Unit - II - COA

Microprogrammed control

II-I CSE

Introduction :-

TO execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired control unit and microprogrammed control unit.

Control memory :-

Microprogrammed control unit :-

Control signals :- group of bits used to select paths in multiplexers, decoders, arithmetic logic units.

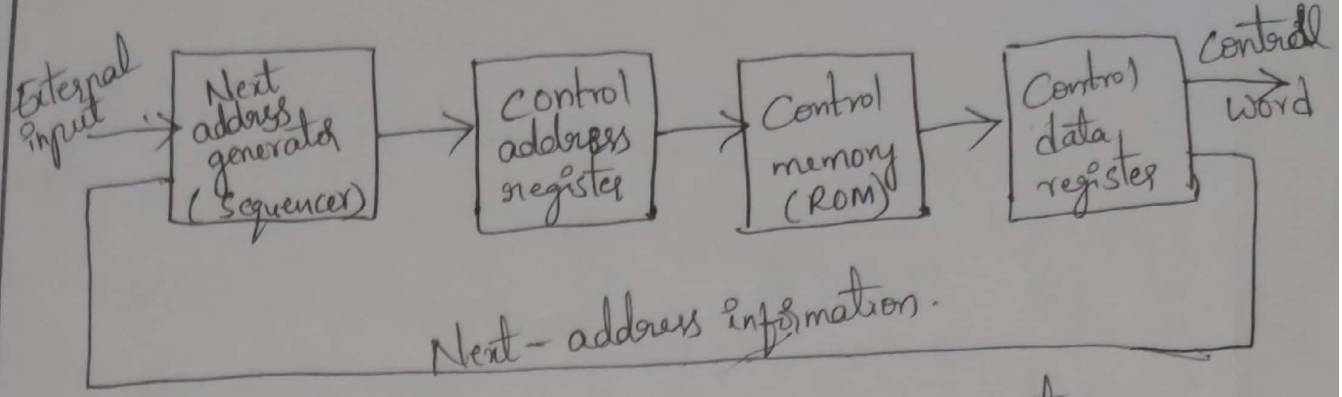
Control variables :- Binary variables that specify microoperations.

Control word :- Strings of 1's and 0's represent control variables.

Control memory :- A memory that is a part of control unit is referred to as a control memory.

Control address register :- The control memory address register specifies the address of the micro instruction.

control data register :- The control data register holds the microinstruction read from memory.



Microprogrammed control Organization.

The above diagram will represents the general configuration of a microprogrammed control unit.

→ The control memory is assumed to be ROM, within which all control information is permanently stored

→ The control memory address register specifies the address of the micro instruction, and the control data register holds the micro instruction read from memory.

→ The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.

→ The control data register holds the present micro-instruction while the next address is computed and read from memory. Sometimes this register called pipeline register.

→ The system can operate without control data register by applying a single phase clock to the address register.

→ The main advantage of the microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware & wiring changes.

* Address Sequencing *

The address sequencing capabilities required in a control memory are

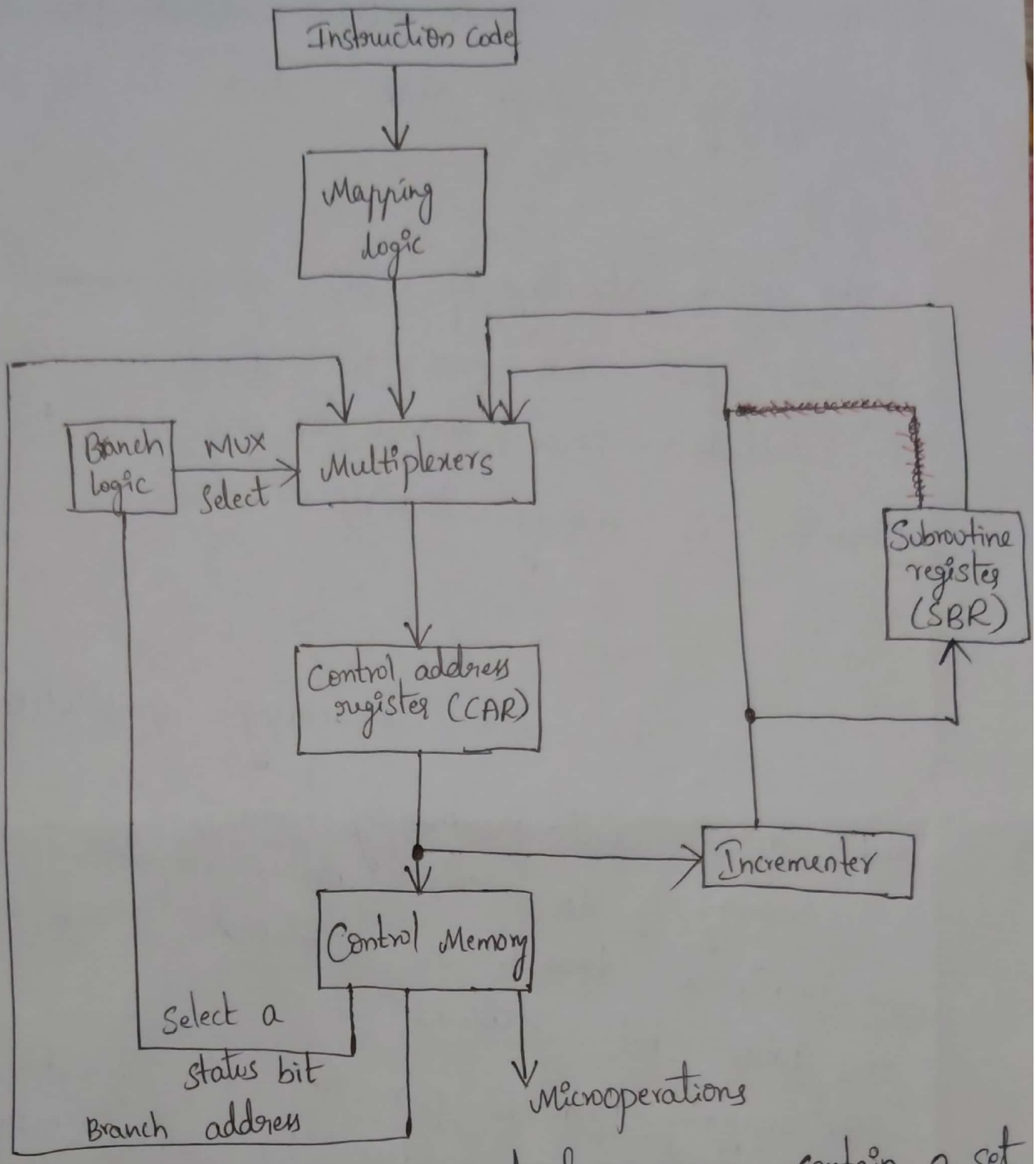
→ Incrementing of the control address register

→ unconditional branch or conditional branch, depending on status bit conditions.

→ A mapping process from the bits of the instruction to an address for control memory.

→ A facility for subroutine call and return.

→ a block diagram shows a control memory and the associated hardware needed for selecting the next microinstruction address.



→ The microinstruction in control memory contain a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.

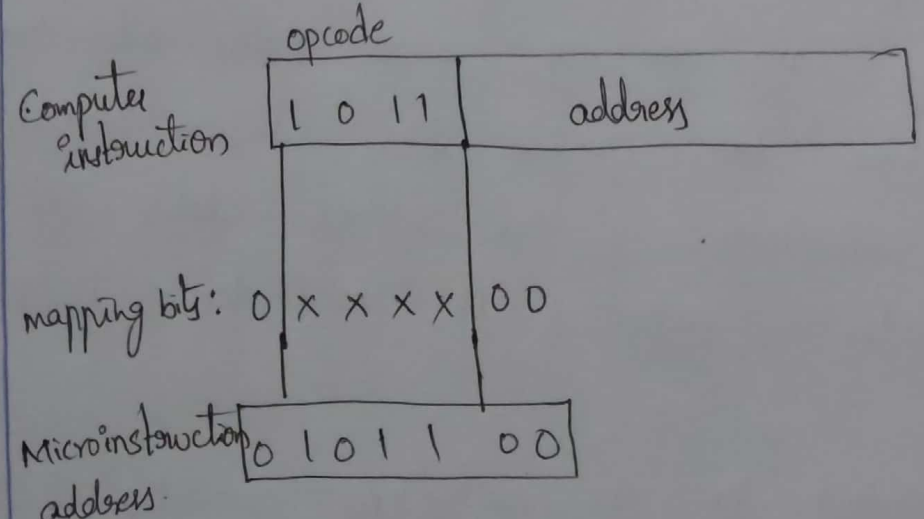
→ The diagram shows four different paths from which the control address register (CAR) receive the address.

