UNIT - I

Digital Computer

Introduction, Block diagram of Digital computer, Definition of computer organisation, Computer Design and computer Architecture

Basic computer Organisations and Design.

Instruction codes, computer Registers, computer Instructions, Timing and control, Instruction cycle, Memory Reference Instructions, Input - o/p and Interrupt, Complete computer Description, h/wired contro unit, Micro progeam control unit.

Computer:- Computer is an electronic machine which take data through i/p device then process and generate its result through o/p device.

full form:- Commonly operating machine particularly used for Technical, Education & Research.

functionalities of computer

1) Computer takes data as i/p.
2) It stores the data/instructions in its memory and used them when required.
3) It process the data and convert it [illegible] useful information.

4) It generates the o/p of the given i/p by the users.

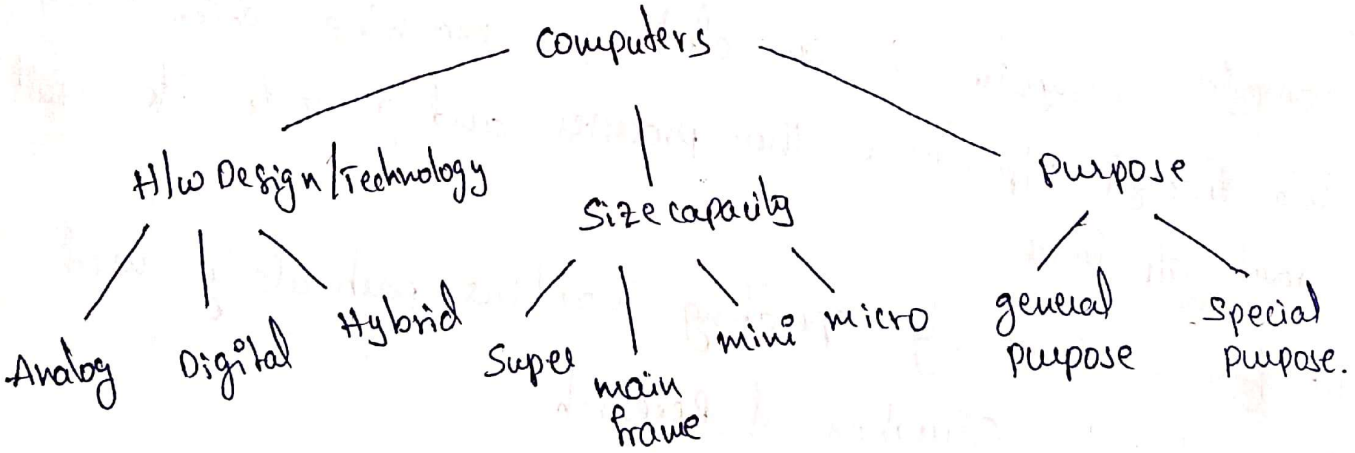
Advantages:-

1) High speed
2) Accuracy.
3) Storage capacity
4) versatality
5) Reliability
6) Automation
7) Reduction in Paper work
8) Reduction in cost.

Disadvantages

1) Dependency
2) Environment
3) No feelings.

classification of Computers



Technology: It is the collection of

1) Techniques
2) Skills
3) Methods
4) Process

used in the production of goods or services or products.

classification of computer based on Technology

The computers can be classified based on the technology used in that as:

1) Digital computer
2) Analog computer
3) Hybrid computer

[Digital computer

Data that can be take certain values called discrete data otherwise continuous data.]

Differences blw Analog, Digital, Hybrid computers

Analog computers	Digital computer	Hybrid computers.
1) Computer which can work on continuous signals Ex: Therma meter, speed meter on bike, watches, seismograph to measure the shaking caused by earthquake (intensity of earthquake). 2) performs particular task, no multi task support	computers which can work on discrete signals (digital signals) Ex: smart phone, personal computer 2) Digital computers are those computers that works on the value of digits i.e : 0 - off 1 - ON	which can work on analog signals & digital signals simultenously Ex: heart measurement machine, 2) Hybrid computers are those computers that has a combination of both analog and digital systems and works on both continuous and descrete data

3) Work on continuous data and gives continuous o/p 4) Work on real time and has no storage capacity. 5) o/p are in the form of graphs, signals, movements etc.	3) Most of the electronic systems are digital computers. 4) These computers are very popular for actual computers works like report preparation, documentation, billing etc. as they can be used for multitasking 5) Faster than Analog computers and has storage capacity 6) Highly accurate & reliable.	3) These computers transfer that the data from analog to digital and vice versa. 4) Mostly used in the systems of ICU of hospitals, Jet plans and other data analysis terminals 5) It has limited storage. 6) It is expensive and complex system

Digital computer

Data that can take certain values called descrete data otherwise continuous data.

→ These computers are capable of procesing data in descrete form.

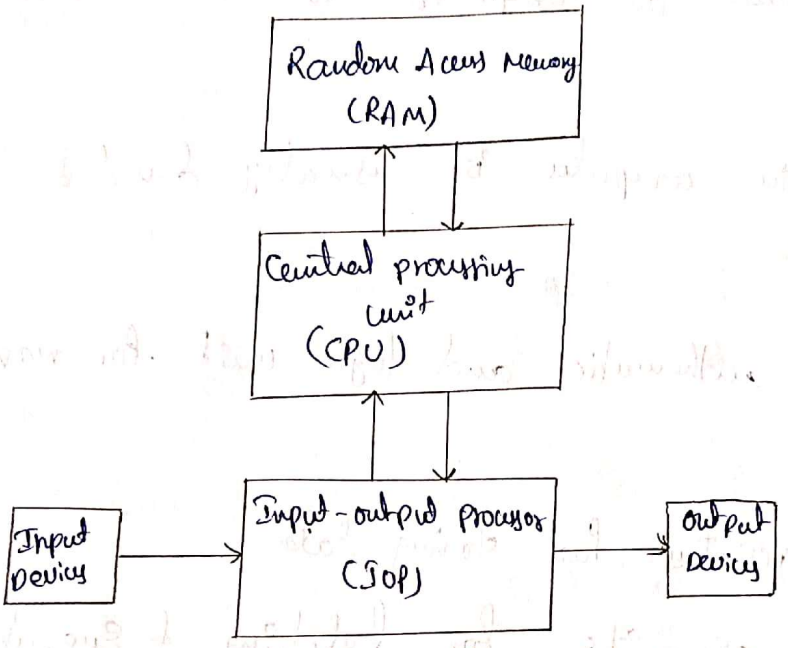
→ In digital technology data which can be in the form of letters symbols or numbers is represented in binary form i.e 0's & 1's

→ By manipulating (count, add, compare) binary digital computer do data processing

→ The digital computers are used in industrial, business and scientific application.

→ They are quite suitable for large volume data processing.

Block Diagram of Digital computer



Block Diagram of Digital computer

The digital computer is a digital system that performs various computational tasks.

→ Digital computers use the binary digit called bit. Information is represented in digital computers in groups of bits.

→ A computer system is sometimes subdivided into two functional entities h/w & s/w.

→ The h/w of the computer consist of all the electronic components and electronic devices that comprises the physical entity

Computer s/w consist of the instruction and data that the computer manipulates to perform various data processing tasks. A sequence of instructions for the computer is called a program. The data that are manipulated by the program constitute the data base.

A computer system is composed of its h/w & s/w available of its use.

The h/w of the computer is usually divided into three major parts.

1) CPU contain arithematic and logic unit for manipulating data.

2) → A no of registers for storing data.

3) → control unit circuits for fetching & executing data/ instructions.

2) The memory of a computer contains storage for instructions and data. It is called a random access memory (RAM) because the CPU can access any location in memory at ~~access memory~~ (~~RAM~~) ~~because~~ ~~the~~ random and retrieve the binary information within a fixed interval of time.

3) The i/p and o/p processor (IOP) containing electronic

information blw the computer and the outside world. The i/p and o/p devices connected to the computer include keyboards, Printers, terminals, magnetic disks devices, and other communication devices.

Definition of computer organisation, computer Design and computer Architecture

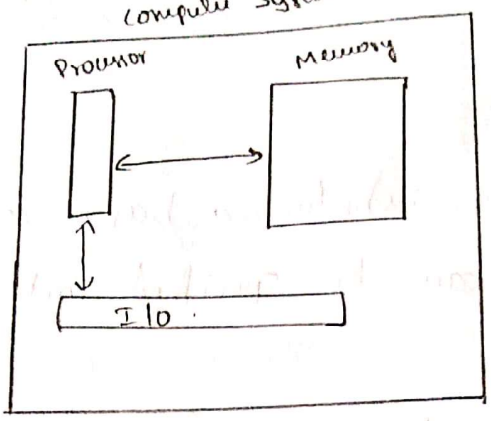
computer organisation :- C.O is concerned with the way the h/w components operate and the way they are connected together to form the computer System. The various components are assumed to be in place and the task is to investigate the organisational structure to verify that the computer parts operate as intended.

Computer Design :- Computer design is concerned with the h/w design of the computer. Once the computer specifications are formulated, It is the task of the designer to develop h/w for the System. Computer Design is concerned with the determination of what h/w should be used and how the parts should be connected. This aspect of computer h/w is sometimes referred to as computer Implementation

Computer Architecture :- Computer Architecture is concerned with the structure and behaviour of the computer as seen by the user. It includes the information formation; instruction set and techniques for addressing memory. The architecture design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories and structuring them together into a computer System.

Computer Architecture	Computer Organisation
1) what computer does	1) How the computer do it.
2) Deals with high level design issues (System design / computer Design)	2) Deals with low level design issues (logic / circuit)
3) Functional, behaviour Ex: (Doctor).	3) structural behavioural. (jim) Ex:
4) Designing of computer "first"	4) Designing of computer "second".
5) Instruction sets, Addressing modes Data type, cache optimisation Physics (logic)	5) Circuit design signals (physical components) ALU, CPU and memory.
6) Transparent from progammer (Ex: A progammer does not worry much how addition is implemented)	6) progammer view (Progammer has to be aware of which instruction

Computer System



Processor: Instructions are executed in processor

It is two parts

1. ALU
2. CU

Memory: used store the data.

I/O: used to transfer the data from user to computer and vice versa.

chapter V (123-157)

Basic Computer Organisation and Design

Instruction codes:-

In this chapter we introduce a basic computer and show how its operation can be specified with register transfer statements.

The internal organisation of a digital computer system is defined by the sequence of microoperations it performs on data stored in its registers. The general purpose digital computer is capable of executing various microoperations and in addition can be instructed as to what specific sequence of operations it must perform.

→ The user of a computer can control the process by means of a program.

→ A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur. The data processing task may be altered by specifying a new program with different instructions or specifying same instructions with different data.

A computer Instruction is a binary code that specifies a sequence of microoperations for the computer.

→ Instruction codes together with data are stored in

→ The computer reads each instruction from memory and places it in a control register.

→ The control then interprets the binary code of instruction and proceeds to execute it by issuing a sequence of micro operations.

→ Every computer has its own unique instruction set.

→ The ability to store and execute instructions, the stored program concept, is the most important property of general purpose computer.

→ An Instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation

Ex: ADD US?

→ The most basic part of an instruction code is its operation part.

→ The operation code (opcode) of an instruction is a group of bits that define such operations as add, subtract, multiply, shift and complement.

→ The no. of bits required for the operation code of an instruction depends on the total no. of operations available in the computer.

→ The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Ex: consider a computer with 64 distinct operations, and one of the operation is ADD. So opcode must be 6 bits because $2^6 = 64$ operation.

for ADD operation opcode is 110010 (let assume).

when this opcode is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.

