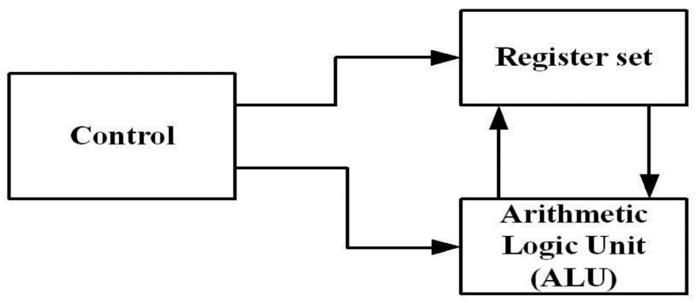
BR Field		Input			MUX 1		Load SBR	
		I_1	I_0	T	S_1	S_0	L	
0	0	0	0	0	0	0	0	Next address
0	0	0	0	1	0	1	0	Specified addr.
0	1	0	1	0	0	0	0	
0	1	0	1	1	0	1	1	
1	0	1	0	x	1	0	0	Subroutine ret.
1	1	1	1	x	1	1	0	Ext. addr.

Central Processing Unit

- The main part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.
- The CPU is made up of three major parts, as shown in Fig

1. The register set stores intermediate data used during the execution of the instructions.
2. The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions.
3. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

Central Processing Unit

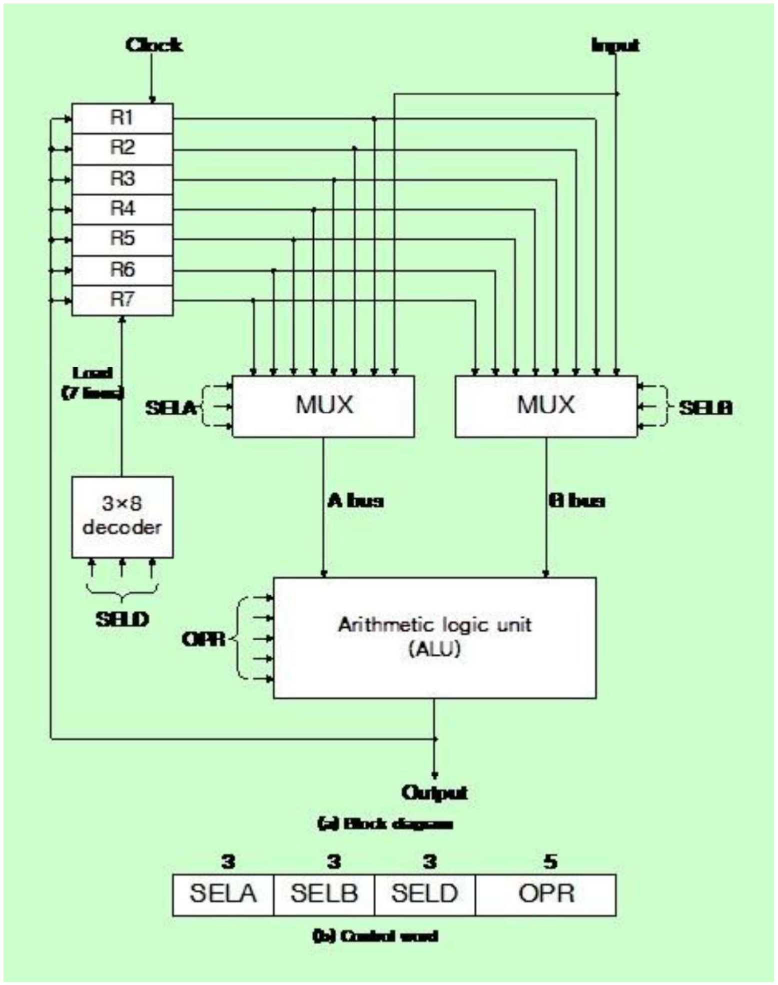


Major components of CPU

General register organization

- Memory locations are needed for storing pointers, counters, return address, temporary results etc.
- Referring to these memory locations is very time consuming because memory access is the most time consuming operation in a computer.
- Therefore it is convenient to store these intermediate values in processor registers.
- When there are many registers in the system they are connected through a common bus system.
- The registers communicate with each other for data transfer as well as for performing some micro operations.
- Hence it is necessary to provide a common unit that performs arithmetic, logic and shift operations in the processor.

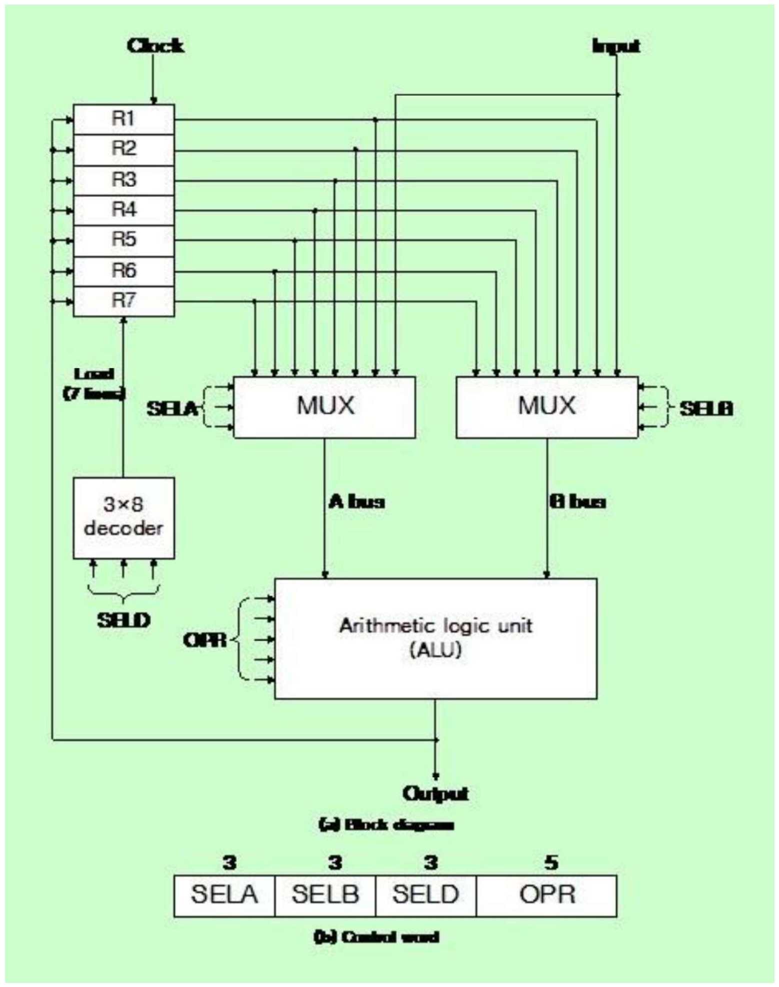
- A bus organization for 7 CPU registers is shown in the figure.
- The outputs of each register is connected to the two multiplexers(MUX) to form the two buses A and B.
- The selection lines in each multiplexer select one register or the input data for the particular bus.
- The A and B buses form the inputs to a common arithmetic logic unit (ALU).
- The operation selected in the ALU determines the arithmetic or logic micro operation that is to be performed.
- The result of micro operation goes into the inputs of all the registers.
- The register that receives the information is selected by a decoder.
- The decoder activates one of the register load inputs, thus providing transfer path between the data in the output bus and the inputs of the selected destination register.
- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.