→ the incrementer increments the content of the control address register by one, to select the next microinstructions in sequence.

→ Branching is achieved by specifying the branch address in one of the fields of the microinstruction.

→ Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.

→ An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine.



Mapping from instruction code to microinstruction address

Conditional branching :-

The branch logic provides decision-making capabilities in the control unit. The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input & output status conditions.

→ The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise the address register is incremented. This can be implemented with a multiplexer.

→ Suppose that there are eight status bit conditions in the system. three bits in the microinstructions are used to specify any one of eight status bit conditions

→ these three bits provide the selection variables for the multiplexer. If the selected status bit is in the 1 state, the output of the 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.

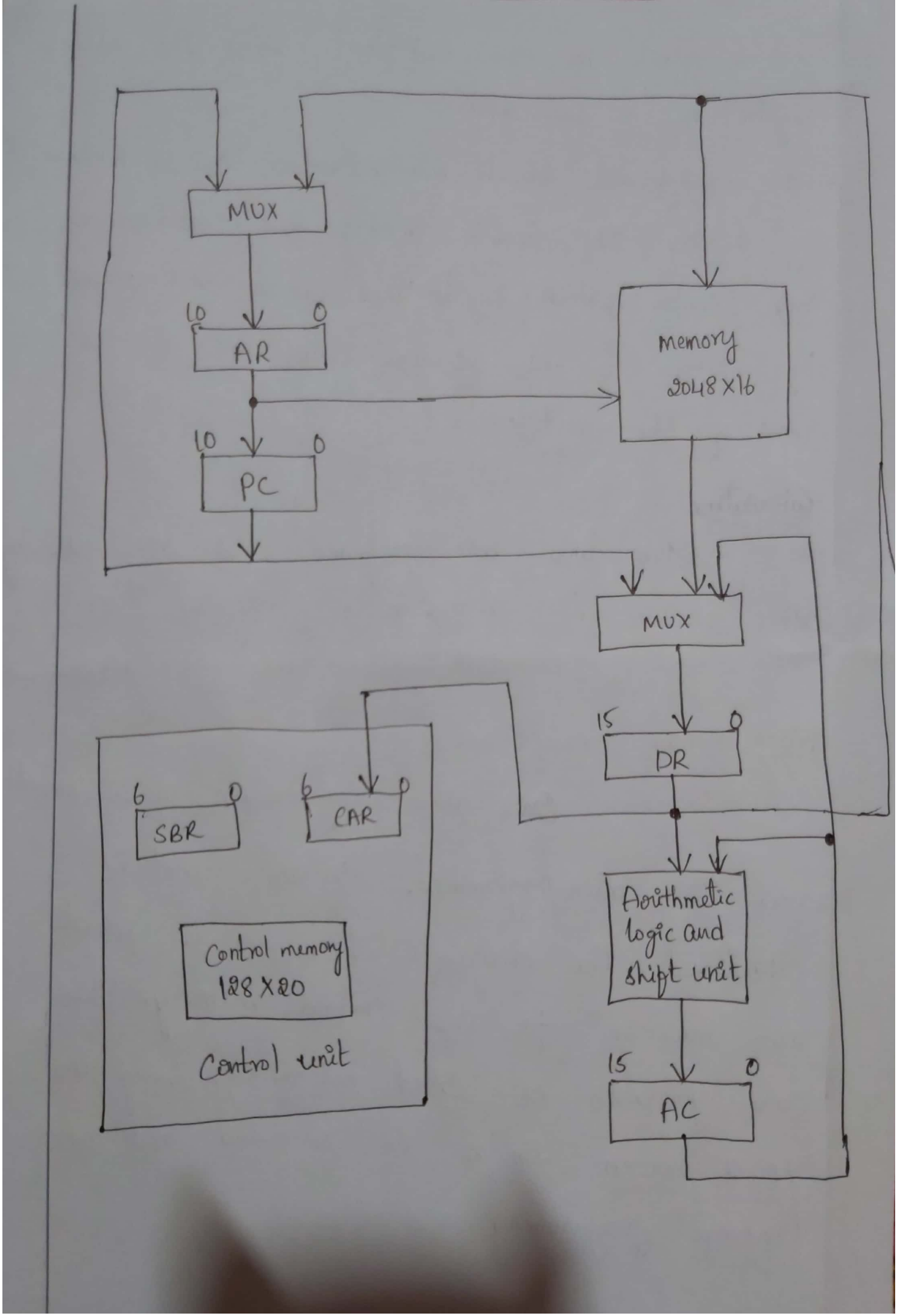
→ An unconditional branch microinstruction can be implemented by loading the branch address from control memory into control address register. This can be accomplished by fixing the value of one status bit at the input of the multiplexer.

Subroutine :-

Subroutines are programs that are used by other routines to accomplish a particular task.

Microprogram example :-

Once the configuration of a computer and its microprogrammed control unit is established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming and is a process similar to conventional machine language programming.



The block diagram shows two memory units: a main memory for storing instruction and data, and a control memory for storing the microprogram.

→ four registers are associated with the processor unit. and two with the control unit. The processor registers are program Counter PC, address register AR, data register DR, and accumulator register AC.

→ The fuction of these register is similar to the basics computer.

→ The control unit has a control address register CAR and a subroutine register SBR. The control memory and its registers are recognised as a microprogrammed control unit.

→ the transfer of information among the registers in the processor is done through multiplexers

→ DR receives information from AC, PC & memory.

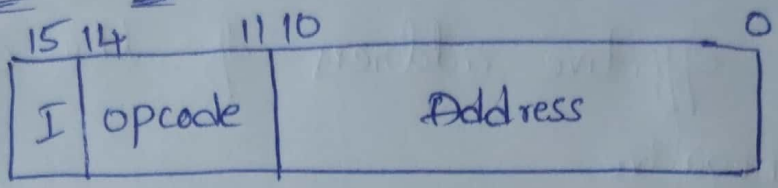
→ AR Can receive information from PC & DR.

→ PC can receive information only from AR.

→ The arithmetic, logic and shift unit performs microoperations with data from AC and DR and places the result

in AC. Note that memory receives its address from AR.
~~Enter~~ Input - data written to memory come from DR,
and data read from memory can go only to DR.

Instruction Format:-



the computer instruction format consist of three fields: a 1-bit field for indirect addressing symbolized by I, a 4-bit operation code opcode, and an 11 bit address field.

→ the opcode of 4 bits can provide 16 possible memory reference instructions

	Symbol	opcode	description
1	ADD	0000	AC ← AC + M(EA)
2	BRANCH	0001	If(AC<0) then (PC ← EA)
3	STORE	0010	M(EA) ← AC
4	EXCHANGE	0011	AC ← M(EA), M(EA) ← AC

→ The ADD instruction adds the content of the operand found in the effective address to the content of AC

→ The BRANCH instruction causes a branch to the effective address of the operand in AC is negative

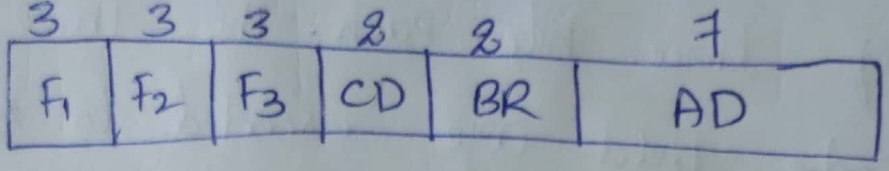
→ The store instruction transfers the content of AC into the memory word specified by the effective address.