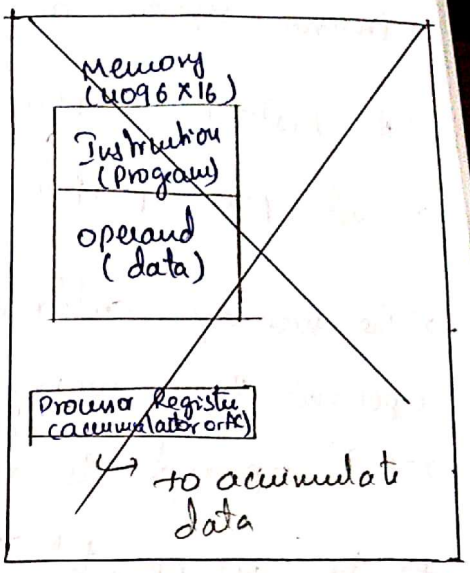
There is difference b/w computer operation and microoperation.

→ An operation is a part of an instruction stored in computer ~~program~~ memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interpret the operation code bits. It then issues a sequence of control signals to initiate microoperations in internal computer registers.

→ for every operation code, the control issues a sequence of microoperations needed for the hardware implementation of

~~Stro~~

Stored progeam Organisation



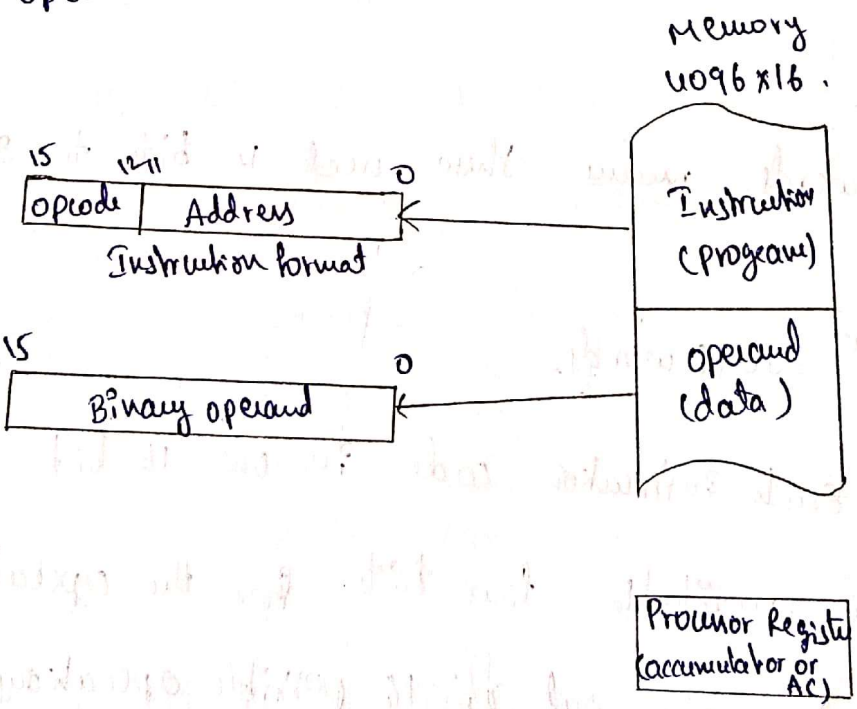
for general pupose dp compute here we are considering 4096 x 16 memory

Here 4096 words of 16 bit each.

Memory can be logically divided into many parts. ~~But here we a~~

But in our system two parts.

1) Instructs (on which instructions are going to Executed).
2) operand



Stored Progeam Organisation

The simplest way to organise a computer is to have one processor register and instruction code format with two parts. The first part specifies the operation to be performed and second part specifies an address.

→ The memory address tells the control where to find an operand in memory.

→ This operand is read from memory and used as the data to be operated on together with data stored in the processor register.

→ Instructions are stored in one section of memory and data in another.

here 4096 words means these need 12 bits to specify address.

$$2^{12} = 4096 \text{ words.}$$

→ If we store each instruction code in one 16 bit memory word, we have available four bits for the operation code (opcode) to specify one out of 16 possible operations and 12 bits to specify the address of an operand

$$2^4 = 16 \text{ operations.}$$

So finally it execute the operation specified by the operation code.

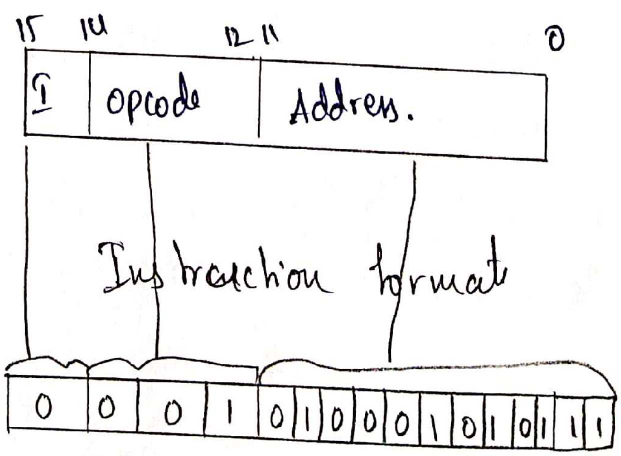
→ In computer the operation is performed with memory operand and the contents of AC (Accumulator).

Indirect Address

→ When the second part of an instruction code specifies an operand, the instruction is said to have immediate operand.

→ When the second part specifies the address of an operand, the instruction said to have a direct address.

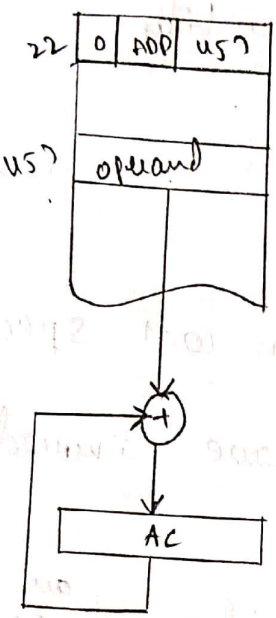
→ When the bits in the second part of the instruction designate an address of a memory word in which the address of the operand found. This is indirect address.



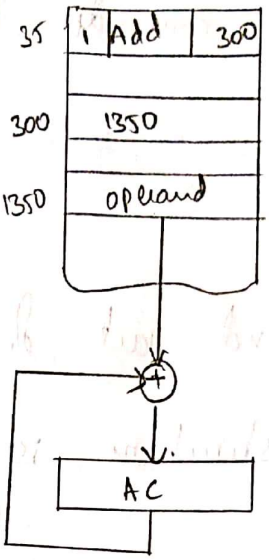
Add Instruction — ADD 457

here I – direct (0)
– Indirect (1)

Direct & Indirect Addressing of memory



Direct Addressing



Indirect Addressing

Computer Registers

Computer Instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it, and so on. This type of instruction sequencing needs a counter to calculate the address to the next instruction after execution of the current instruction is completed. It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory.

The computer needs processor register for manipulating data and a register for holding a memory address.

List of Registers for the Basic computer

Register Symbol	Number of bits	Register Name	function.
DR	16	Data Register	Holds memory operand.
AR	12	Address Register	Holds Address for the memory
AC	16	Accumulator	Processor Register
IR	16	Instruction Register	Holds instruction code
PC	12	Progeam counter	Holdsaddress of an instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	output Register	Holds output character.

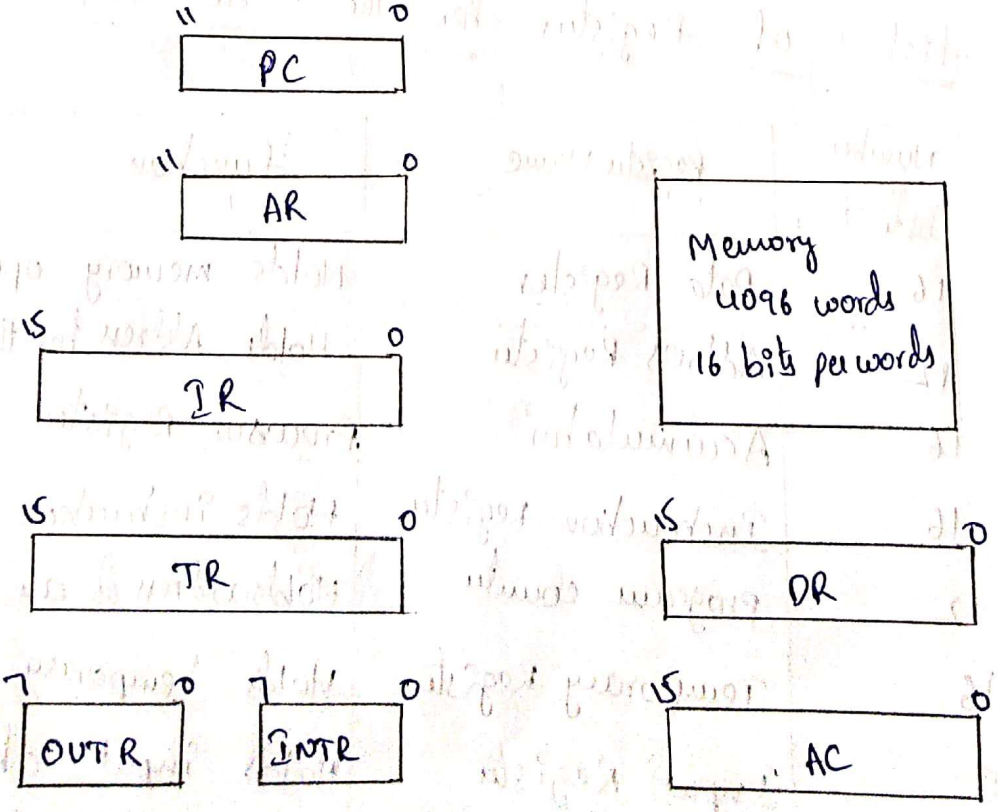
Program counter:- PC is 12 bits and it hold the address of the next instruction to be read from memory after the current instruction is executed. When the instruction is fetched, the value of Program counter is incremented.

The contents of PC can be changed by

1) unconditional Jumps
2) conditional Jumps
3) Sub routines.

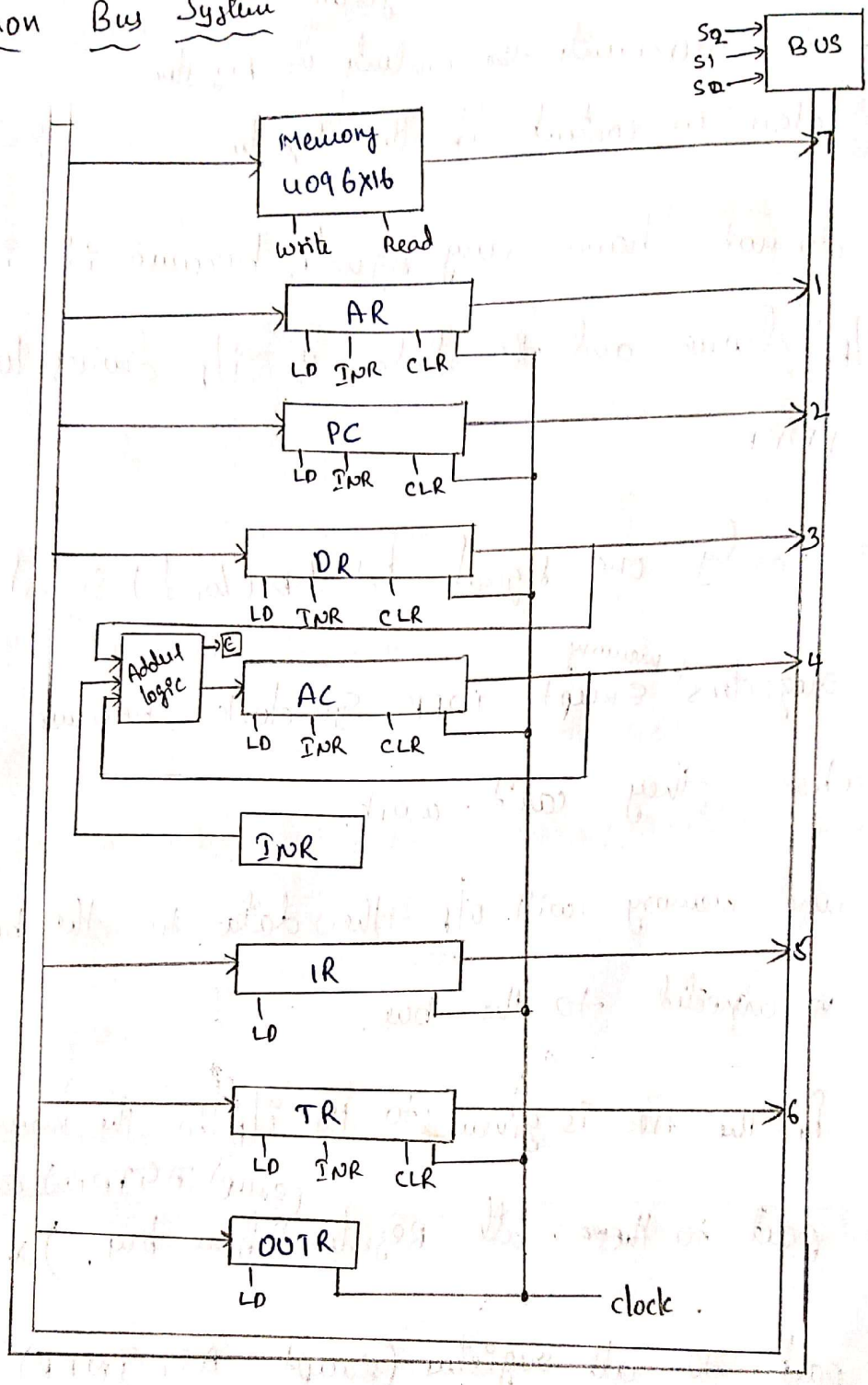
Note: when branch instruction calls to transfer to a ~~nonseq~~ nonconsecutive instruction in the program.