- Eg; to perform the following operation.
- $R_1 \leftarrow R_2 + R_3$

The control must provide binary selection variables to the following selector inputs.

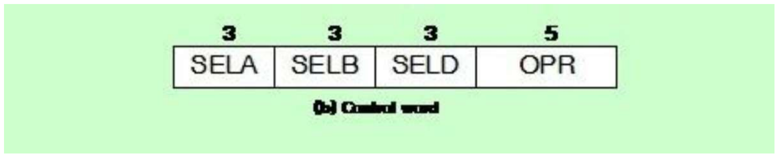
1. MUX A selector (SELA): to place the contents of R_2 into bus A.
2. MUX B selector (SELB) : to place the content of R_3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A+B.
4. Decoder destination selector(SELD): to transfer the content of the output bus into R1.



- The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle.
- The data from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and onto the input of the destination register, all during the clock cycle interval.
- Then , when the next clock transition occurs, the binary information from the output bus is transferred into R_1.
- To achieve a fast response time, the ALU is constructed with high-speed circuits.
- The buses are implemented with multiplexers or three-state gates

Control word

- Control word is defined as a word whose individual bits represent various control signals.
- There are 14 selection inputs in the unit, and their combined value specifies a control word.
- The 14 bit control word is defined in the following fig, it consists of 4 fields.



- Three fields contain 3 bits each and last field contains 5 bits.
- The three bits of SELA select a source register for the A input of the ALU.
- The three bits of SELB select a source register for the B input of the ALU.
- The three bits of SELD select a destination register using the decoder and its seven load outputs.
- The five bits of OPR select one of the operations in the ALU.
- The 14 bit control word when applied to the selection inputs specify a particular microoperation.

- The encoding of the register selections is specified in table
- **The 3-bit binary code** listed in the first column of the table specifies the binary code for each of the three fields.

The register selected by fields SELA, SELB, and SELD is the one whose decimal number is equivalent to the binary number in the code. When SELA or SELB is 000, the corresponding multiplexer selects the external input data.

When SELD = 000, no destination register is selected but the contents of the output bus are available in the external output.

The ALU provides arithmetic and logic operations.

The CPU must also provide shift operations. The shifter may be placed in the input of the ALU to provide a preshift capability, or at the output of the ALU to provide postshifting capability.

In some cases, the shift operations are included with the ALU.

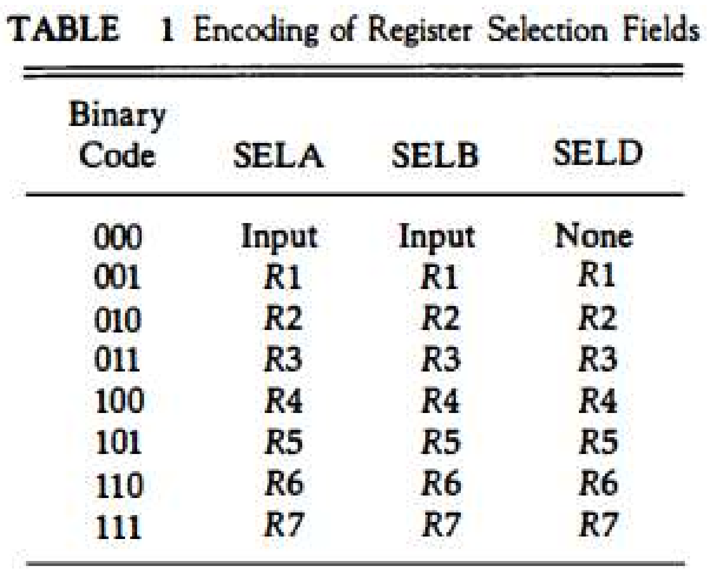
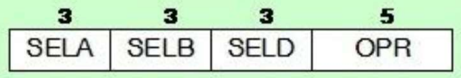


TABLE 1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7



(b) Control word

- The encoding of the ALU operations for the CPU is shown in the following table.
- The OPR field has 5 bits and each operation is designated with a symbolic name.
- **Examples of microoperations**
- **A control word of 14 bits** is needed to specify a microoperation in the CPU. The control word for a given microoperation can be derived from the selection variables.

For example, the subtract microoperation given by the statement

R1 ← R2 - R3

- specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A - B.

Thus the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding listed in Tables 1 and 2.