→ The EXCHANGE instruction swap the data between AC and the memory word specified by the effective address.

→ In order to not to complicate the microprogramming example, only 4 instructions are considered here.

→ The other 12 instructions can be included and each instruction must be microprogrammed by the procedure.

Microinstruction Format



F_1, F_2, F_3 : Microoperation fields
CD : condition for branching
BR : Branch field
AD : Address field

The above diag format representing the microinstruction format for control memory.

→ The 20 bits of the microinstructions are divided into four functional parts.

→ The three fields F_1, F_2 & F_3 specify microoperations for the computer.

→ The CD field selects the status bit conditions. The BR field specifies the type of the branch to be used. The AD field contains a branch address.

→ The address field is seven bits, since the control memory has $128 = 2^7$ words.

→ The microoperations are subdivided into three fields of three bits each. The three bits in each ~~code~~ field are encoded to specify 7 distinct operations.

→ This gives a total of 21 microoperations.

→ No more than 3 microoperations can be choose -n for a microinstruction, one from each field.

→ If fewer than three microinstructions used one or more of the fields will use the binary code 000 for no operation.

Symbols and Binary code for microinstruction fields

F_1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	AND
010	AC ← 0	CLRAC
011	AC ← AC + 1	INC AC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F_2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F_3	Microoperation	Symbol
000	None	Nop
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow shl\ AC$	SHL
100	$AC \leftarrow shr\ AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Condition field:- The condition field consist of two bits which are encoded to specify four status bit conditions.

→ The first condition always 1, so that a reference to CD = 00, will always find the condition to be true.

This field is used in conjuction with branch field, it provides an unconditional branch operation.

CD	condition	Symbol	comments
00	Always = 1	U	Unconditiond branch
01	DR(15)	I	Indirect addres bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

→ The Indirect bit I is available from bit 15 of DR after an instruction is read from memory.

→ The sign bit of AC provides next status bit.

The Zero value is symbolized by Z. (Zero.), is a binary variable whose value is equal to 1, if all the bits in AC are equal to Zero.

→ We will use the symbols U, I, S and Z for the four status bits when we write micro program in symbolic form.

Branch field :-

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

→ The branch field consist of 2 bits it is used in conjunction with address field AD, to choose the address of the next microinstruction.

→ When BR = 00, the control performs jump operation, when BR = 01 it perform a call to subroutine (CALL) operation.

→ The jump and call operations depend on the value of CD field.

→ If the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to to the control address register CAR, otherwise CAR is incremented by 1.

→ The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR.

→ The mapping from the operation code bits of the instruction to an address for CAR is executed when BR = 11

Symbolic microinstructions :-

→ The symbols can be used to specify microinstructions in symbolic form. A symbolic microprogram can be translated into its binary equivalent by means of an assembler. each line of the assembly language program defines a symbolic microinstruction.

→ each symbolic micro instruction is divided into five fields.

- → label
- → microoperations
- → CD
- → BR and
- → AD.

The field specify individual information.

→ The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:)

→ The microoperation field consist of one, two or three symbols, seperated by commas., from those defined ..

→ The CD field has one of the letters U, I, S or Z

→ The BR field consist one of the four symbols

→ The AD field specifies a value for the address field of the microinstructions. in one of the three possible ways.

a. with a symbolic address, which must also appear as a label.

b. with the symbol next to designate the next address in sequence.

c. when the branch field contains return value (RET) or MAP symbol, the AD field is left empty and is converted to seven zero's by the assembler.

we will use also the pseudoinstruction ORG to define the origin, or first address, of a microprogram routine. Thus the symbol ORG 64 informs the assembler to place the next microinstruction in control memory at decimal address 64, which is equivalent to the binary address 1000000

The Fetch routine:—

A convenient location for the fetch routine is address 64. The microinstructions needed for the fetch routine are

AR ← PC

DR ← M[AR], PC ← PC+1

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

→ The address of the instruction is transferred from PC to AR and the instruction is then read from memory into DR.

→ The address part is transferred to AR and then control is transferred to one of 16 routines by mapping the operation code. Part of the instruction from DR into CAR.

Fetch and decode :-

The fetch routine needs three microinstructions, which are placed control memory at address 64, 65 and 66. Using the assembly language we can write the symbolic microprogram for the fetch routine as follows.

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

The translation of the symbolic microprogram to binary produces the following binary program.

Binary Address	F_1	F_2	F_3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

(~~Partial~~) Symbolic microprogram

The first microinstruction in the ADD routine calls Subroutine Indirect, conditioned on status bit I. If I = 1, a branch to Indirect occurs and the return address is stored in the Subroutine register. SBR. The Indirect subroutine has two instructions.

```
INDIRECT:  READ   U   JMP  NEXT
           DRTAR  U   RET
```

~~Dot~~ * Microprogrammed control unit *

microprogrammed control unit can be classified into two types based on the type of microinstruction stored in the control memory

→ Horizental micro-programmed control unit

→ Vertical micro-programmed control unit

→ In Horizental micro-programmed control unit, the control signals are represented in the decoded binary format (1 bit/CS).

In horizental microprogrammed control unit for n signals require n bit encoding.

→ In vertical microprogrammed control unit, the control signals are represented in the encoded binary format. Here 'n' control signals require $\log_2 n$ bit encoding.

Horizental microprogrammed unit	vertical microprogrammed unit
→ It supports longer control word. wide memory word.	→ width is narrow.. it supports shorter control word.
→ n signals are encoded into 'n' bits	→ n control signals encoded into $\log_2 n$ bits.
→ High degree of parallel operations possible	→ limited ability to express parallelism.
→ No additional hardware required.	→ additional hardware in the form of decoders are required to generate control signals.
→ It is faster	→ It is slower.
→ Control word bits are not fully utilized	→ Control words are fully utilized.

Example

Suppose an instruction set architecture of a general purpose ~~register~~ machine has a total of 126 control signals. The number of bit required in control ~~unit~~ word for horizental and vertical micro-instruction encoding are 127, 7.

→ The horizental encoding, one bit is used for each control signal; therefore, we shall require 126 bits.