The address part of branch instruction is transferred to PC to become the address of the next instruction.



Basic computer registers and memory

Common Bus System



All thes memory and Registers have some signals.

In memory we have two signals read & write signals.

AR
PC
TR } have LD, INR, CLR signals.

LD – load the contents to Register

INR – is increments the contents of Register

CLR – clear the content of the Register.

→ INPR do not have any signals. because it is connected to the i/p device and the data of i/p devices loaded into INPR.

→ IR have only one signal to LD (load) signal.

→ All the registers & memory except INPR Syn clock. because without Synchronization they can't work.

→ Registers and memory will o/p the data to the bus. i.e they are connected to the bus.

→ The o/p of the AR is given to the i/p to the memory.

~~→ The i/p part to these all Registers from bus.~~ (except AC, INPR) came

→ The i/p part to all registers (except AC, INPR) came from bus.

→ AC :- ADDer & logic unit have three sets of i/ps

1. from DR
2. from Accumulator itself

→ Adder & logic unit will give o/p to the Accumulator.

→ Adder & logic unit have (E) → will store carry or carry bit.

(i.e. the end carry out of the addition is transferred to flipflop E (Extended AC bit).

Note:- The content of any Register can be applied on to the bus and operation can be performed in the adder & logic circuit during the same clock cycle.

The clock transition at the end of the cycle transfers the contents of the bus into the designated destination register and the output of the adder and logic circuit into AC.

for Example.

the two microoperations

DR ← AC & AC ← DR can be executed at the same time.

→ Bus is multiplexer. There are 3 multiplexer signal S_1, S_2, S_3. depend on S_0 S_1 S_2 the o/p line of any register can be selected.

$S_0 S_1 S_2 = 011 = 3$ then o/p from DR can be transferred to the bus.

→ $S_0 S_1 S_2 = 111 = 7$, the memory data would be transferred to

Ex: Transfer data from memory to AC.

1) loading contents of memory on bus by providing select lines

$S_0S_1S_2 = 111$.

2) Now the bus have contents of memory.

3) No we have to load the data into the data register by enabling load signal of DR. (Intermediate step).

4) Now change the select lines signal

$S_0S_1S_2 = 011$.

Then contents of DR should could transfered to new bus as well as Adder & logic unit.

5) finally contents of Adder & logic unit could be transfered to Accumulator by enabling load (LD) of Accumulator.

i.e. this can be done by placing the content of AC of the bus (with $S_2S_1S_0 = 100$), enabling the LD(load) i/p of DR, transfering the content of DR through the adder & logic circuit into AC, and enabling the LD (load) i/p of AC, all during the same clock cycle. The two transfer occur upon the arrival of the clock pulse transition at [illegible] of the clock cycle.

Computer Instructions:-

The basic computer has three instruction code formats. Each format has 16 bits.

→ The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

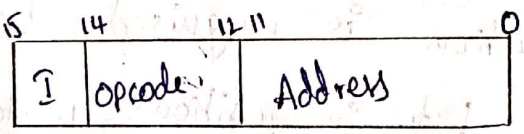
1) Memory Reference Instruction

A memory reference instruction uses 12 bits to specify an address and one bit to specify the two addressing mode I.

I = 0 for direct address

I = 1 for Indirect address.

~~2) Register Reference~~



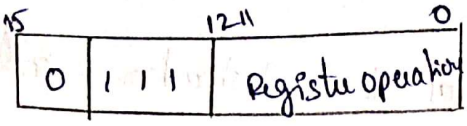
(opcode = 000 through 110).

Memory - Reference Instruction

2) Register Reference Instructions:- These are recognized by the operation code 111 with a '0' in the leftmost bit of the instruction.

→ A register reference instruction specifies an operation on or test of the AC register.

→ An operand from memory is not needed. ∴ the other 12 bit

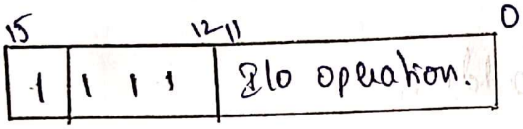


Register Reference instruction.

3) Input - output Instruction :-

These instruction does not need a reference to memory and is recognised by the operation code 111 with a '1' in the left most bit of the instruction.

The remaining 12 bits are used to specify the type of i/p - o/p operation or test performed.



(c) Input - Output Instruction.

→ The type of instruction is recognised by the computer control from the four bits in positions 12 to 15 of the instruction.

→ If the three opcode bits in position 12 to 14 are not equal to 111, the instruction is a memory reference type. and the bit in position 15 is taken as the addressing mode I.

→ If the 3 bit opcode is equal to 111, control then inspects the bit in position 15.

→ If this bit < 0 — The instruction is a registe reference type
1 — The instruction is an i/p - o/p type.

→ only 3 bits are used for operation code so eight operations are possible and in Register-Reference and i/p-o/p instruction use the remaining 12 bit as part of operation code, so the total no. of instruction can exceed eight.

→ The total no. of instructions choosen for the basic computer is Equal to 25.

Hexadecimal code for Instructions

The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

→ By using the hexadecimal Equivalent we reduced the 16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits.

1) Memory Reference Instructions:- It has an address part of 12 bits. The address part is denoted as 3-x's

→ The last bit of the instruction is designated by the Symbol I.

→ when I = 0, the last four bits of instruction have a hexadeci

→ When I=1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E

Symbol	Hexadecimal code I=0	Hexadecimal code I=1	Description.
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero.

2) Register Reference Instructions.

Register Reference Instructions use 16 bits to specify an operation. The leftmost four bits are always 0111 which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12 bits

Symbol	Hexadecimal code	Description
CLA	7800	clear AC
CLE	7400	clear E
CMA	7200	complement AC

CIR	7080	Circulate Right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	skip next instruction if AC positive
SNA	7008	Skip next instruction if AC negative
SZA	7004	Skip next instruction if AC zero
SZE	7002	skip next instruction if E is zero
HLT	7001	Halt computer.

The Input-output instruction It also use 16 bits to specify an operation. The last four bits are always 1111, Equivalent to hexadecimal F.

Symbol	Hexadecimal	Description.
INP	F800	Input character to AC
OUT	F400	output character from AC.
SKI	F200	Skip on input flag
SKO	F100	Skip on output flag.
ION.	F080	Interrupt on
IOF	F040	Interrupt off

Instruction Set completeness :-

A computer should have a set of instructions so that the user can construct machine language program to evaluate any function that is known to be computable.

The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories.

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

Timing and Control

→ The timing for all registers in the basic computer is controlled by a master clock generator.

→ The clock pulses are applied to all flip flops and registers in the system, including the flip flops and registers in the control unit.

→ The clock pulses do not change the state of a register unless the register is enabled by a control signal.

→ The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations for the accumulator

→ There are two major types of control organisation

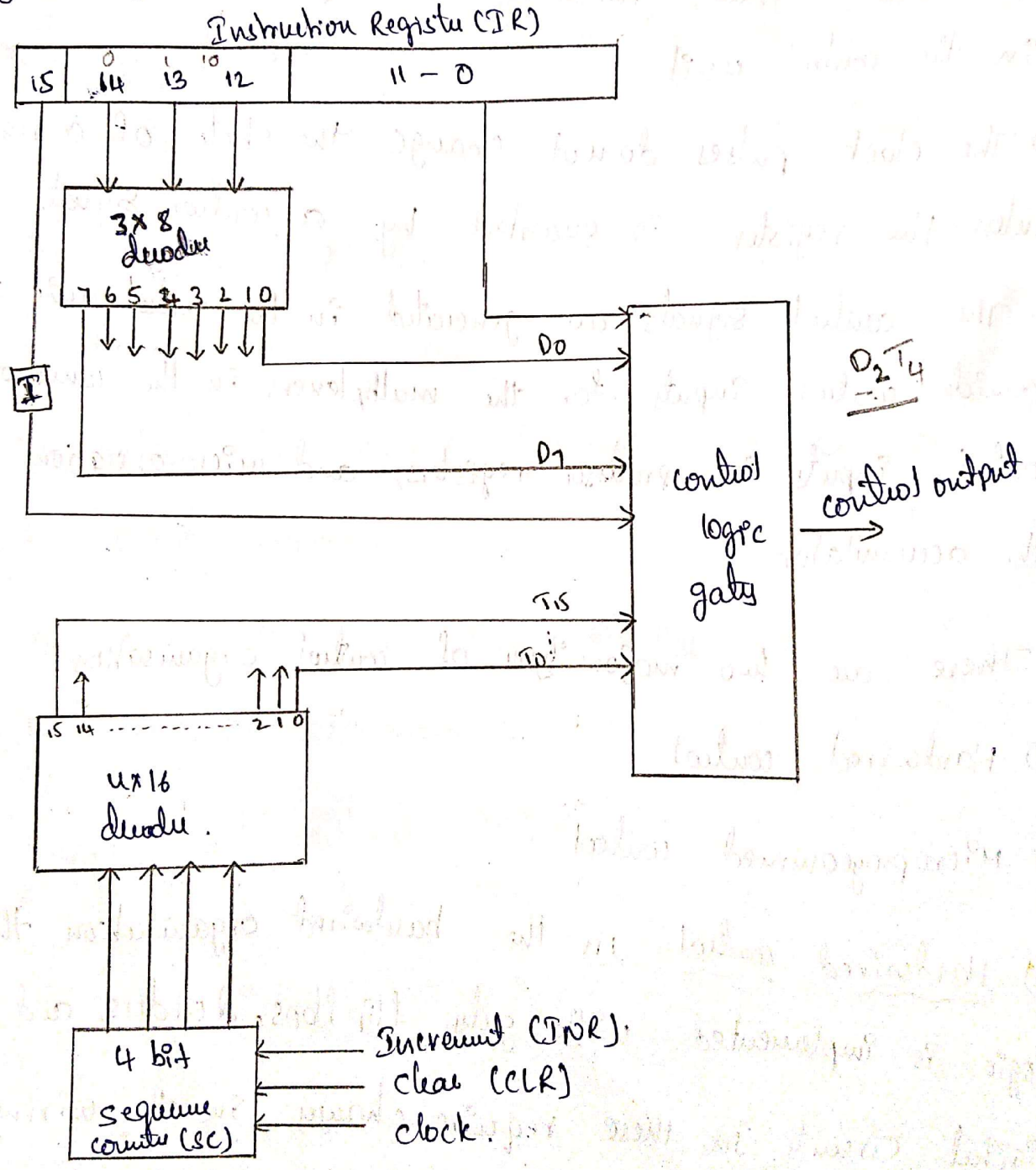
1) Hardwired control

2) Microprogrammed control

1) Hardwired control In the hardwired organisation, the control logic is implemented with gates, flip flops, decoders, and other digital circuits. It these require changes in the wiring among the various components if the design has to be modified or changed

2) Micro programmed control :- In this, the control information is stored in control memory. The control memory is programmed to initiate the required sequence of microoperations.