The binary control word for the subtract microoperation is

010 011 001 00101 and is obtained as follows:

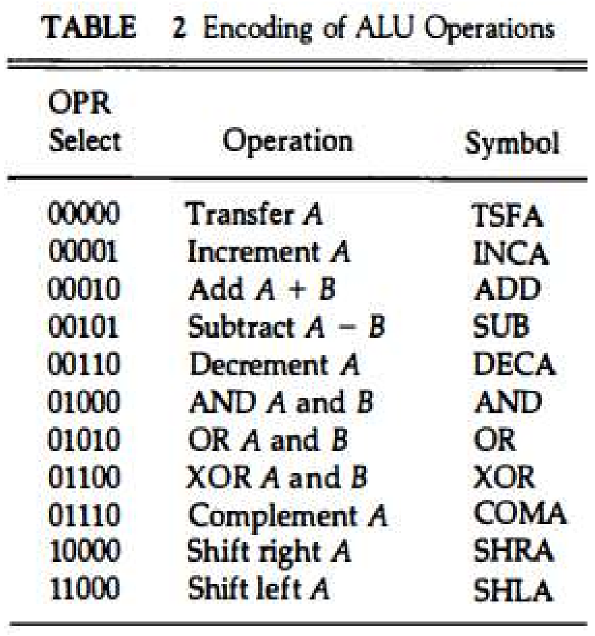
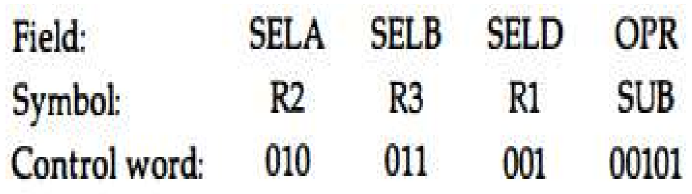


TABLE 2 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA



Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

- **The control word** for this microoperation and a few others are listed in Table 3.
- **The increment and transfer microoperations** do not use the B input of the ALU.
- **For these cases**, the B field is marked with a dash. We assign 000 to any unused field when formulating the binary control word, although any other binary number may be used.
- **To place the content of a register** into the output terminals we place the content of the register into the A input of the ALU, but none of the registers are selected to accept the data.
- **The ALU operation TSFA** places the data from the register, through the ALU, into the output terminals.
- **The direct transfer** from input to output is accomplished with a control word **of all 0's (making the B field 000).**
- **A register** can be cleared to 0 with an exclusive-OR operation. This is because x $\oplus$ x = 0.
- **It is apparent** from these examples that many other microoperations can be generated in the CPU.
- **The most efficient way** to generate control words with a large number of bits is to store them in a memory unit.
- **A memory unit** that stores control words is referred to as a control memory.
- **By reading consecutive control words from memory**, it is possible to initiate the desired sequence of microoperations for the CPU.
- **This type of control** is referred to as microprogrammed control.

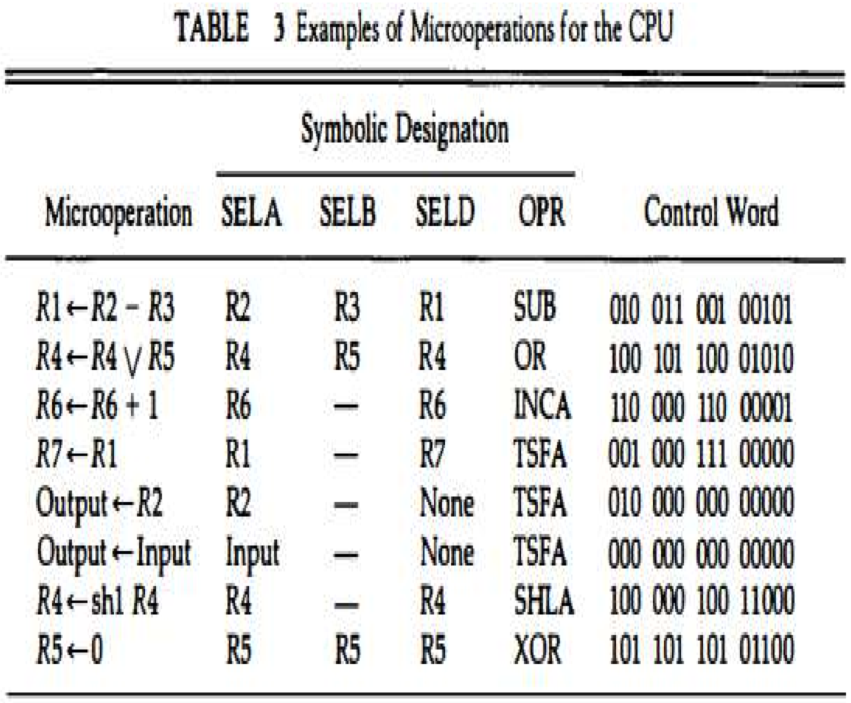


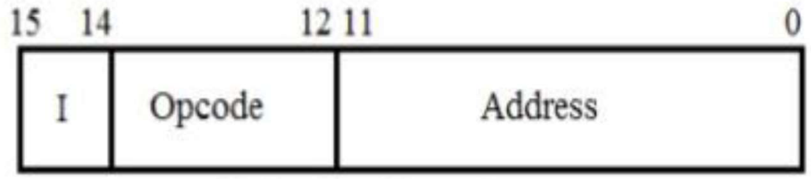
TABLE 3 Examples of Microoperations for the CPU

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Instruction formats

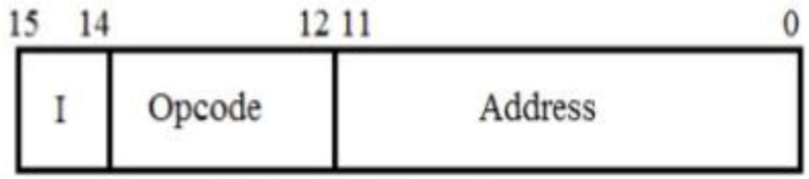
- An instruction is a group of bits that instructs the computer to do some operation. These bits are arranged in some instruction code format.
- Control unit in the CPU will interpret each instruction code and provide the necessary control functions needed to process the instruction.
- A computer will usually have a variety of instruction code formats.
- The format of an instruction is represented 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 groups called fields.
- The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be perform
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.



Instruction Format

- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruct format of a computer depends on the internal organization of its registers.
- Most computers have one of three types of CPU organizations:
- 1. Single accumulator organization.
- 2. General register organization.
- 3. Stack organization.