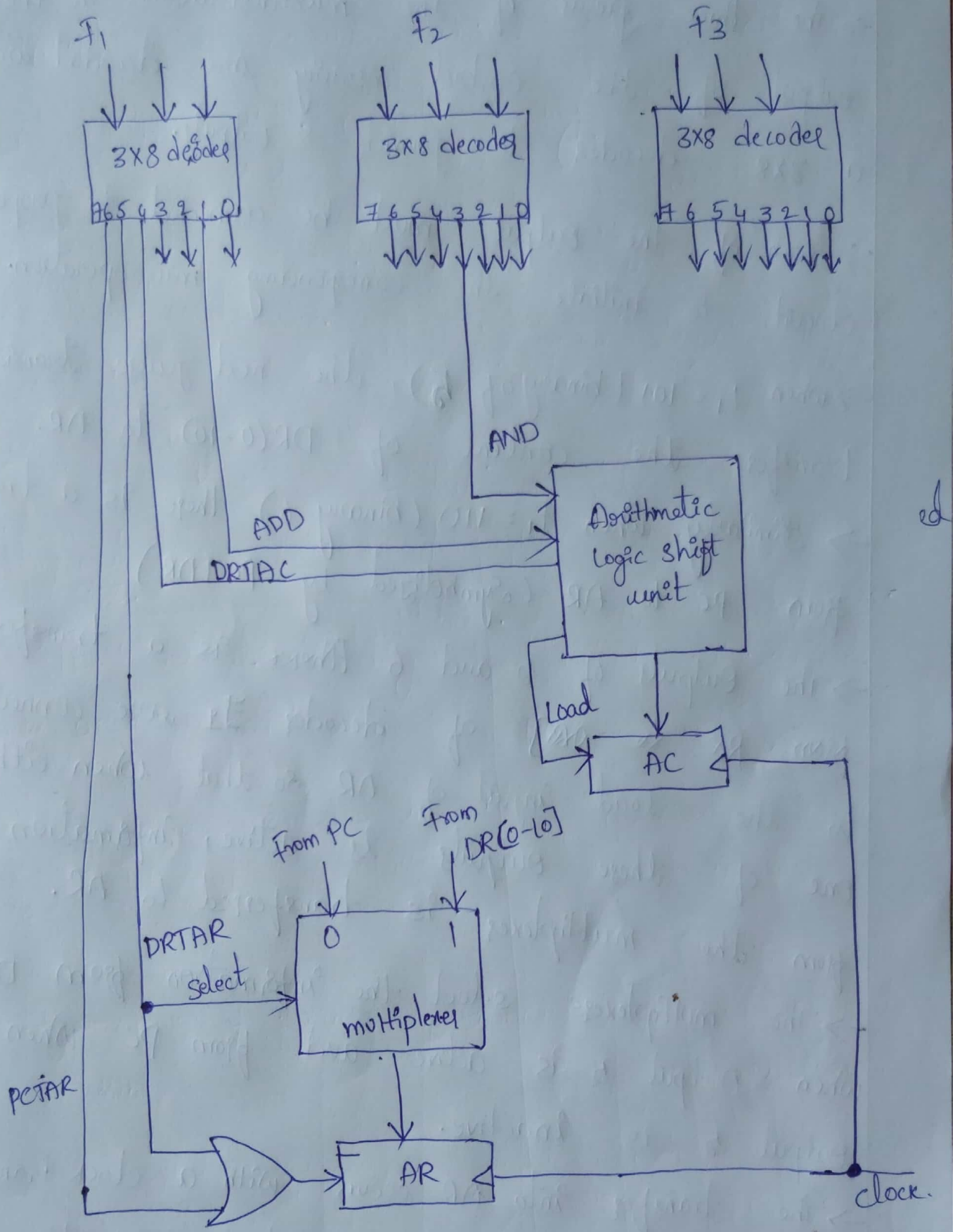
→ For vertical encoding the 126 control signals can be encoded with $(\log_2 126) = 7$ bits.

* 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 diagram shows three decoders and some of the connections that must be made from their outputs.



Decoding of microoperation fields.

→ The three fields of the microinstruction in the output of the control memory are decoded with a 3×8 decoder to provide 8 outputs.

→ Each of the output must be connected to proper circuit to initiate the corresponding microoperation.

→ when $F_1 = 101$ (binary of 5), the next pulse transition transfers the content of DR(0-10) to AR.

→ Similarly when $F_1 = 110$ (binary 6) there is a transfer from PC to AR (symbolized by PC TAR)

→ The outputs of 5 and 6 [there is a transfer from RC to AR] of decoder F_1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexer is transferred to AR.

→ The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive.

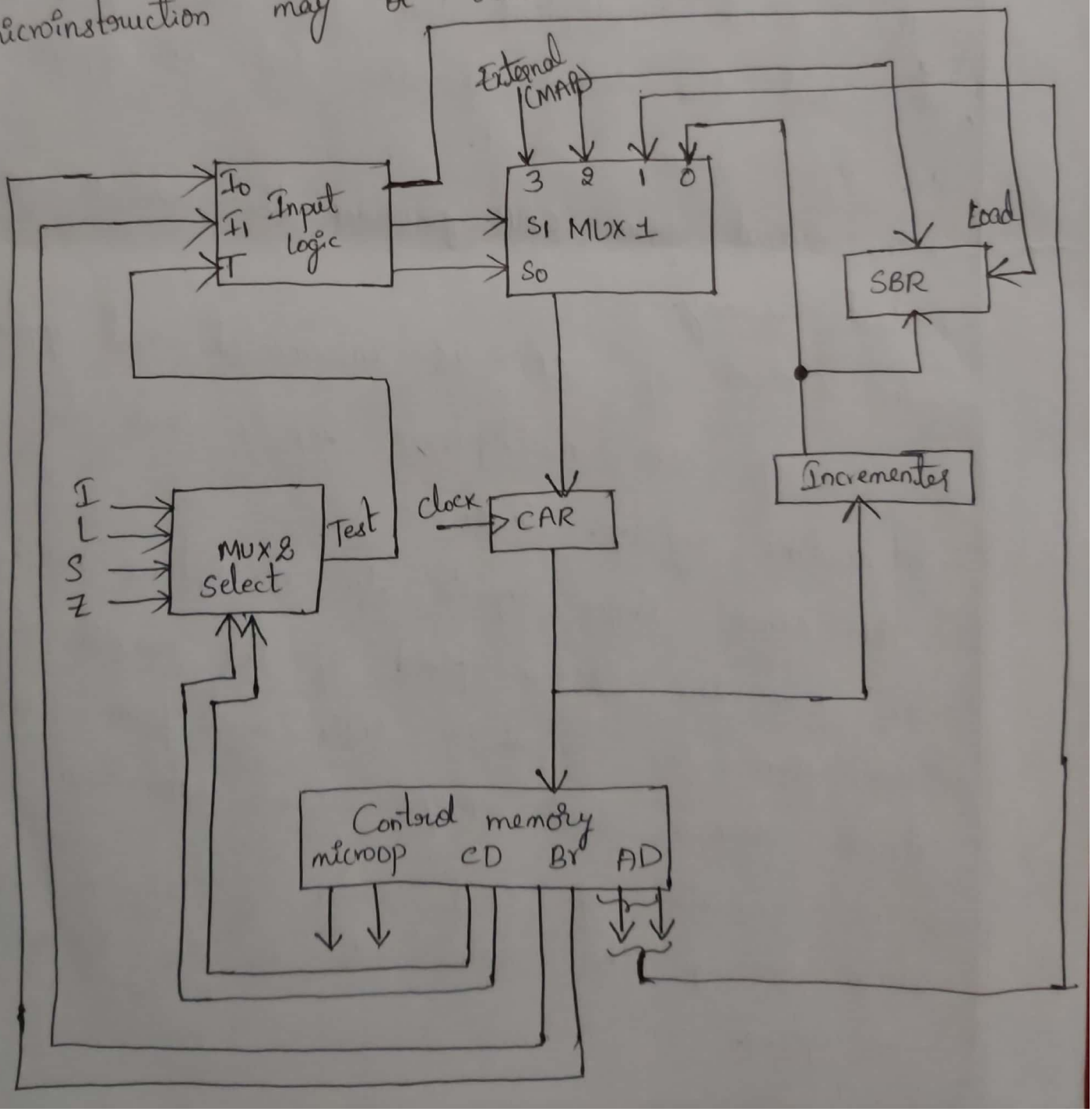
→ The transfer into AR occurs with a clock transition only when output 5 & output 6 of the decoder is active.

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

Microprogram Sequencer :-

The basic components of a microprogram-med control unit are the control memory and the circuits that select the next address. The address selection part is called microprogram sequencer.

→ The purpose of a microprogram sequencer is to present an address to the control memory so that microinstruction may be read and executed.



The block diagram of the microprogram sequencer contain control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.

→ There were two multiplexer in the circuit, the first multiplexer selects an address from one of four sources and routes it into a control address register CAR.

→ The second multiplexer test the value of a selected status bit and the result of the test is applied to an input logic circuit.

→ The output from CAR provides the address for the control memory.

→ The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register. SBR.

→ The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction.

→ The condition (CD) field of the microinstruction selects one of the status bits in the second multiplexer.

→ If the bit selected is equal to 1, the T (test) variable is equal to 1: otherwise, it is equal to 0.

→ The T value together with the two bits from the BR (Branch) field go to an input logic circuit.

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

→ Typical Sequencer operations are: increment, branch & jump, call and return from subroutine, load an external address, push & pop the stack, and other address sequencing operations.