→ If any changes or modifications are required can be done by updating the microprogram in control memory.



control unit of basic computer

Control unit consist of two decoders, a sequence counter, and a number of control logic gates.

→ An instruction read from memory is placed in the instruction register (IR), where it divided into three parts. the I bit, the operation code, and bits 0 through 11.

→ The operation code in bits 12 through 14 are decoded with a 3x8 decoder.

→ The eight outputs of the decoder ($2^3=8$) are designed by the D_0 through D_7.

→ The subscripted decimal number is equivalent to the binary value of the corresponding operation code.

→ Bit 15 of the instruction is transferred to a flip flop ~~designed~~ designated by the symbol I.

→ Bits 0 through 11 are applied to the control logic gates.

→ The 4 bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

→ The sequence counter SC can be incremented or cleared Synchronously.

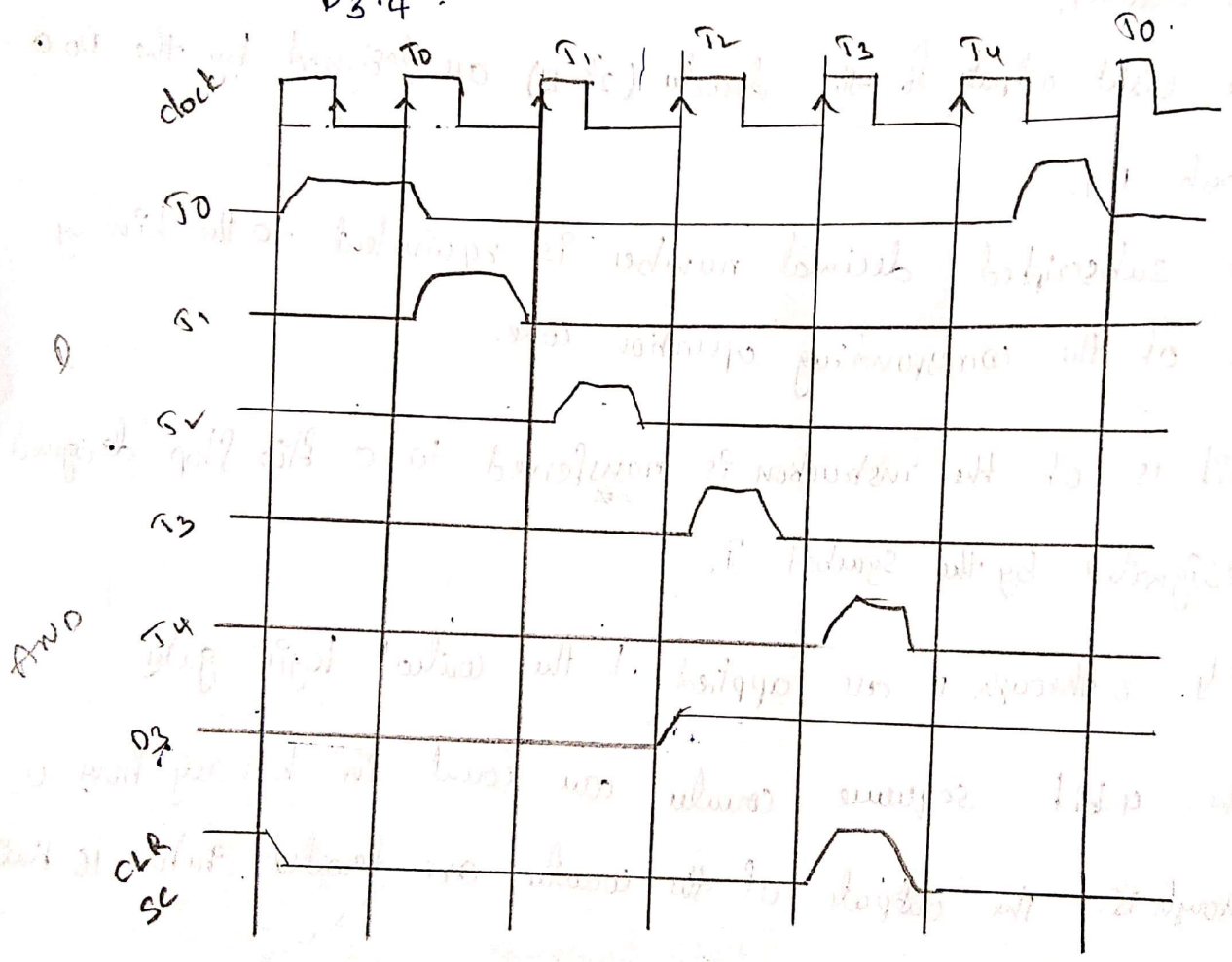
→ Most of the time, the counter is incremented to provide the sequence of timing [illegible] 4x16 decoder

→ Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.

→ As an example, consider the case where SC is incremented to provide timing signals T_0 T_1 T_2 T_3 T_4 in sequence.

→ At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically, by the statement

$D_3T_4 : SC \leftarrow 0$



Example of control signals

Sequence counter is incremented or cleared.

At the time of T_3 D_3 is active, then sequence counter

→ If SC is not cleared, the timing signals will continue with T_5, T_6 and up to T_{15} and back to T_0.

→ output D_3 from the operation decoder becomes active at the end of timing signal T_4.

→ when Timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active.

→ This signal is applied to the CLR input of SC. on the next positive clock transition the counter is cleared to 0. This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Instruction Cycle:-

A program residing in the memory unit of the computer consist of a sequence of instructions.

→ The program is executed in the computer by going through a cycle for each instruction.

→ Each instruction cycle in turn is subdivided into a sequence of subcycle or phases.

→ In the basic computer each instruction cycle consist of the following phases.

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

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continuous indefinitely unless a HALT instruction is encountered.

Fetch and Decode :-

Initially, the program counter PC is loaded with the address of the first instruction in the program.

→ The sequence counter SC is cleared to 0, providing a decoding timing signal T_0.

→ After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2 and so on.

→ The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0 : AR ← PC

T_1 : IR ← M[AR] PC ← PC + 1

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

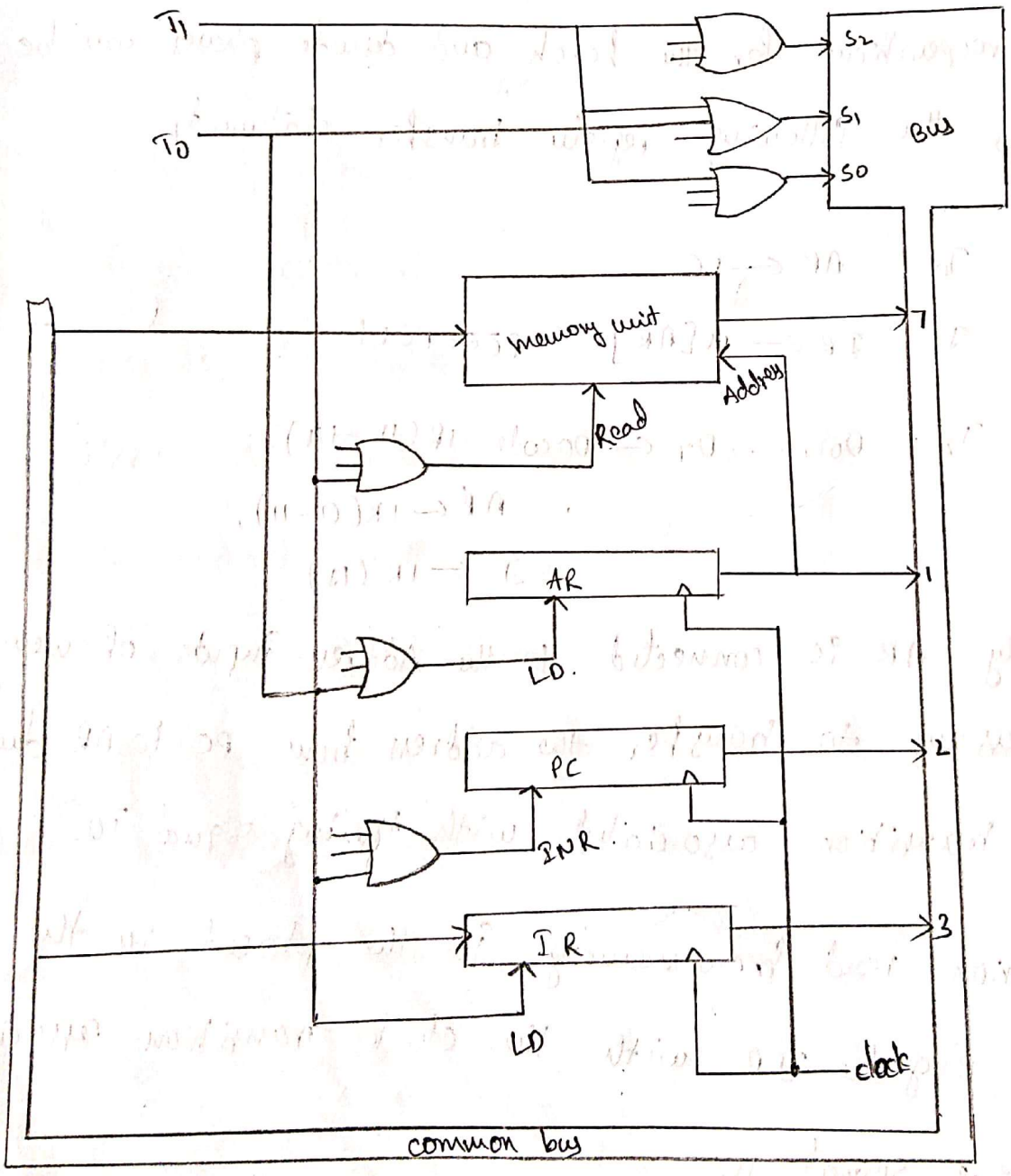
→ Since only AR is connected to the Address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T_0.

→ The instruction read from memory is the placed in the instruction Register IR with the clock transition associated with timing signal T_1.

→ At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.

→ At the time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip flop I, and the address part of the instruction is transferred to AR.

→ Note that SC is incremented after each clock pulse to produce the sequence T_0, T_1, and T_2.



Register transfers for the fetch phase.

The above figure shows how the first two register transfer stmts are implemented in the bus system.

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

1) place the content of PC into onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.

2) Transfer the content of the bus to AR by enabling the LD input of AR.

→ The next clock transition initiates the transfer from PC to AR since $T_0 = 1$

In order to implement the second stmt.

T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$.

→ It is necessary to use timing signal T_1 to provide the following connections in the bus system.

1) Enable the read i/p of memory

2) place the content of memory onto the bus by making $S_2S_1S_0 = 111$.

3) Transfer the content of the bus to IR by enabling the LD input of IR.

4) Increment PC by one

The next clock transition initiates the read and increment operations since $T_1 = 1$.

Determine the type of Instruction

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

→ The figure presents initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.