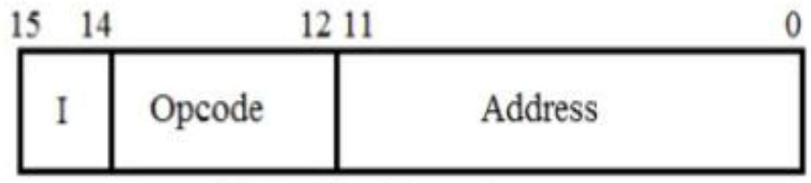


Instruction Format

- Single Accumulator Organization:
- In an accumulator type organization all the operations are performed with an implied accumulator register.
- The instruction format in this type of computer uses one address field.
- For example, the instruction that specifies an arithmetic addition defined by an assembly language instruction as

  ADD X here X is the address of the operand.
- The ADD instruction in this case results in the operation

  $AC \leftarrow AC + M[X]$.
- AC is the accumulator register and M[X] symbolizes the memory word located at address X



Instruction Format

General register organization:

- The instruction format in this type of computer needs three register address fields.
- Eg 1; the instruction for an arithmetic addition may be written in an assembly language as

ADD R1, R2, R3

This denote the operation $R1 \leftarrow R2 + R3$.

- The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
- Eg2; Thus the instruction ADD R1, R2 would denote the operation

$R1 \leftarrow R1 + R2$.

Only register addresses for R1 and R2 need be specified in this instruction.

- General register-type computers employ two or three address fields in their instruction format.
- Each address field may specify a processor register or a memory word.
- An instruction symbolized by ADD R1, X would specify the operation $R1 \leftarrow R1 + M[X]$.
- It has two address fields, one for register R1 and the other for the memory address X.

Stack organization:

- The stack-organized CPU has PUSH and POP instructions which require an address field.
- Thus the instruction PUSH X will push the word at address X to the top of the stack.
- The stack pointer is updated automatically.
- Operation-type instructions do not need an address field in stack-organized computers.
- This is because the operation is performed on the two items that are on top of the stack.
- The instruction ADD in a stack computer consists of an operation code only with no address field.
- This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack.
- There is no need to specify operands with an address field since all operands are implied to be in the stack.

- Most computers fall into one of the three types of organizations.
- Some computers combine features from more than one organizational structure.

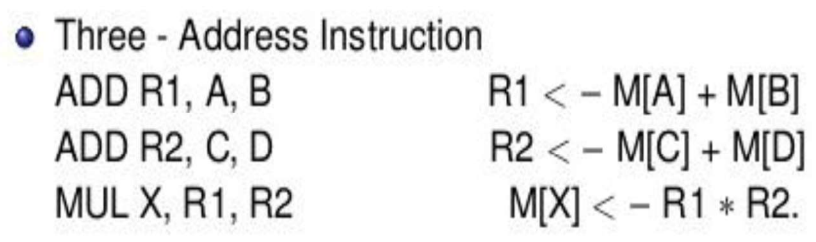
- To illustrate The influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

$$X= (A+B) * (C+D)$$

- using zero, one, two, or three address instructions.
- using the symbols ADD, SUB, MUL and DIV for four arithmetic operations.
- MOV for the transfer type operations;
- LOAD and STORE for transfer to and from memory and AC register.
- Assuming that the operands are in memory addresses A, B, C, and D and the result must be stored in memory at address X and also the CPU has general purpose registers R1, R2, R3 and R4.

Three Address Instructions:

- Three-address instruction formats can use each address field to specify either a processor register or a memory operand.
- The program assembly language that evaluates X = (A+B) * (C+D) is shown below, together with comments that explain the register transfer operation of each instruction.



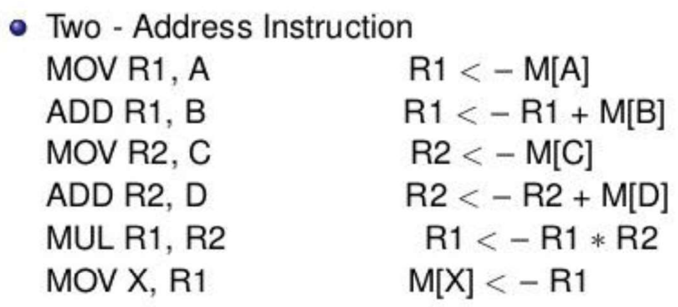
- Three - Address Instruction

```
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.
```

- The symbol M [A] denotes the operand at memory address symbolized by A.
- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.
- The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

- Two Address Instructions:

- Two-address instructions formats use each address to specify either a processor register or memory word.
- The program to evaluate X = (A+B) * (C+D) is as follows

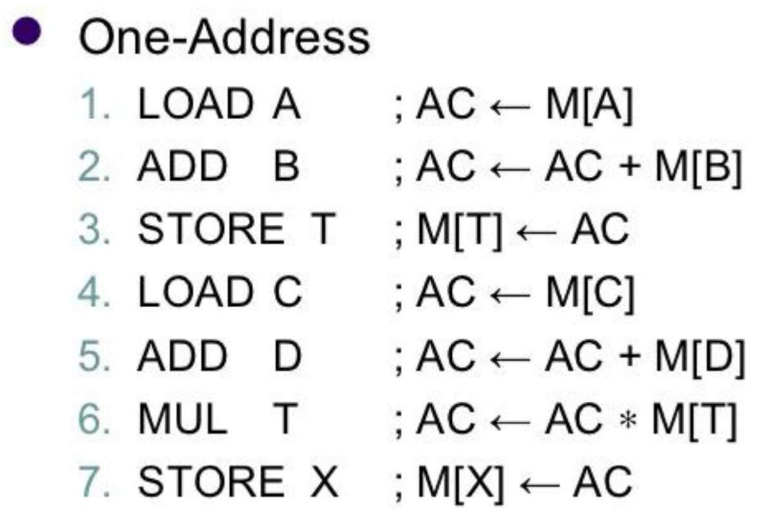


- Two - Address Instruction

```
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
```