→ With three inputs, the sequencer can provide up to eight address sequencing operations.

BR field		Input I_1	I_0	T	MUX 1 S_1	S_0	Load SBR L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	X	1	0	0
1	1	1	1	X	1	1	0

The input logic circuit has three inputs.
→ I_0, I_1 and T_1, and three outputs S_0, S_1 and L.
→ Variables S_0 and S_1 select one of the source address for CAR. Variables L enables the load input in SBR.
→ The binary values of the two selection binary variables determine the path in the multiplexer.
For example with $S_1S_0 = 10$, multiplexer input numbe-r 2 is selected and eastablishes a transfer path from SBR to CAR.
→ Inputs I_0 and I_1 are identical to the bit values in the BR field. The bit values S_1 and S_0 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 call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).

The truth table can be used to obtained the simplified boolean futions for the input logic circuit.

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$

The circuit can be constructed with three AND gates, an OR gate, and an inverter.

Central processor unit :-

General register Organization :-

Memory locations are needed for storing pointers, counters, return address temporary results etc.

→ Refering to these memory locations is very time consuming because memory access is the most time consuming operation in a computer.

→ Therefore it is convienient to store these intermediate values in processor register.

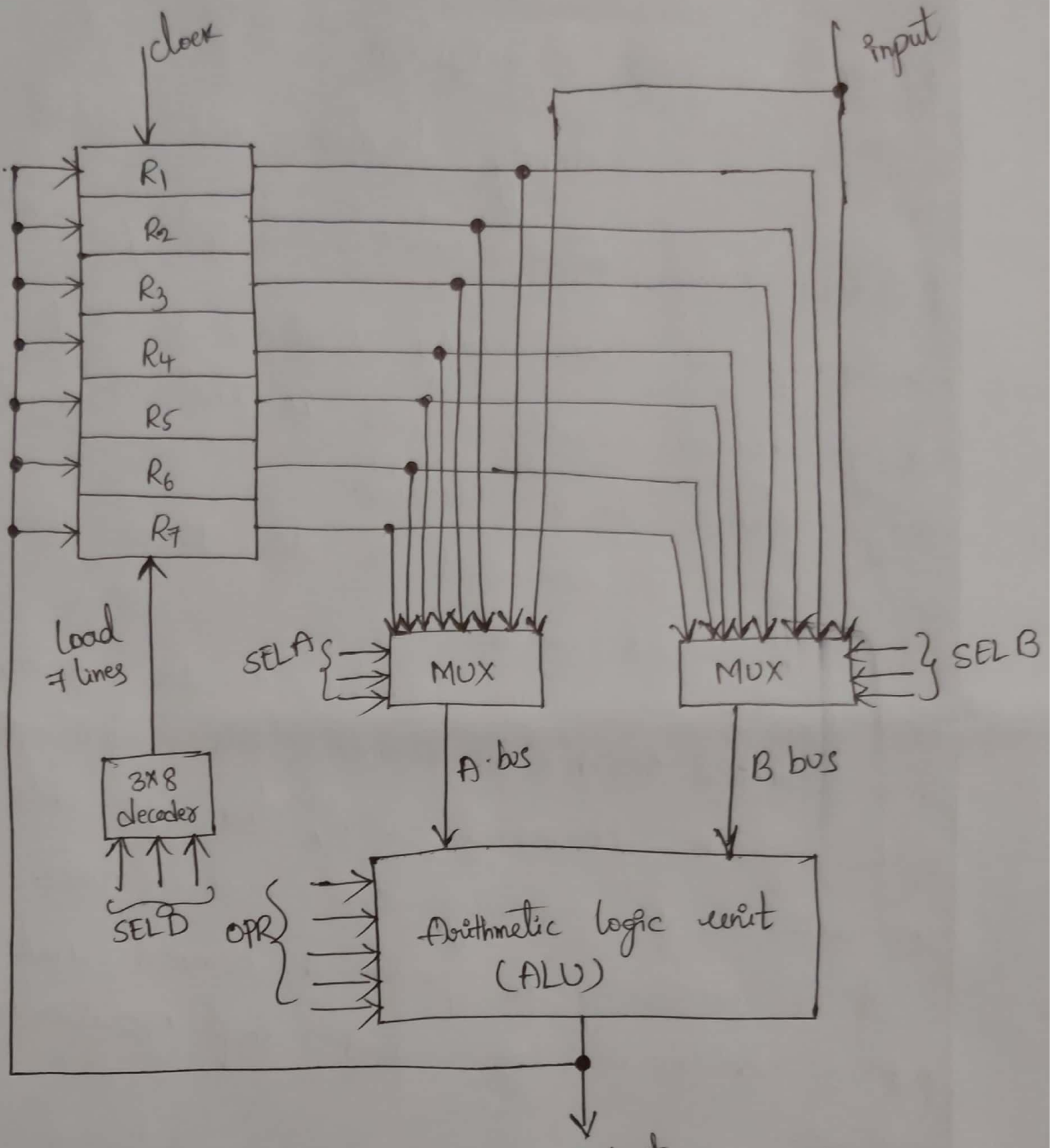
→ When there are many registers in the system they are connected through a common bus system.

→ The registers communicate with each other for data transfer as well as for performing some micro operations.

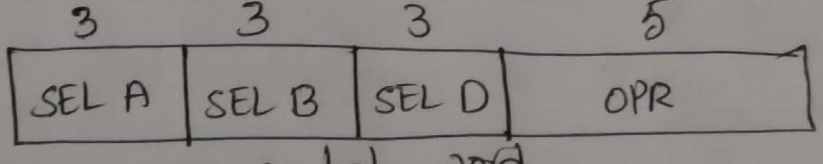
→ It is necessary to provide a common unit that performs arithmetic, logic and shift operations in the processor.

→ A bus organization for 7 CPU registers is shown in figure.

→ The outputs of each register is connected to the two multiplexers (MUX) to form the two buses A and B.



(a) Block diagram of general register organization



(b). Control word

Register set with common ALU

→ The selection lines in each multiplexer select one register & 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 ALU determines the arithmetic or logic microoperation that is to be performed.

→ The result of microoperation 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 (SEL B): to place the contents 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 R_1.

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 contain 4 fields.

→ Three fields contain 3 bits each and last field contain 5 bits.

→ The three bits of SEL A select a source register for the A input of the ALU.

→ The three bits of SEL B select a source register for the B input of the ALU.

→ The three bits of SEL D select a destination register using the decoder and its seven load inputs.

→ 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.

Binary code	SEL A	SEL B	SEL D
000	Input	Input	None.
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

→ The three 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 SEL A, SEL B, and SELD is the one whose decimal number is equivalent to the binary number in the code. when SEL A & SEL B is 000, the corresponding multiplexer selects the external input data.

→ when SEL D = 000, no destination register is selected but the contents of the output bus are available in the external output.

→ ALU provides arithmetic and logic operations.

→ The encoding of the ALU operations for the CPU is shown in the following table.

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

For example, the subtract microoperation given by the statement

$$R_1 \leftarrow R_2 - R_3$$

The binary control word for the subtract microoperation is

010 011 001 00101 and is obtained as follows

Feild	SEL A	SEL B	SEL D	OPR
Symbol	R_2	R_3	R_1	SUB
control word	010	011	001	00101

examples of microoperations of the CPU:

Symbolic designation

Microoperation	SEL A	SEL B	SEL D	OPR	Control word
$R_1 \leftarrow R_2 - R_3$	R_2	R_3	R_1	SUB	010 011 001 00101
$R_4 \leftarrow R_4 \vee R_5$	R_4	R_5	R_4	OR	100 101 100 01010
$R_6 \leftarrow R_6 + 1$	R_6	–	R_6	INCA	110 000 110 00000
$R_7 \leftarrow R_1$	R_1	–	R_7	TSFA	001 000 111 00000
Output $\leftarrow R_2$	R_2	–	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	–	None	TSFA	000 000 000 00000
$R_4 \leftarrow$ shl R_4	R_4	–	R_4	SHLA	100 000 100 11000
$R_5 \leftarrow 0$	R_5	R_5	R_5	XOR	101 101 101 01100

→ 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.