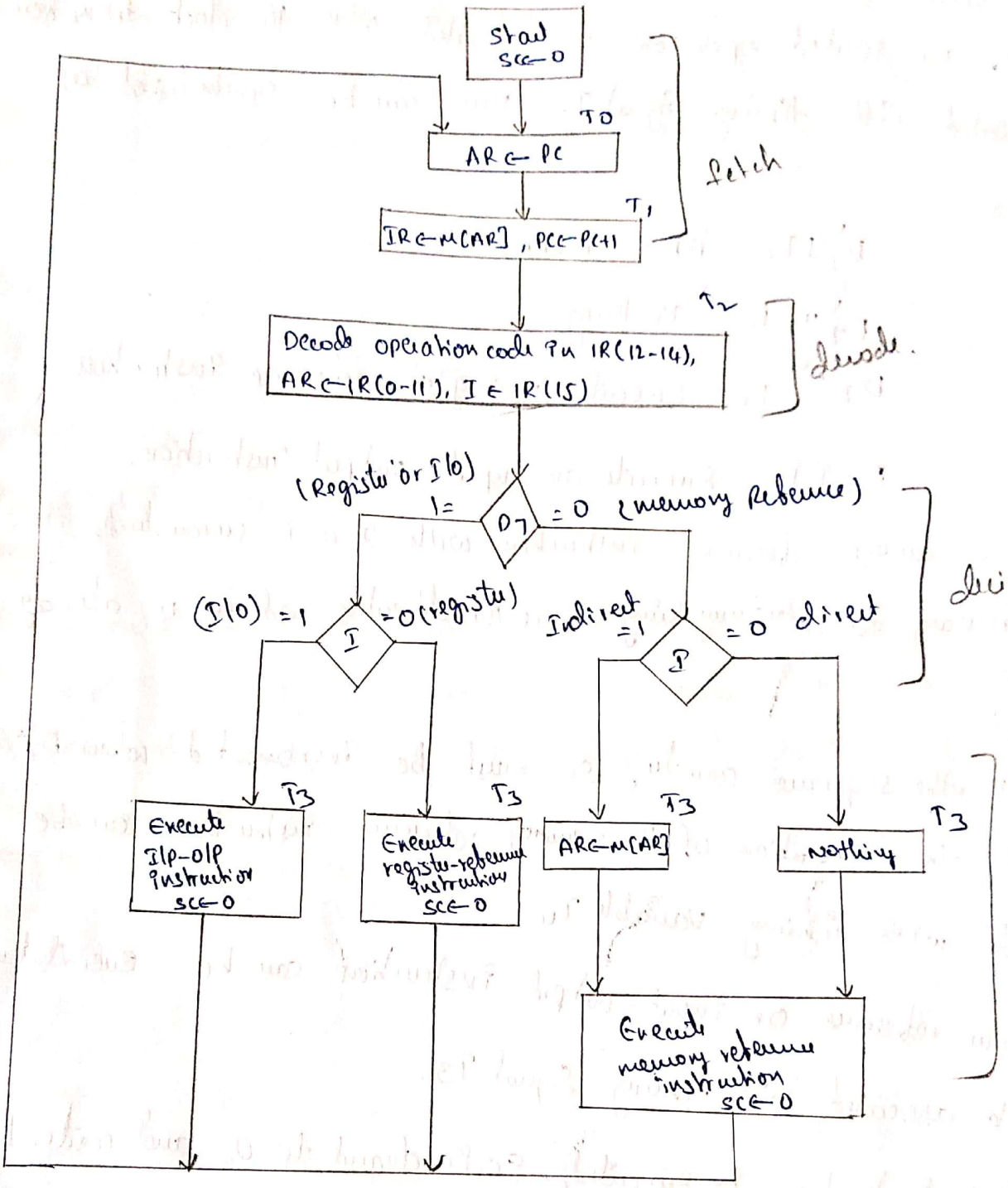
→ Decode o/p D_7 is equal to 1 if the operation code is equal to binary 111. so if $D_7 = 1$, the instruction must be register-reference or i/p-o/p type.

→ If $D_7 = 0$, the operation code must be one of the other seven through 000 through 110, specifying a memory reference instruction.

→ control then inspects the value of the first bit of the instruction, which is now available in flip flop I.

→ If $D_7 = 0$ and $I = 1$, we have a memory reference instruction with an indirect address.

→ It is then necessary to read the effective address from memory. The microoperation for the indirect address condition can be symbolized by the register transfer stmt



Flow chart for instruction cycle (initial configuration.

Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD i/p of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows.

$D_7' I T_3$: AR ← M[AR]

$D_7' I' T_3$: Nothing

$D_7 I' T_3$: Execute a register reference instruction

$D_7 I T_3$: Execute an input-output instruction.

When a memory reference instruction with $I=0$ is encountered, it is not necessary to do anything since the effective address is already in AR.

However, the sequence counter SC must be incremented when $D_7' T_3 = 1$, so that the execution of the memory reference instruction can be continued with timing variable T_4.

A register reference or input-output instruction can be executed with the clock associated with timing signal T_3.

→ After the instruction is executed, SC is cleared to 0, and control returns to the fetch phase with $T_0 = 1$.

→ Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. Here we no need to write SC ← SC+1, but it will be implied that the control goes to the next signal in sequence.

→ When SC is to be cleared, we will include the stmt SC ← 0.

Register Reference Instructions

Register reference instructions are recognised by the control when $D_7 = 1 + I = 0$.

→ The remain 12 bit are avaiable in IR(0-11). They transfered to A during time T_2.

→ These instructions are executed with the clock transition associated with timing variable T_3.

→ Each control function needs the Boolean relation $D_7 \bar{I} T_3$, is denoted by 'r'.

→ The control function is decided by one of the bits in IR(0-11).

→ By assigning the symbol B_i to bit i of IR, all control function can be simply denoted by rB_i.

Eg: CLA has hexadecimal code 7800. = 0111 1000 0000 0000.
(I, D_7, B_{11})

∴ The control function that initiate the microoperation for this instruction is $D_7 \bar{I} T_3 B_{11} = rB_{11}$. The execution of a register reference instruction is completed at time T_3 if The $SC \leftarrow 0$ & the control goes back to fetch the next instruction with timing signal T_0.

---

	r:	$SC \leftarrow 0$	clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	clear AC
CLE	rB_{10}:	$E \leftarrow 0$	clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	complement AC
CME	rB_8:	$E \leftarrow \bar{E}$	complement E
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	increment AC
SPA	rB_4:	if $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	skip if negative
SZA	rB_2:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	skip if AC zero.
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start-stop flip flop)	Halt computer

The first seven register reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E register. The next four instructions cause

Memory Reference Instructions

In order to specify the microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.

Consider seven memory reference instructions. The decoded output D_i for $i = 0, 1, 2, 3, 4, 5$ and 6 from the operation decoder that belongs to each instruction is included in the table.

Symbol	Operation decoder	Symbol Description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow$ Cout
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, if $M[AR] + 1 = 0$ then $PC \leftarrow PC+1$.

The effective address of the instruction is in the address register AR and was placed there during timing signal T_2

The Execution of the memory reference instructions start with timing signal T_4.

AND to AC :-

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

D_0T_4 : $DR \leftarrow M[AR]$

D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$.

→ The control function for this instruction uses the operation decoder D_0 since this o/p of the decoder is active when the instruction has an AND operation whose binary code value is 000.

→ Two timing signals are needed to execute the instruction.

→ The clock transition associated with timing signals T_4 transfers the operand from memory into DR.

→ The clock transition associated with the next timing signal T_5 transfer to AC the result of the AND logic operation b/w the contents of DR and AC.

→ The same clock transition clears SC to 0, transferring control

ADD to AC This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and o/p carry Cout is transferred to the E (Extended accumulator) flip-flop.

The micro operations needed to execute this instruction are

$D_i T_4 : DR \leftarrow M[AR]$

$D_i T_5 : AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \; SC \leftarrow 0.$

The same two timing signals, T_4 and T_5 are used again but with operation decoder D_1 instead of D_0, which was used for the AND instruction.

→ After the instruction is fetched from memory and decoded, only one o/p of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory reference instruction.

LDA : Load to AC

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

$D_2 T_4 : DR \leftarrow M[AR]$

In common bus system as we shown in figure there is no direct path from the bus in AC.

→ The adder and logic circuit receive information from DR which can be transferred into AC.

→ Therefore, it is necessary to read memory word into DR first and then transfer the content of DR into AC.

→ The reason for not connecting the bus to the inputs of AC is the delay encountered in the adder and logic circuit. It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle. By not connecting the bus to the inputs of AC we can maintain one clock cycle per microoperation.

STA: store AC

This instruction stores the contents of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation

$$D_3T_4: M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle.

→ PC is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence.

→ The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.

→ The instruction is executed with one microoperation

$$D_4T_4: PC \leftarrow AR,\ SC \leftarrow 0.$$

→ The effective address from AR is transferred through the common bus to PC. Resetting SC to 0 transfers control to T_0. The next instruction is then fetched and executed from the memory address given by the new value in PC.

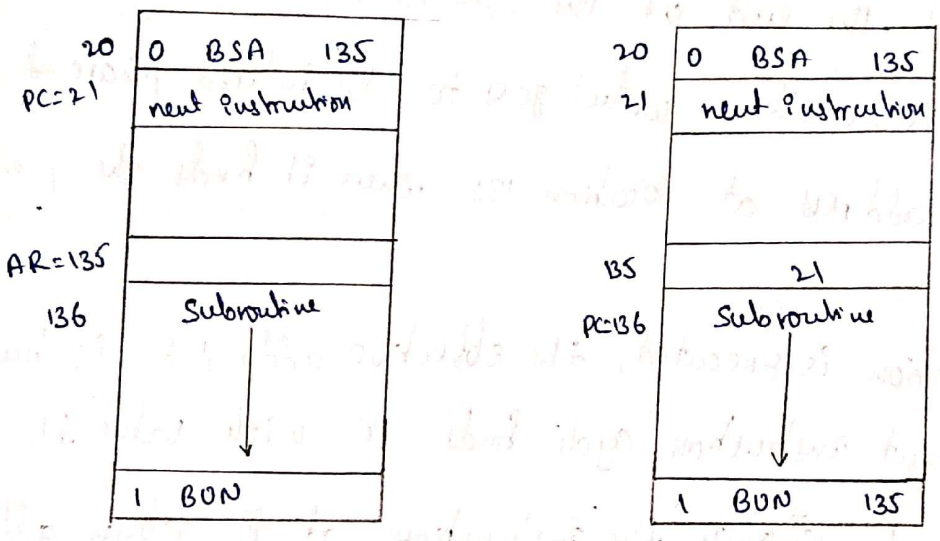
BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure.

The BSA instruction stores the address of the

next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the Subroutine. This operation was specified by following register transfer:

$$M[AR] \leftarrow PC, \quad PC \leftarrow AR+1.$$



(a) Memory PC, and AR at time T_4 (b) Memory and PC after execution.

Example of BSA instruction Execution

A numerical example shown in above table.

→ The BSA ~~function~~ instruction is assumed to be in memory at address 20.

→ The I bit is 0 and the address part of the instruction has the binary equivalent of 135.

→ After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program. AR hold effective

The BSA instruction performs the following numerical operations

$M[135] \leftarrow 21 \qquad PC \leftarrow 135 + 1 = 136.$

In figure 2, the return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.

When the BUN instruction is executed, the effective address 21, is transferred to PC. The next instruction cycle finds PC with value 21, so control continues to execute the instruction at the return address.

The BSA instruction performs the function usually referred to as a subroutine call. The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return.

In most commercial computers, the return address associated with a subroutine is stored in either a processor register or in a portion of memory called a stack.

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the computer.

→ To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

D_5T_4 : $M[AR] \leftarrow PC$, $AR \leftarrow AR+1$.

D_5T_5 : $PC \leftarrow AR$, $SC \leftarrow 0$.

→ Timing signal T_4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR i/p of AR. The memory write operation is completed and AR is incremented by the time the next clock transition occurs.

→ The bus is used at T_5 to transfer the content of AR to PC.

ISZ : Increment and Skip if zero.

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.