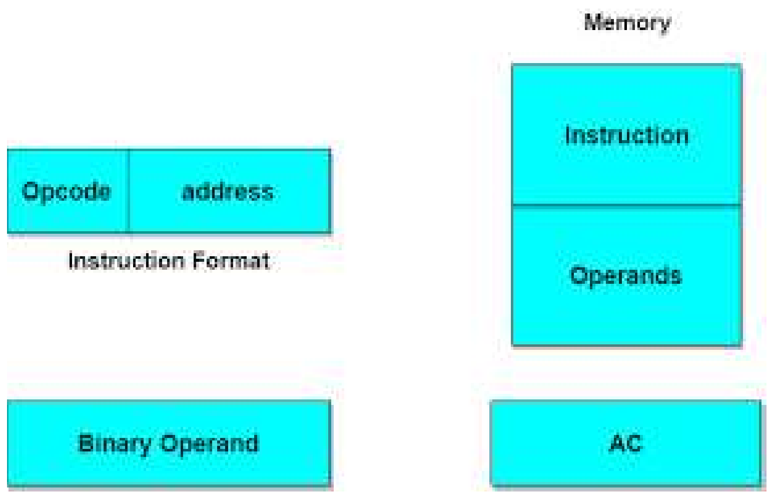
- The MOV instruction moves or transfers the operands to and from memory and processor registers.
- The first symbol listed in an instruction is assumed be both a source and the destination where the result of the operation transferred.

- One Address Instructions:
- One-address instructions use an implied accumulator (AC) register for all data manipulation.
- AC contains the result of all operations.
- The program to evaluate X=(A+B) * (C+D) is



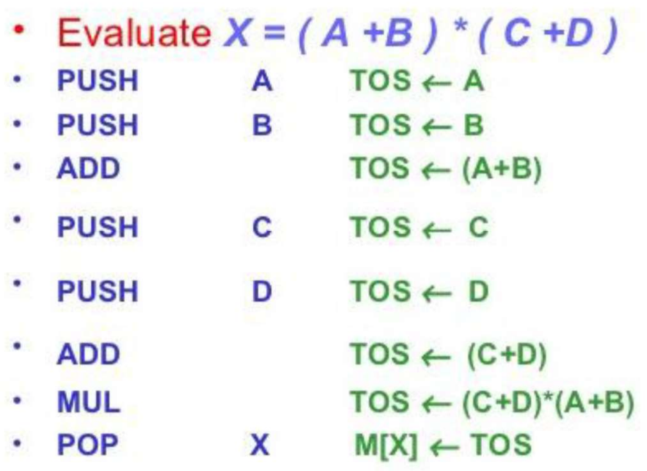
- One-Address

```
1. LOAD A      ; AC ← M[A]
2. ADD   B     ; AC ← AC + M[B]
3. STORE  T    ; M[T] ← AC
4. LOAD C      ; AC ← M[C]
5. ADD   D     ; AC ← AC + M[D]
6. MUL   T     ; AC ← AC * M[T]
7. STORE  X    ; M[X] ← AC
```



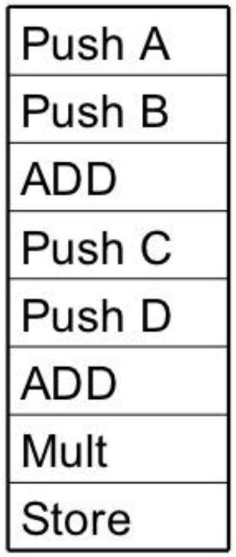
- All operations are done between the AC register and a memory operand.
- T is the address of a temporary memory location required for storing the intermediate result.

- Zero Address Instructions:
- A stack-organized computer does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
- The following program shows how X = (A+B) * (C+D) will be written for a stack-organized computer. (TOS stands for top of stack).



- Evaluate $X = (A + B) * (C + D)$

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 ← (C+D)*(A+B)
POP	X	M[X] ← TOS



Push A
Push B
ADD
Push C
Push D
ADD
Mult
Store

Addressing modes

- Operands are chosen during program execution depending on the addressing mode of the instruction.
- Computers use addressing mode techniques to

1. To provide facilities such 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

- Types of addressing modes
- Implied Mode
- Immediate Mode
- Register Mode
- Register Indirect Mode
- Autoincrement or Autodecrement Mode
- Direct Address Mode
- Indirect Address Mode
- Relative Address Mode
- Indexed Addressing Mode
- Base Register Addressing Mode

- Most addressing modes modify the address field of the instruction; there are two modes that need no address field at all. These are ***implied and immediate modes***

➢Implied Mode:

- In this mode the operands are specified 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.
- All register reference instructions that use an accumulator are implied mode instructions.
- Zero address in a stack organization computer is implied mode instructions.

➢Immediate Mode:

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

- Register Mode:
- When the address specifies a processor register, the instruction is said to be in the 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.

Register Indirect Mode:

- In this mode the instruction specifies a register in 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.
- The advantage of a register indirect mode instruction is that the address field of the instruction uses few bits to select a register than would have been required to specify a memory address directly.

Auto-increment or Auto-Decrement Mode:

- This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.

- The basic two mode of addressing used in CPU are direct and indirect address mode.

- Direct Address Mode:
- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- In a branch-type instruction the address field specifies the actual branch address.

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.
- A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU.
- The effective address in these modes is obtained from the following computation:
- Effective address =address part of instruction + content of CPU register
- The CPU register used in the computation may be the program counter, an index register, or a base register.
- We have a different addressing mode which is used for a different application.

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.

Indexed Addressing Mode:

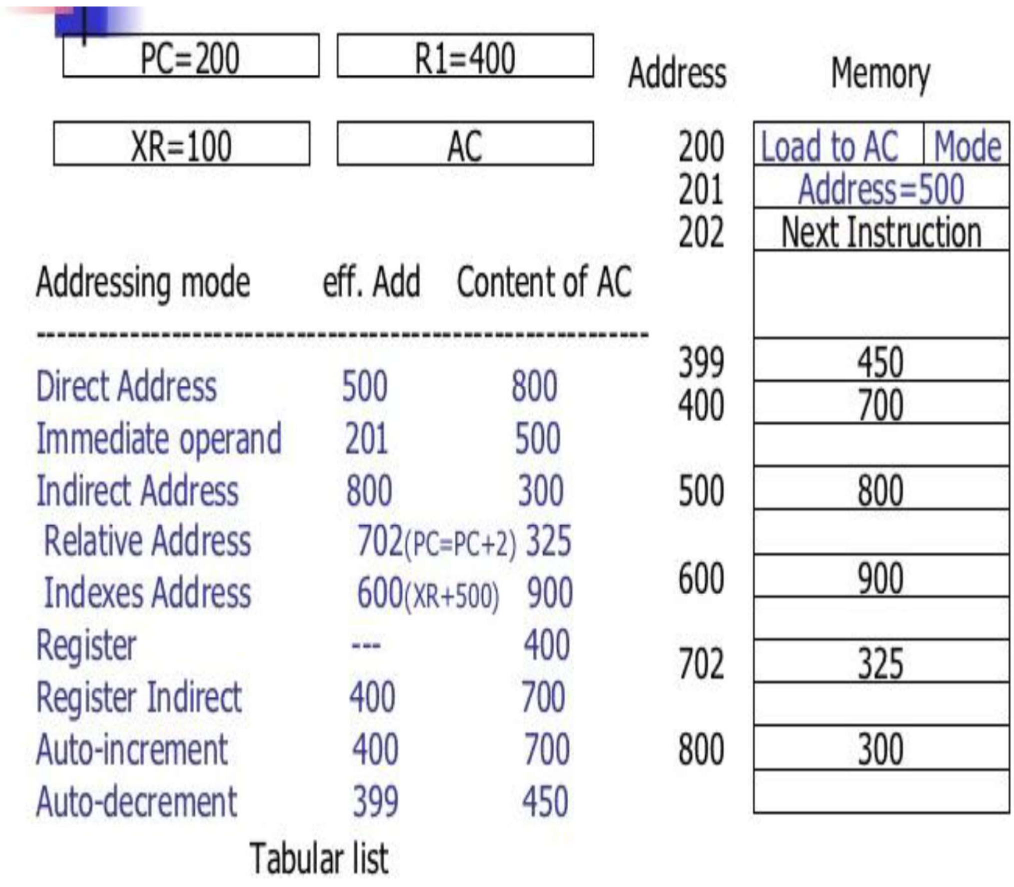
- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- An index register is a special CPU register that contains an index value.

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.

Example

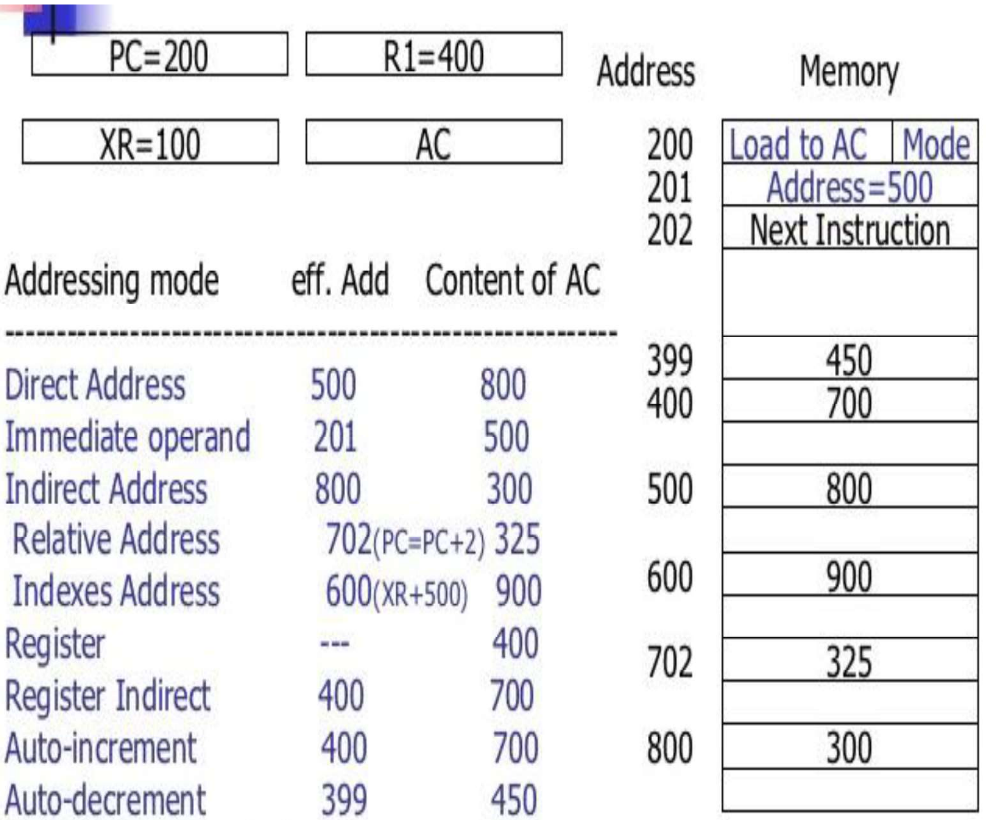
- To show the differences between the various modes, we will show the effect of the addressing modes on the instruction defined in Fig
- The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500.
- The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.
- PC has the value 200 for fetching this instruction. The content of processor register R1 is 400, and the content of an index register XR is 100.
- AC receives the operand after the instruction is executed
- In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.
- In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC
- In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.



Addressing mode	eff. Add	Content of AC
Direct Address	500	800
Immediate operand	201	500
Indirect Address	800	300
Relative Address	702(PC=PC+2)	325
Indexes Address	600(XR+500)	900
Register	---	400
Register Indirect	400	700
Auto-increment	400	700
Auto-decrement	399	450

Tabular list

- In the relative mode the effective address is 500 + 202 =702 and the operand is 325. (the value in PC after the fetch phase and during the execute phase is 202.)
- In the index mode the effective address is XR+ 500 = 100 + 500 = 600 and the operand is 900.
- In the register mode the operand is in R1 and 400 is loaded into AC.
- In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700.
- The auto-increment mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction.
- The auto-decrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.