Instruction Formats :-

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 performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way of the operands or the effective address is determined.

→ The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.

→ The mode field of an instruction code specify choose operands from the given address.

→ The number of address fields in the instruction format of a computer depends on the internal organization of its registers. most computers fall into one of three types of CPU organizations.

- Single accumulator organization
- general register organization
- Stack organization.

1. Single accumulator Organization :- the instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic field addition is defined by an assembly language instruction as

ADD X

where X is the address of the operand. the ADD instruction in this case results in the operation

AC ← AC + M[X]

→ AC is the accumulator register and M[X] symbolizes the memory word located at address X.

2. General register Organization :-

The instruction format in this type of computer needs three register address fields. The instruction for an arithmetic addition may be written in an assembly language as to denote

ADD R_1, R_2, R_3

$R_1 \leftarrow R_2 + R_3$

→ The number of address fields in the instruction can be reduced from three to two if the destination register is the same as the source register.

Stack Organization :-

The computers with stack organization would have PUSH and POP instructions which require an address field.

Eg:- PUSH X

The above instruction 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 written as

ADD.

→ depending on no of addresses it can be classified into five categories.

Three - Address Instructions :-

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

→ The Program is assembly language that evaluate

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

ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

→ The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.

→ The disadvantage of is that [it results in short period] binary coded instructions require too many bits to specify three addresses.

Two address instructions :-

Two address instructions are the most common in commercial computers. Each address field can specify either a processor register or a memory word.

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

The MOV instruction moves & transfer the operands to and from memory and process& registers.

One address registers :-

One-address instruction use an implied accumulator (AC) register for all data manipulation.

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$

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 computes does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions. The following program shows how X=(A+B)*(C+D) will be written for a stack-organized computes.

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

To evaluate arithmetic expressions in a stack computes, it is necessary to convert the expression into postfix notation. now the expression can be

AB+CD+*

RISC Instructions :-

The instruction set of a typical RISC processor is restricted to the use of load and store instructions when communicating between memory and CPU. The following is a program to evaluate

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

LOAD	R_1, A	$R_1 \leftarrow M[A]$
LOAD	R_2, B	$R_2 \leftarrow M[B]$
LOAD	R_3, C	$R_3 \leftarrow M[C]$
LOAD	R_4, D	$R_4 \leftarrow M[D]$
ADD	R_1, R_1, R_2	$R_1 \leftarrow R_1 + R_2$
ADD	R_3, R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL	R_1, R_1, R_3	$R_1 \leftarrow R_1 * R_3$
STORE	X, R_1	$M[X] \leftarrow R_1$

The load instructions transfer the operands from memory to CPU registers. The add and multiply operations are executed with data in the registers without accessing memory. The result of the computations is then stored in memory with a store instruction.

Addressing modes :-

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers & memory words.

→ Computers use addressing mode techniques for the purpose of accommodating one & both of the following provisions.

1. To give programming versatility
2. To reduce the number of bits in the addressing field of the instruction.

→ Addressing modes are mainly used to specify the effective address of an operand.

→ Addressing modes can be classified into 10 types.

1. Implied mode
2. Immediate mode
3. Register mode
4. Register Indirect mode
5. Autoincrement mode
6. Autodecrement mode
7. Direct address mode
8. Indirect address mode
9. Relative address mode
10. Indexed addressing mode, Base register mode.

Implied Addressing mode :-

→ The definition of the instruction itself specify the operands implicity.

→ It is also called as implicit addressing mode.

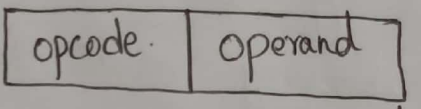
Eg :- CMA.

Immediate addressing mode :-

In this addressing mode,

→ the operand is specified in the instruction explicitly.

→ Instead of address field, an operand field is present that contains the operand.



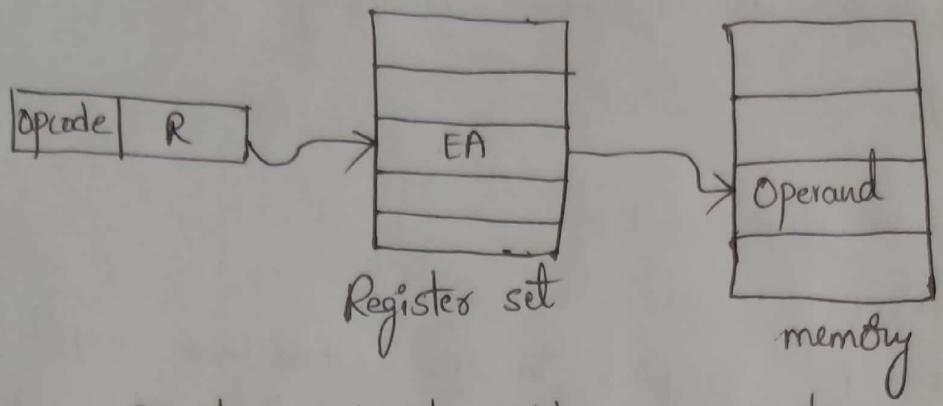
Example :- Add 10 will increment the value stored in the accumulator by 10.

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 addressing mode, the address field of the instruction refers to a CPU register that contain the effective address of the operand.

→ Only one reference to memory is required to fetch the operands.



Register Indirect Addressing mode

Auto increment addressing mode :-

This addressing mode is a special case of register indirect addressing mode.

effective address of the operand = content of register

→ first, the operand value is fetched.

→ Then, the instruction register. R_{AUTO} value is incremented by 'step size'd'

Auto decrement addressing mode :-

→ This addressing mode is again a special case of register indirect addressing mode where-

Effective address of the operand = content of register − step size.

In this addressing mode,

→ First, the content of the register is decremented by step size 'd'.

→ step size 'd' depends on the size of operand accessed.

→ after decrementing, the operand is read.

→ Only one reference to memory is required to fetch the operand.

Direct addressing mode :-

In this addressing mode,

→ The addressing field of the instruction contains the effective address of the operand.

→ Only one reference to memory is required to fetch the operand.

→ It is also called as absolute addressing mode.

Indirect addressing mode :-

In this addressing mode,

→ The address field of the instruction specifies the address of memory location that contains the effective address of the operand.

→ two references to memory are required to fetch the operand.

Relative address mode :-

In this addressing mode,

→ effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

> effective address = Content of program counter + Address part of the instruction