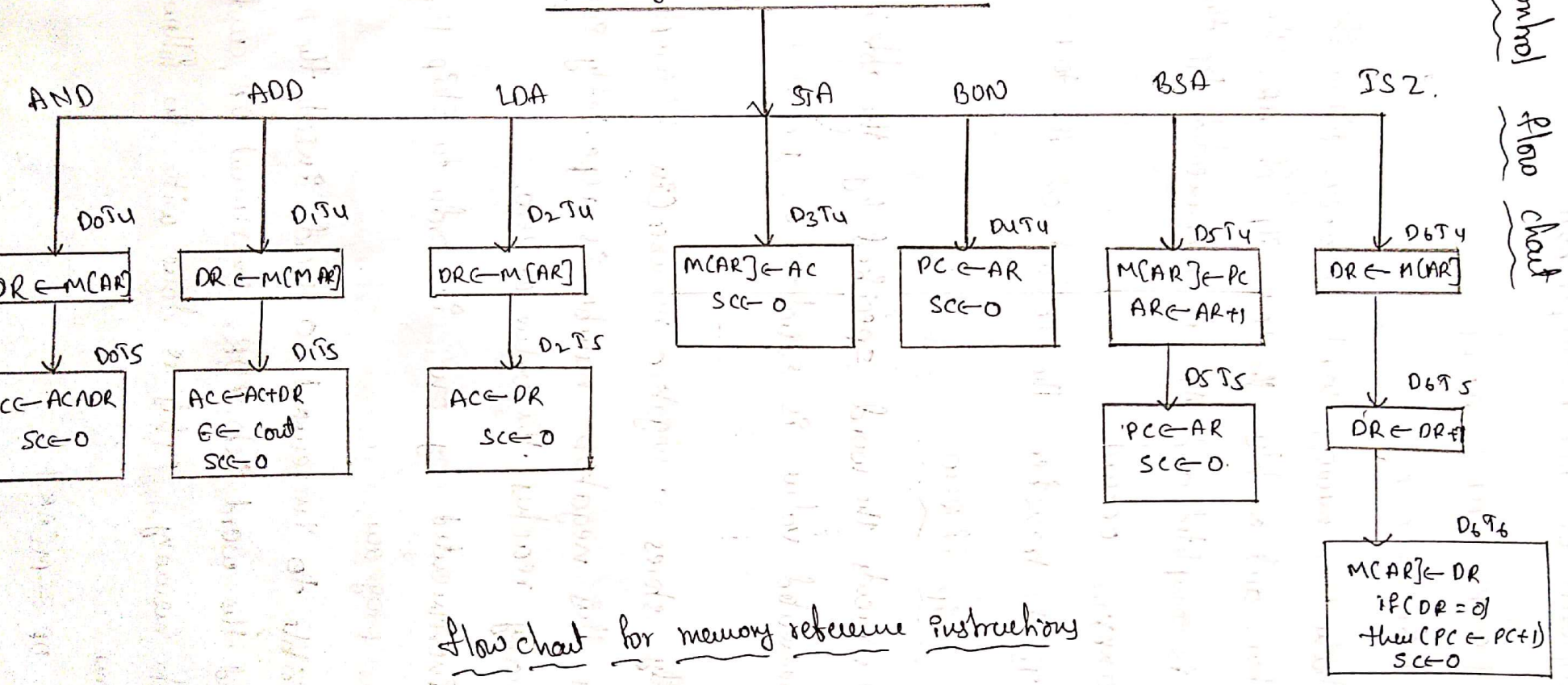
→ The programmer usually stores a negative number (in 2's complement) in the memory word, as this negative number is repeatedly incremented by one, if eventually reaches the value of zero.

→ At that time PC is incremented by one in order to skip the next instruction in the program.

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

D_6T_4 : $DR \leftarrow M[AR]$

D_6T_5 : $DR \leftarrow DR+1$



flow chart for memory reference instructions

Control flow chart

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

→ The control functions are indicated on top of each box. The microopera-tions that are performed during time T_4, T_5, T_6 depend on the operation code value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded.

→ The sequence counter SC is cleared to 0 with the last timing signal in each case. This causes a transfer of control to timing signal T_0 to start the next instruction cycle.

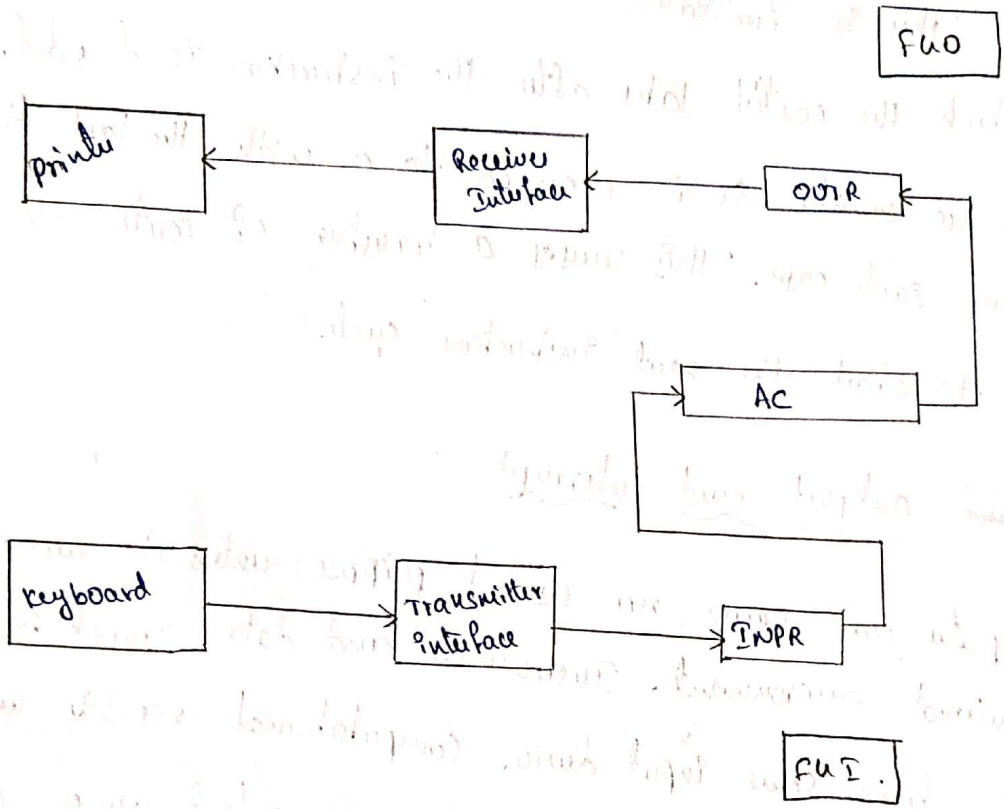
Input and output and Interrupt

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results ~~may~~ must be transmitted to the user through some output device. Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for I/P & O/P communication, we will use as a terminal unit with a key board & printer.

Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information for from the key board is shifted into the input register INPR. The serial information ~~for the keyboard is shifted to~~ printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel.

Input-output terminal | Serial communication interface | Computer Registers and Flip Flops.



Input - output configuration

The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.

→ The input register INPR consists of eight bits and holds an alpha numeric input information.

→ The 1-bit input flag FLI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.

→ The flag is needed to synchronize the timing rate difference b/w the input device and the computer.

→ The process of information transfer is as follows. Initially, the input flag FGI is cleared to 0.

→ When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.

→ As long as the flag is set, the information in INPR cannot be changed by striking another key.

→ The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.

→ Once the flag is cleared, new information can be shifted into INPR by striking another key.

→ The output register OUTR works similarly but the direction of information flow is reversed.

→ Initially the output flag FGO is set to 1.

→ The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The o/p device accepts the coded information, prints the corresponding character, and when the operation is completed it sets FGO to 1. The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

Input - output Instructions:-

Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.

→ Input - output instructions have an operation code 1111. and recognised by the control when $D_7 = 1$ and $I = 1$. The remaining bits of the instruction specify the particular operation.

→ These instructions are executed with the clock transition associated with timing signal T_3.

→ Each control function needs a Boolean relation $D_7 I T_3$, which we designate for convenience by the symbol p.

→ The control function is distinguished by one of the bits in IR(6-11)

→ By assigning the symbol B_i to bit i of IR, all control functions can be denoted by pB_i for $i = 6$ through 11.

→ The sequence counter SC is cleared to 0 when $p = D_7 I T_3 = 1$.

→ The INP instruction transfers the i/p information from INPR into the eight low-order bits of AC and also clears the input flag to 0.

→ The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0.

→ The next two instructions check the status of the flags and cause a skip of the next instruction if the flag is 1.

→ The instruction that is skipped will ~~normally~~ normally be a branch instruction to return and check the flag again.

→ The branch instruction is not skipped if the flag is 0.
→ If the flag is 1, the branch instruction is skipped and an i/p or output instruction is executed
→ The last two instructions set and clear an interrupt enable flip flop IEN.

$D_7 I T_3 = P$ (common to all i/p - output instructions)
$IR(i) = B_i$ (bit in IR (6-11) that specifies the instruction)

	P :	SC ← 0	clear SC
INP	PB_{11} :	AC (0-7) ← INPR, FGI ← 0	Input characte
OUT	PB_{10} :	OUTR ← AC (0-7), FGO ← 0	output characte
SKI	PB_9 :	If (FGI = 1) then (PC ← PC+1)	skip on i/p flag
SKO	PB_8 :	If (FGO = 1) then PC ← PC+1)	skip on o/p flag
ION	PB_7 :	IEN ← 1	Interuppt enable on
IOF	PB_6 :	IEN ← 0	Interuppt enable off.

Progeam Interrupt

The procss of communi cation just described is referred to as progeammed control transfer.

→ The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
→ The difference of information flow rate btw the computer and that of the i/p-o/p devie makes this type of transfer inefficient.
→ consider a computer that can go through an instruction cycle in

→ Assume that the I/O device can transfer information at a maximum rate of 10 characters per second.

⇒ This is equivalent to one character every 100,000 μs.

1 sec = 1000000 μs.

→ Two instructions are executed when the computer checks the flag bit and decides not to transfer the information.

⇒ This means that at the maximum rate, the computer will check the flag 50,000 times b/w each transfer.

→ The computer is wasting time while checking the flag instead of doing some other useful processing task.

⇒ An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.

→ In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.

→ while the computer is running a program, it does not check the flags.

⇒ However, when a flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set.

→ The computer deviates from what it is doing to take care of the i/p or o/p transfer.