Addressing mode	eff. Add	Content of AC
Direct Address	500	800
Immediate operand	201	500
Indirect Address	800	300
Relative Address	702(PC=PC+2)	325
Indexes Address	600(XR+500)	900
Register	---	400
Register Indirect	400	700
Auto-increment	400	700
Auto-decrement	399	450

Tabular list

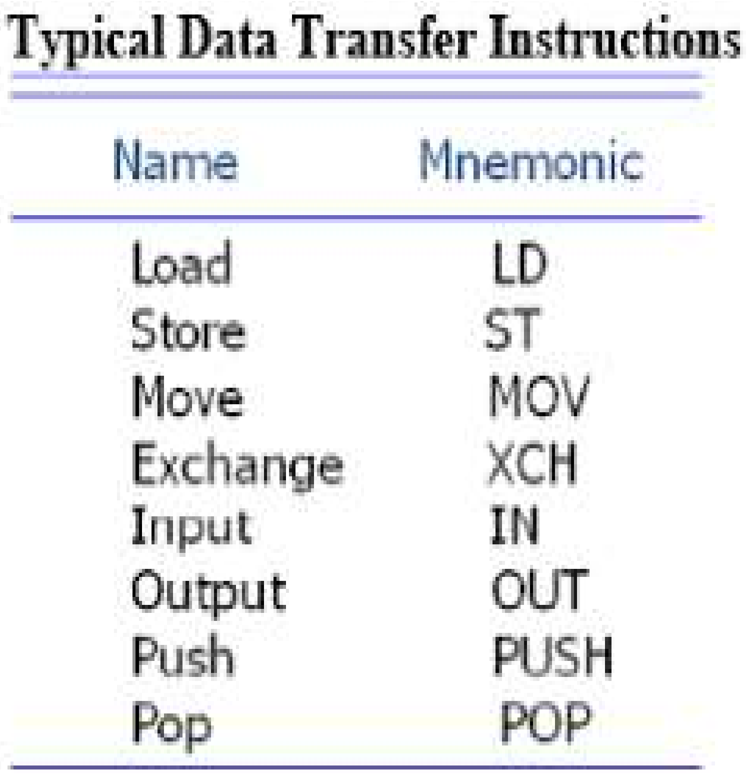
Data Transfer and Manipulation

- Most computer instructions can be classified into three categories:

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

➢ Data Transfer Instructions:

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- Table gives a list of eight data transfer instructions used in many computers.


Typical Data Transfer Instructions

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

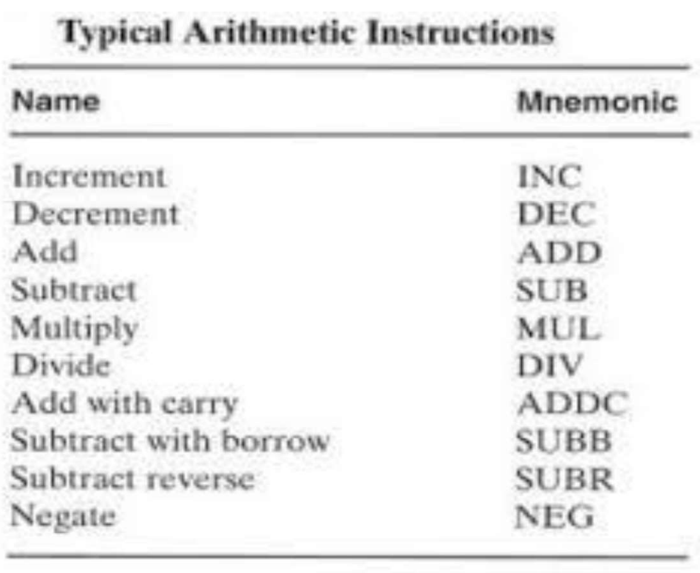
➢ Data Manipulation Instructions:

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Arithmetic instructions

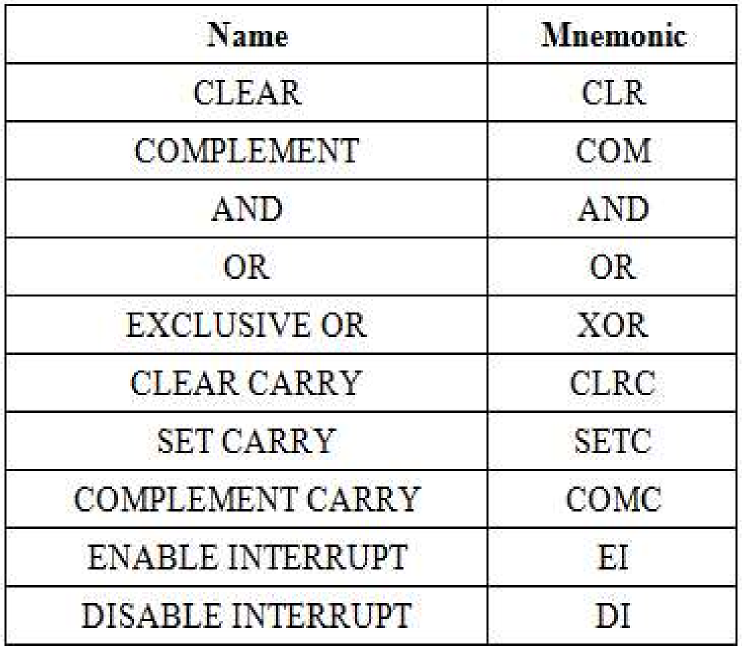
- The four basic arithmetic operations are addition, subtraction, multiplication and division.
- Most computers provide instructions for all four operations.
- Some small computers have only addition and possibly subtraction instructions.
- The multiplication and division must then be generated by mean software subroutines.
- A list of typical arithmetic instructions is given in Table 8-7.



Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Subtract reverse	SUBR
Negate	NEG

- Logical and bit manipulation instructions
- Logical instructions perform binary operations on strings of bits store, registers.
- They are useful for manipulating individual bits or a group of that represent binary-coded information.
- The logical instructions consider each bit of the operand separately and treat it as a Boolean variable.
- By proper application of the logical instructions it is possible to change bit values, to clear a group of bits, or to insert new bit values into operands stored in register memory words.
- Some typical logical and bit manipulation instructions are listed in Table.



Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

- Shift Instruction
- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations.
- In either case the shift may be to the right or to the left.
- Table 8-9 lists four types of shift instructions

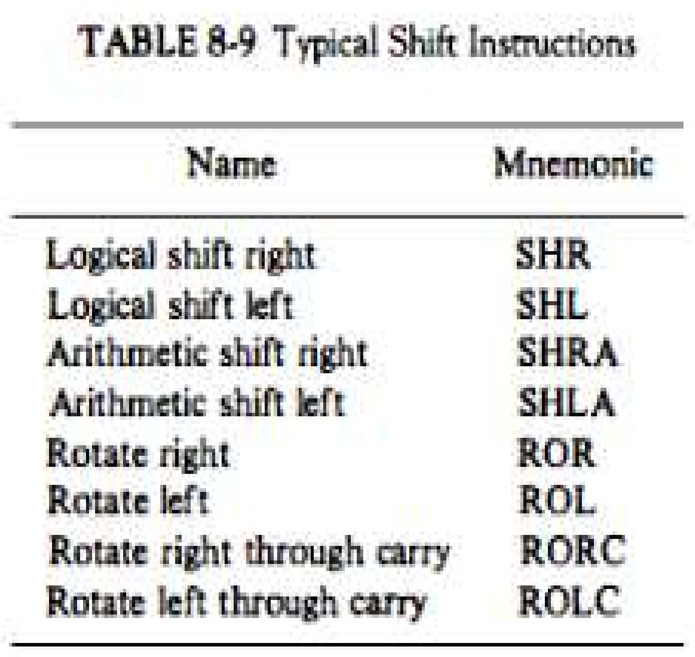
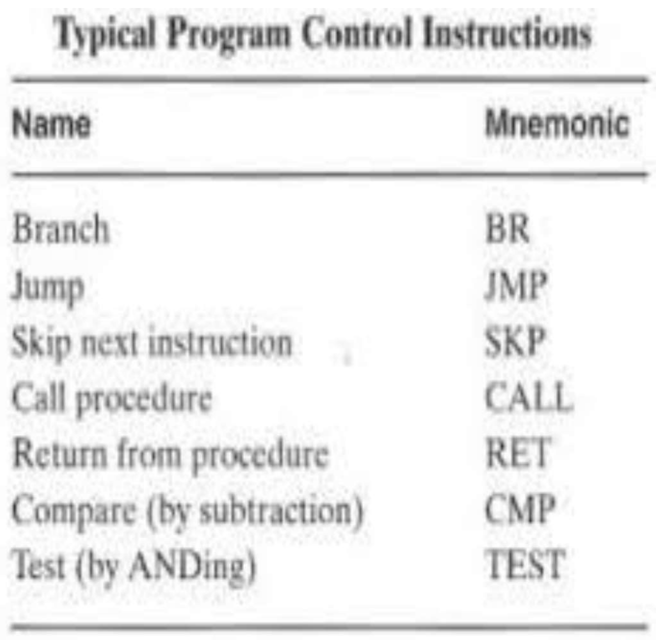


TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program control

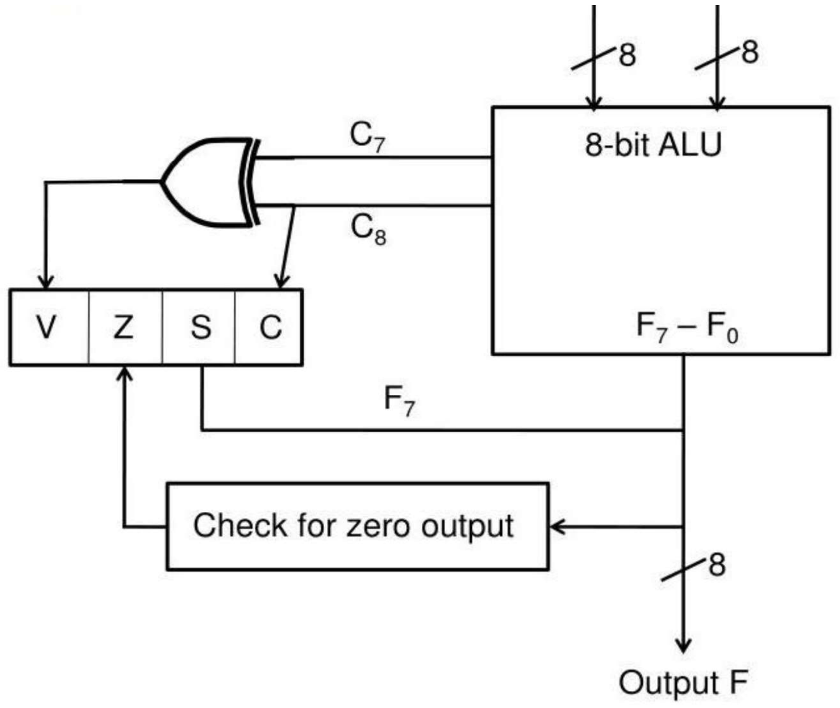
- Program control instructions specify conditions for altering the content of the program counter.
- The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.
- This instruction provides control over the flow of program execution and a capability for branching to different program segments.
- Some typical program control instructions are listed in Table
- Branch and jump instructions may be conditional or unconditional.
- An unconditional branch instruction causes a branch to the specified address without any conditions.
- The conditional branch instruction specifies a condition such as branch if positive or branch if zero.
- The skip instruction does not need an address field and is therefore a zero-address instruction.
- A conditional skip instruction will skip the next instruction if the condition is met. This is accomplished by incrementing program counter.
- The call and return instructions are used in conjunction with subroutines.
- The compare instruction forms a subtraction between two operands, but the result of the operation not retained. However, certain status bit conditions are set as a result of operation.
- Similarly, the test instruction performs the logical AND of two operands and updates certain status bits without retaining the result or changing the operands.



Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip next instruction	SKP
Call procedure	CALL
Return from procedure	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TEST

- Status Bit Conditions:
- The ALU circuit in the CPU have status register for storing the status bit conditions.
- Status bits are also called condition-code bits or flag bits.
- Following Figure shows block diagram of an 8-bit ALU with a 4-bit status register
- The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.
- Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
- S (sign) is set to 1 if the highest-order bit F7 is 1. It is set to 0 if the bit is 0.
- Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is clear to 0 otherwise. In other words, Z = 1 if the output is zero and Z =0 if the output is not zero.
- Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries equal to 1, and cleared to 0 otherwise.
- The above status bits are used in conditional jump and branch instructions.


8
8
8-bit ALU
C_7
C_8
V
Z
S
C
$F_7 - F_0$
F_7
Check for zero output
8
Output F