Indexed addressing mode & Base register addressing mode

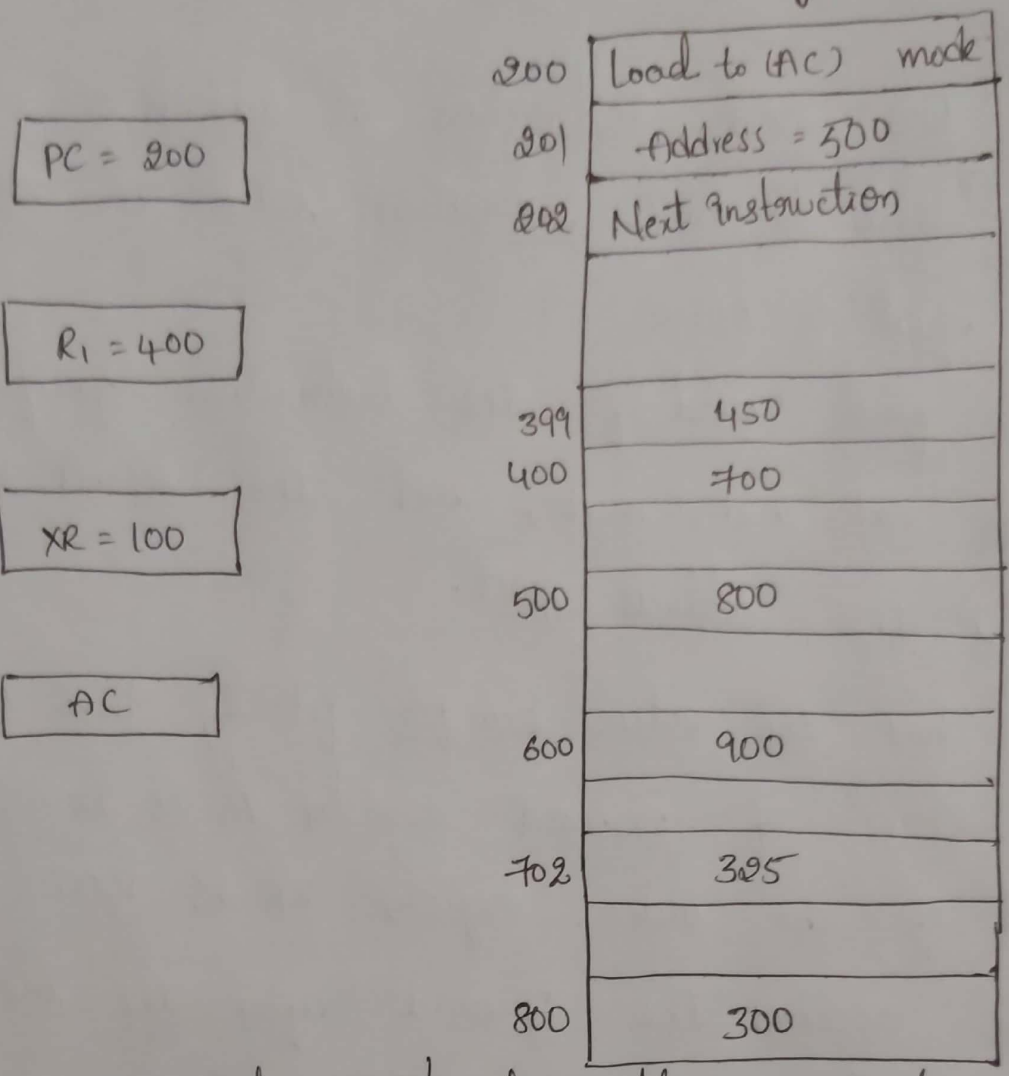
In Indexed addressing mode, effective address of the operand is obtained by adding the content of Index register with the address part of the instruction

> effective address = Content of Index register + Address part of the instruction

→ In base addressing mode, effective address of the operand is obtained by adding the content of base register with the address part of the instruction.

> effective address = Content of base register + Address part of the instruction

Numerical example



Numerical example for addressing mode.

Addressing mode	effective address	content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	305
Indexed address	600	900
Register	—	400
Register Indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

To show the differences between the various modes, the effect of the address modes on the instruction defined.

→ 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 R_1 is 400, and the content of an index register XR is 100.

→ AC receives the operand after the instruction is executed.

→ The mode field of the instruction can specify any one of a number of modes. For each possible mode we calculate the effective address and the operand that must be loaded into AC.

→ 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.

→ In the relative mode the effective address is 500+202 = 702 and the Operand is 325. (i.e PC=202)

→ In the index mode the effective address is XR+500 = 100+500 = 600. and the operand is 900.

→ In the registermode the operand is in R_1 and 400 is loaded into AC (there is no effective address in this case).

→ In the register indirect mode the effective address is equal to 400, equal to the content of R_1 and the operand loaded into AC. is 700.

→ The autoincrement mode is the same as the register indirect mode except that R_1 is incremented to 401 after the execution of the instruction.

→ The auto decrement mode decrements R_1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.

Data transfer and manipulation :-

Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks. The instruction set of different computers differ from each other. mostly in the way the operands are determined from the address and mode field.

→ most computer instructions can be classified into three categories.

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

1. Data transfer instructions :-

→ Data transfer instructions move data from one location to another without changing the binary information content.

→ the most common transfers are between memory and processor registers, between processor registers and input & output, and between the processor registers themselves.

→ eight data transfer instructions used in many computers each instruction is a mnemonic symbol. it must be realized that different computers use different mnemonics for the same instruction name.

Typical Data transfer instructions

Name	Mnemonic
Load	LD
store	ST
move	MOV
exchange	XCH
Input	IN
Output	OUT
Push	PUSH
POP	POP

→ The Load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.

→ The Store instruction designates a transfer from a processor register into memory

→ The move instruction has been used in computers with multiple CPU registers. It has been used data transfers between CPU registers and memory or between two memory words.

→ The exchange instruction swaps information between two registers or a register and a memory word.

→ The input and output instructions transfer data among processor registers and input or output terminals

→ The push and pop instructions transfer data between processor registers and a memory stack.

→ Instructions can be associated with a variety of addressing modes. As an example, Consider the load to accumulator instruction when used with eight different addressing modes.

Eight Addressing modes for the load instructions.

Mode	Assembly Convention	Register transfer
Direct address	LDADR	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	LDADR(X)	AC ← M[ADR+XR]
Register	LD R1	AC ← R_1
Register Indirect	LD(R1)	AC ← M[R1]
Auto increment	LD(R1)+	AC ← M[R1], R_1 ← R_1+1

→ ADR stands for raw address, NBR is a number or Operand, X is an index register, R_1 is a processor register, and AC is the accumulator register.

→ The @ character symbolizes an indirect address.

→ The $ character before an address makes the address relative to the program counter PC.

→ The # character precedes the operand in an immediate mode instruction.

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.

- → Arithmetic instructions
- → Logical and bit manipulation instructions
- → Shift instructions.

Arithmetic instructions :-

The four basic arithmetic operations are addition, subtraction, multiplication and division.

→ most computers provide instructions for all four operations.

typical arithmetic instructions

Name	Mnemonic form
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
multiply	MUL
Divide	DIV
Add with carry	ADDC
subtract with borrow	SUBB
Negate (2's complement)	NEG

→ The increment instructions adds 1 to the value stored in a register or memory word.

→ The decrement instruction subtracts 1 from a value stored in a register or memory word.

→ The add, subtract, multiply and divide instructions may be available for different types of data.

→ The mnemonics for three add instructions that specify different data types are shown below.

ADDI	Add two binary integer numbers
ADDF	Add two floating-point numbers
ADDD	Add two decimal numbers in BCD.

Logical and Bit manipulation instructions :-

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary coded information.

Typical logical and Bit manipulation instructions

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

→ The clear instruction causes the specified operand to be replaced by 0's

→ The complement instruction produces the 1's complement by inverting all the bits of the operand.

→ The AND, OR, and XOR instructions produce the corresponding logical operations on individual bits of the operands.

→ The XOR instruction is used to selectively complement bits of an operands.

→ Individual bits such as a carry can be cleared, set & complemented with appropriate instructions.

→ flip-flop that controls the interrupt facility and is either enabled & disabled by means of bit manipulation instructions

Shift instructions :-

These instructions used to shift the content of an operand. shifts are operations in which the bits of a word moved to the left & right.

→ shift instructions may specify either logical shifts, arithmetic shifts, & rotate-type operations. In either case the shift may be to the right or to the left.

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

→ The logical shift inserts 0 to the end bit position. The end position is the leftmost bit for shift right and the rightmost bit position for the shift left.

→ Arithmetic shifts usually conform with the rules for signed-2's complement numbers.

→ The arithmetic shift-right instruction must preserve the sign bit in the leftmost position. The sign bit is shifted to the right together with the rest of the number, but the sign bit itself remains unchanged. This is a shift right-operation with the end bit remaining in the same.

→ The arithmetic shift-left instruction. Inserts 0 to the end position and is identical to the logical shift-left instruction.

→ The rotate instructions produce a circular shift. Bits shifted out at one end of the word are not lost as in a logical shift but are circulated back into the other end.

→ The rotate through Carry instruction treats a carry bit as an extension of the register whose word is being rotated.

→ A possible instruction code format of a shift instruction may include five fields as follows.