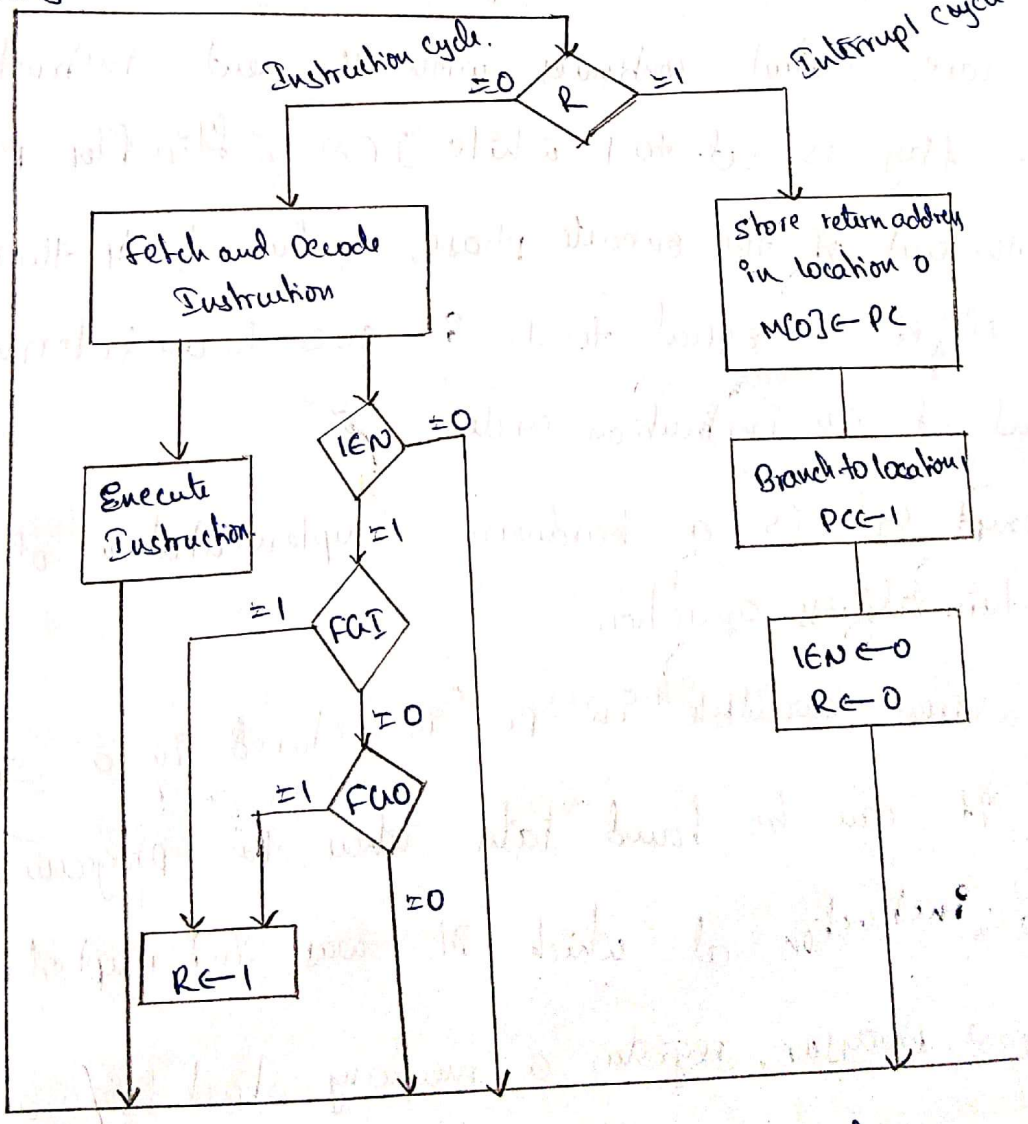
→ It then returns to the current program to continue what it was doing before the interrupt.

→ The interrupt enable flip flop IEN can be set and cleared with two instructions.

→ when IEN is cleared to 0. (with the IOF instruction), the flags cannot interrupt the computer.

→ When IEN is set to 1 (with the ION instruction), the computer can be interrupted.

→ These two instructions provide the programmer with capability of making a decision as to whether or not to use the interrupt facility.



flowchart for interrupt cycle.

→ An interrupt flip flop R is included in the computer.

→ When R=0, the computer goes through an instruction cycle.

→ During the execute phase of the instruction cycle. IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.

→ If IEN is 1, control checks the flag bits,

→ If both flags are 0, it indicates that neither the i/p nor the o/p registers are ready for transfer of information.

→ In this case, control continues with the next instruction cycle.

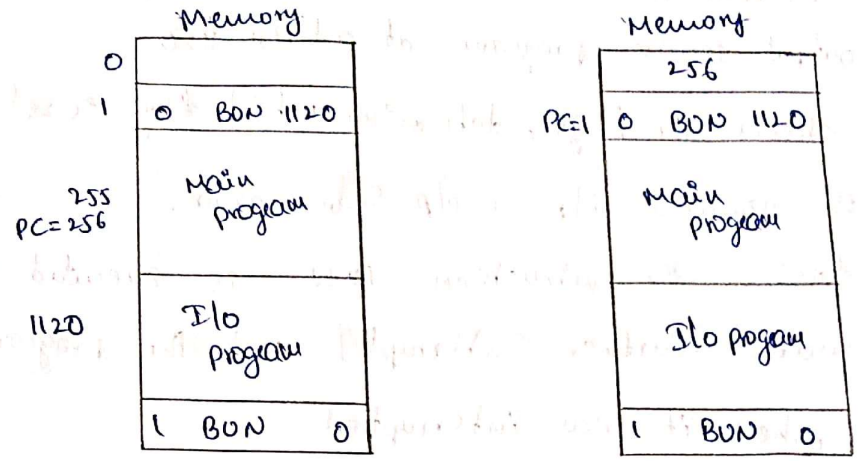
→ If either flag is set to 1 while IEN=1, flip flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

→ The interrupt cycle is a hardware implementation of a branch and save return address operation.

→ The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor, register, a memory stack or a specific memory location.

→ Here we choose the memory location at address 0 as the place for

→ control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.



(a) Before Interrupt (b) after Interrupt cycle.

Demonstration of the interrupt cycle.

Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.

→ At this time the return address 256 is in PC. The programmer has previously placed an i/p-o/p service program in memory starting from address 1120 and a BUN 1120 instruction at address

→ when control reaches timing signal T0 and finds that R= it proceeds with interrupt cycle.

→ The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.

→ At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC.

→ The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.

→ This program checks the flags, determines which flag is set, and then transfers the required I/P or O/P information.

→ Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts) and the program returns to the location where it was interrupted.

→ The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O Service program.

→ After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I=1) to read the effective address.

→ The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle.

→ The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Interrupt Cycle:-

The interrupt cycle is initiated after the last execute phase if the interrupt flip flop R is equal to 1.

→ This flip flop is set to 1 if IEN=1 and either FAI or FAO are equal to 1.

→ This can happen with any clock transition except when timing signals T_0, T_1, or T_2 are active.

→ The condition for setting flip flop R to 1 can be expressed with the following register transfer stmt:

$$T_0' T_1' T_2' (IEN)(FAI + FAO) : R \leftarrow 1.$$

→ The symbol + b/w FAI and FAO in the control function designates a logic OR operation. This is ANDed with IEN and $T_0' T_1' T_2'$.

→ we now modify the fetch and decode phases of the instruction cycle. (~~Instead of using only timing signals with R' so that the fetch and~~

→ Instead of using only timing signals T_0, T_1 and T_2 we will AND the three timing signals with R' so that the fetch and decode phases will be recognised from three control function $R'T_0$, $R'T_1$, and $R'T_2$.

→ The reason for this is that after the instruction is executed and SC is cleared to 0, the control will go through a fetch phase

→ Otherwise, if R=1, the control will go through an interrupt cycle.

→ The interrupt cycle stores the return address (available in PC) into memory location 0, branches to memory location 1, and clears IEN, R, and SC to 0.

⇒ This can be done with the following sequence of microoperations

RT_0 : AR ← 0, TR ← PC

RT_1 : M[AR] ← TR, PC ← 0

RT_2 : PC ← PC+1, IEN ← 0, R ← 0, SC ← 0

→ During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR.

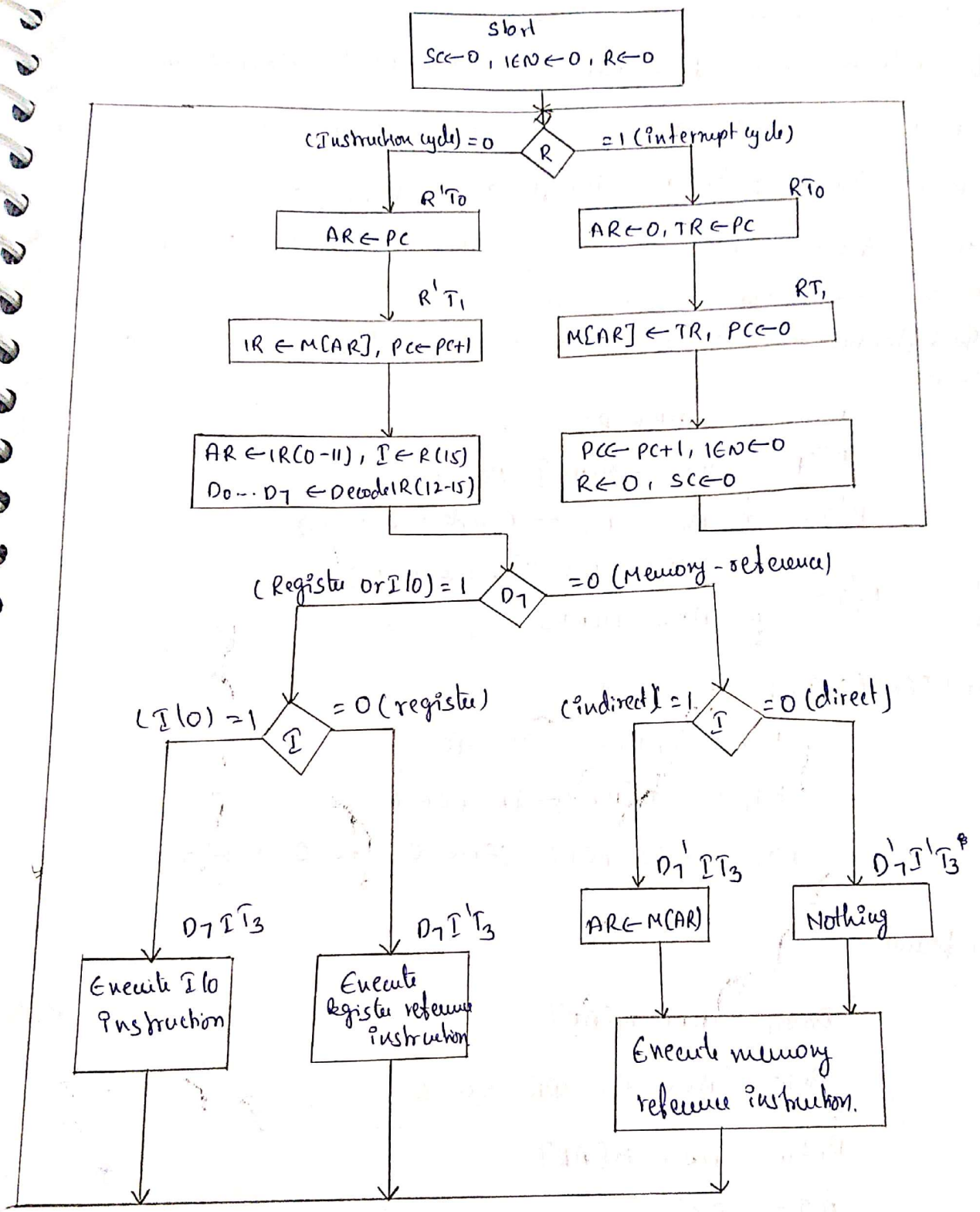
→ With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.

→ The third timing signal increments PC to 1, clears IEN and R and control goes back to T_0 by clearing SC to 0.

→ The beginning of the next instruction cycle has the conditions $R'T_0$ and the content of PC is equal to 1.

→ The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

Complete Computer Description



flow chart for computer operation.

The final flowchart of the instruction cycle, including the interrupt cycle for the basic computer shown above.

→ The interrupt flip flop R may be set at any time during the indirect or execute phases.

→ Control Returns to timing signal T_0 after SC is cleared to 0.

→ If R = 1, the computer goes through an interrupt cycle.

→ If R = 0, the computer goes through an instruction cycle.

→ If the instruction is one of memory reference instruction, then that is Executed.

→ If the instruction is one of register reference instruction, then that is Executed.

→ If the instruction is one of I/O reference instruction, then that is Executed.

Fetch $R'T_0$: $AR \leftarrow PC$

$R'T_1$: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$.

Decode $R'T_2$: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12-14)$

$AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$.

Indirect $D_7'IT_3$: $AR \leftarrow M[AR]$

Interrupt:

$T_0'T_1'T_2'(IEN)(FGI + FGO)$: $R \leftarrow 1$

RT_0 : $AR \leftarrow 0$, $TR \leftarrow PC$

RT_1 : $M[AR] \leftarrow TR$, $PC \leftarrow 0$

RT_2 : $PC \leftarrow PC+1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$.