OP REG TYPE RL COUNT

→ Here OP is the operation code field. 'REG' is a register address that specifies the location of the operand.

→ Type is a 2-bit field specifying the four different type of shifts.

→ RL is a 1-bit field specifying a shift right & left.

→ COUNT is a K-bit field specifying up to $2^K - 1$ shifts

Program Control Instructions :-

Program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data-processing operations.

Typical program control instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

→ The branch and jump instructions are used interchangeably to mean the samething, but sometimes they are used to denote different addressing modes.

→ Branch and jump instructions may be conditional & unconditional.

→ an unconditional branch instructions causes a branch to the specified address without any conditions.

→ The conditiond branch instructions specifies a condition such as branch if positive & branch if zero.

→ If the condition met, the program counter is loaded with the branch address and the next instruction is taken from the address. If the condition not met, the program counter is not changed and the next instruction is taken from the next location in sequence.

→ The skip instructions 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.

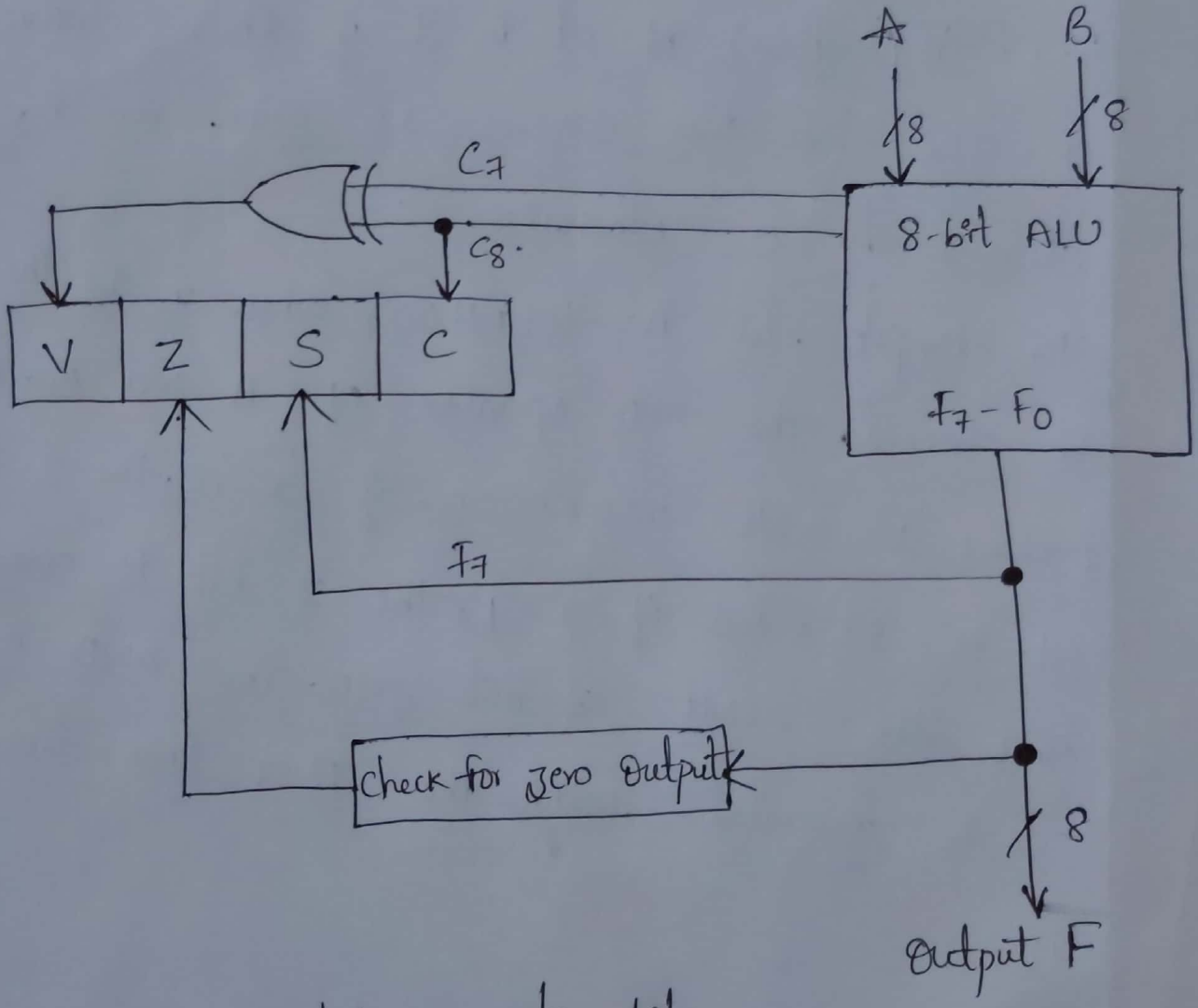
→ The call and return instructions are used in conjunction with subroutines.

→ The compare and test instructions do not change the program sequence directly. The compare instruction performs a subtraction between two operands, but the result of the operation is not retained.

→ Test instruction performs the logical AND of two operands and updates certain status bit of interest are the carry bit, the sign bit, a zero indication, and an overflow condition.

Status bit conditions :-

Status bits are also called condition-code bits or flag bits. The 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.



Status register bits.

1. Bit C (Carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.

2. Bit S (Sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if the bit is zero.

3. Bit Z (Zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise.

4. Bit V (Overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise.

For example, let A = 101x1100, where x is the bit to be checked. The AND operation of A and B = 00010000 produces a result 000x0000. If x = 0, the Z status bit is set, but if x = 1, the Z bit is cleared since the result is not zero. The AND operation can be generated with the TEST instruction.

Conditional branch statements

conditional Branch statements

Mnemonic	Branch condition	Tested condition.
BZ	Branch if zero	$Z=1$
BNZ	Branch if not zero	$Z=0$
BC	Branch if carry	$C=1$
BNC	Branch if no carry	$C=0$
BP	Branch if plus	$S=0$
BM	Branch if minus	$S=1$
BV	Branch if overflow	$V=1$
BNV	Branch if no overflow	$V=0$

unsigned compare conditions (A-B)

BHI	Branch if higher	$A>B$
BHE	Branch if higher & equal	$A \geq B$
BLO	Branch if lower	$A<B$
BLOE	Branch if lower & equal	$A \leq B$
BE	Branch if equal	$A=B$
BNE	Branch if not equal	$A \neq B$

Signed compare conditions (A-B)

Mnemonic	Branch condition	Tested condition
BGT	Branch if greater than	$A > B$
BGE	Branch if greater & equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less & equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

If the stated condition is true, program control is transferred to the address specified by the instruction, if not, control continues with the instruction that follows.

Subroutine call and return

A subroutine is a self-contained sequence of instructions that performs a given computational task. The instruction is executed by performing two operations.

1. The address of the next instruction available in the program counter is stored in a temporary locations so the subroutine knows where to return.

(2) Control is transferred to the beginning of the subroutine.

$SP \leftarrow SP-1$	Decrement stack pointer
$M[SP] \leftarrow PC$	push content of PC onto the stack
$PC \leftarrow$ effective address	Transfer control to the subroutine.

The new return address is pushed into the stack, and so on. The instruction that returns from the last subroutine is implemented by the microoperations

$PC \leftarrow M[SP]$	Pop stack and transfer to PC
$SP \leftarrow SP+1$	Increment stack pointer.

Program Interrupt :-

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external & internal generated request. Control returns to the original program after the service program is executed.

The interrupt procedure quite similar to a subroutine call except for three variations.

1. The interrupt is usually initiated by an internal & external signal rather than from the execution of an instruction

2. The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction

3. an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only program counter.

Types of interrupt :-

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal Interrupts
3. Software interrupts

External interrupts come from input-output devices, from a timing device, from a circuit monitoring the power supply, & from any other external source.

→ Internal interrupts arise from illegal & erroneous use of an instruction & data. Internal ~~interrupt~~ interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow.

→ A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.

* UNIT-II completed *