Memory reference:

AND D_0T_4 : $DR \leftarrow M[AR]$

D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

ADD D_1T_4 : $DR \leftarrow M[AR]$

D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow 0$

LDA D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR$, $SC \leftarrow 0$

STA D_3T_4 : $M[AR] \leftarrow AC$, $SC \leftarrow 0$

BUN D_4T_4 : $PC \leftarrow AR$, $SC \leftarrow 0$

BSA D_5T_4 : $M[AR] \leftarrow PC,\ AR \leftarrow AR+1$

D_5T_5 : $PC \leftarrow AR,\ SC \leftarrow 0$

ISZ D_6T_4 : $DR \leftarrow M[AR]$

D_6T_5 : $DR \leftarrow DR+1$

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

Register Reference:

$D_7I'T_3 = r$ (common to all register-reference instructions)

$IR(i) = B_i\ (i = 0,1,2,3,\ldots,11)$

r: $SC \leftarrow 0$

CLA	rB_{11}	$AC \leftarrow 0$
CLE	rB_{10}	$E \leftarrow 0$
CMA	rB_9	$AC \leftarrow \overline{AC}$
CME	rB_8	$E \leftarrow \overline{E}$
CIR	rB_7	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5	$AC \leftarrow AC+1$
SPA	rB_4	If $(AC(15)=0)$ then $(PC \leftarrow PC+1)$
SNA	rB_3	If $(AC(15)=1)$ then $(PC \leftarrow PC+1)$
SZA	rB_2	If $(AC=0)$ then $(PC \leftarrow PC+1)$
SZE	rB_1	If $(E=0)$ then $(PC \leftarrow PC+1)$
HLT	rB_0	$S \leftarrow 0$

Input output:

$D_7IT_3 = p$ (common to all I/O instructions)

$IR(i) = B_i\ (i = 6,7,8,9,10,11)$

p : $SC \leftarrow 0$

INP	pB_{11}	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9	If $(FGI=1)$ then $(PC \leftarrow PC+1)$
SKO	pB_8	If $(FGO=1)$ then $(PC \leftarrow PC+1)$
ION	pB_7	$IEN \leftarrow 1$
IOF		

hardwired control unit

Timing and control concept

Micro programmed control unit

Microprogramming is a second alternative for designing the control unit of a digital computer.

→ The principle of microprogramming is an elegant and systematic method for controlling the microoperation sequence in a digital computer.

→ The control function that specifies the microoperation is a binary variable. when it is one binary state, the corresponding microoperation is executed.

→ A control variable in the opposite binary state does not change the state of the registers in the system.

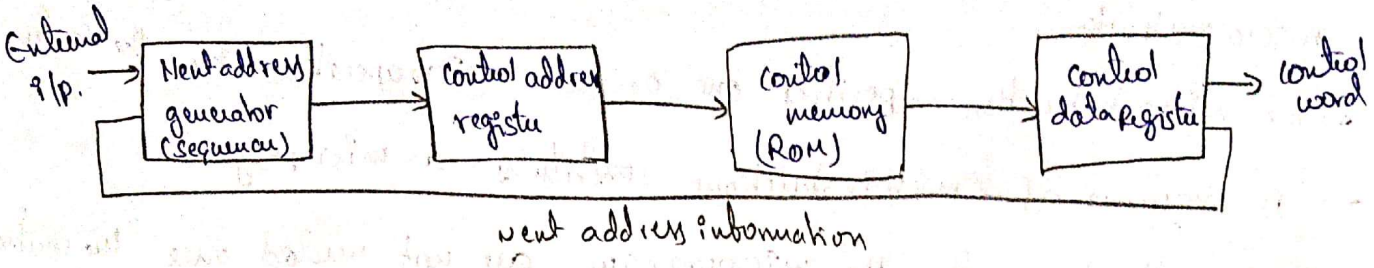
→ The active state of a control variable may be either the '1' state or the '0' state, depending on the application.

→ In a bus organised system, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.

The control unit initiates a series of sequential steps of microoperations. During any given time, certain microoperations are to be initiated, while others remain idle.

→ As such control words can be programmed to perform various operations on the components of the system.
→ A control unit whose binary control variable are stored in memory is called a microprogrammed control unit.
→ Each word in control memory contains with in it a ~~microoperation~~ microinstruction.
→ The microinstruction specifies one or more microoperations for the system.
→ A sequence of microinstructions constitutes a microprogram.
→ Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a read only memory (ROM).
→ The content of the words in memory ROM are fixed and cannot be altered by simple programming since no writing capability available in the ROM.
→ control units that use dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk.
→ control unit that use dynamic microprogramming employ a writable control memory.
→ This type of memory can be used for writing but is used mostly for reading.
→ A memory that is part of a control unit is referred to as a control memory.
→ A computer that employes a microprogrammed control unit will have two seperate memories.
1. main memory — for storing the user programs, (data can be manipulated by ALU).
2. control memory → holds a fixed microprogram that cannot be altered by the ordinary user.

→ Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the microoperations to fetch the instruction from main memory, to evaluate the effective address, to execute the operations specified by the instruction, and to return control to fetch phase in order to repeat the cycle for the next instruction.



Microprogrammed control organization.

→ The control memory is assumed to be a ROM, within which all control information is permanently stored. The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.

→ The microinstruction contains a control word that specifies one or more microoperations for the data processor.

→ once these operations are executed, the control must determine next address.

→ The next address may also be a function of external i/p conditions.

→ while the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the ~~micro~~ next microinstruction.

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

→ The address of next microinstruction can be determined many ways

→ 1. Typical functions of a microprogram sequencer are incrementing the control address register by one.

2. loading into the control address register an address from control memory.

3. transferring an external address

4. loading an initial address to start the control operations.

→ The control data register holds the present microinstruction while the next address is computed and read from memory.

→ The data register sometimes called a pipeline register.

→ It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.

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

→ If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

→ The hardware configuration should not be changed for different operations

→ The only thing that must be changed is the microprogram residing in control memory.

Address Sequencing

→ Microinstructions are stored in control memory in groups, with each group specifying a routine.

→ Each computer instruction has its own microprogram routine in control memory to generate the microoperation that execute the instruction

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

→ An initial address is loaded into the control address register when power is turned on in the computer

→ This address is usually the address of the first microinstruction that activates the instruction fetch routine.

→ The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstruction. At the end of the fetch routine, the instruction is in the instruction register of the computer.

→ The control memory must go through the routine that determines the effective address of the operand.

→ A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.

→ The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned

on the status of the mode bits of the instruction.

→ when effective address computation routine is completed, the address of the operand is available in the memory address register.

→ The next step is to generate the microoperations that execute the instruction fetched from memory.

→ The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.

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

→ The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process.

→ A mapping procedure is a rule that transforms the instruction code into a control memory address.

→ once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of microoperations will depend on values of certain status bits processor registers.

→ Microprograms that employ subroutines will require an external register for storing the return address.

→ Return address cannot be stored in ROM because the unit has no writing capability.

→ when the execution of the instruction is completed, control must return

In summary, the address sequencing capabilities required in a control memory are:

1) Incrementing of the control address register.
2) Unconditional branch or conditional branch, depending on status bit conditions.
3) A mapping process from the bits of the instruction to an address for control memory.
4) A facility for subroutine call and return.

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

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

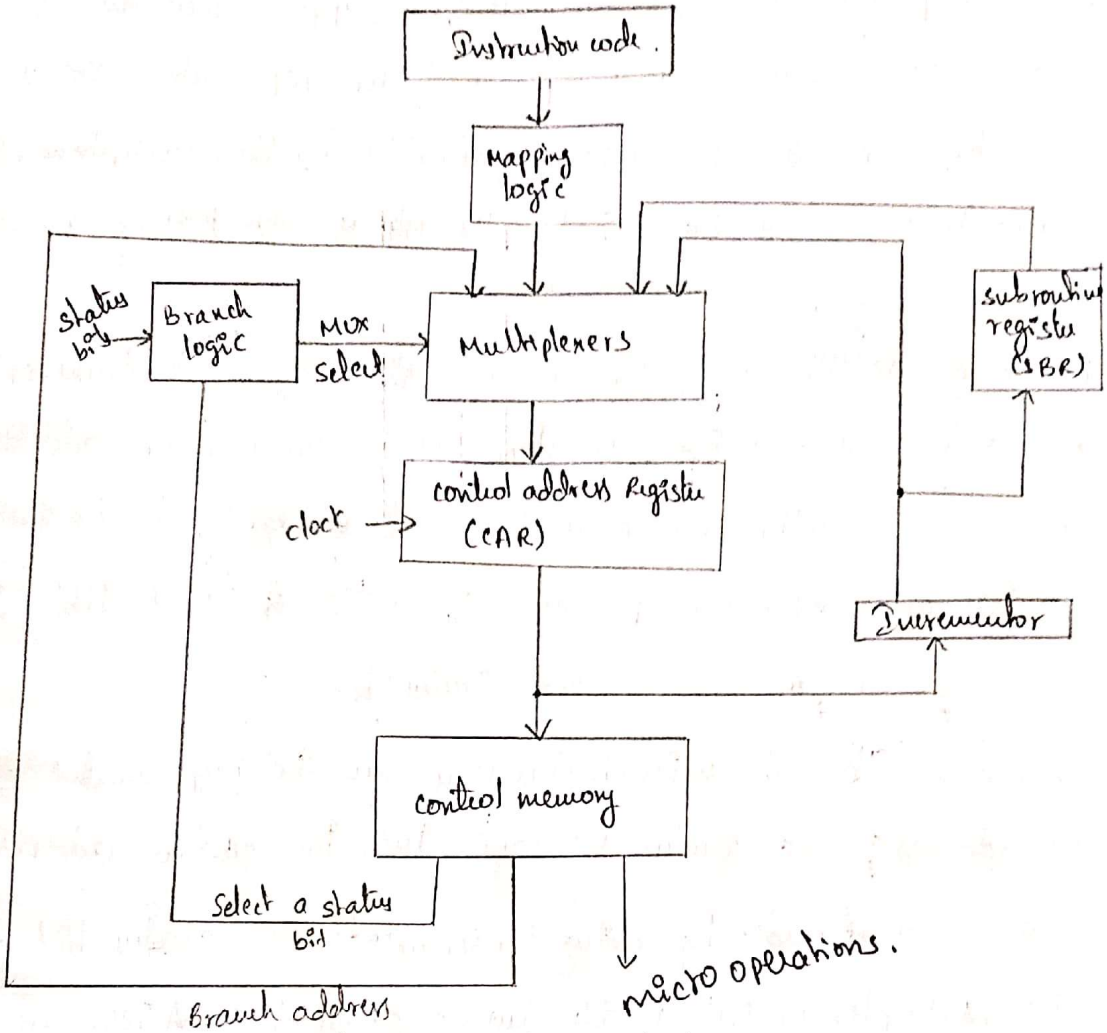
→ The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.

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

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

→ An external address is transferred into control memory via a mapping logic circuit.

→ The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the



Selection of address for control Memory.

Conditional Branching

The branch logic provides decision-making capabilities in the control unit. The status condition 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/P or O/P status conditions.

→ Information in these bits can be tested and actions initated based on their condition: whether their value is 1 or 0.

→ The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

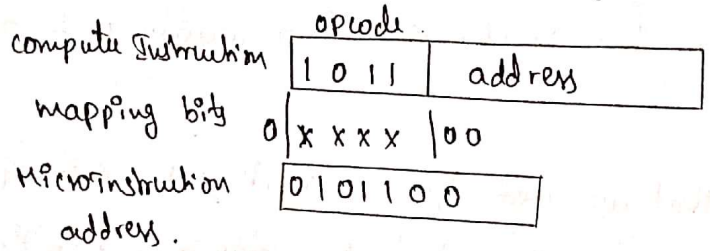
→ The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented.

→ This can be implemented with a multiplexer. Suppose there are 8 status conditions are there then 3 bits are used for representing status condition.
→ These three bits provide the selection variables for the multiplexer. If the selected status bit is in the 1 state, the o/p of the multiplexer is 1; otherwise it is '0'.
→ A '1' o/p in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.
→ A '0' o/p in the multiplexer cause the address register to be incremented.
→ In this configuration, the microprogram follows on of two possible paths, depending on the value of the selected status bit.
→ An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register.
→ This can be accomplished by fixing the value of one status bit at the i/p of the multiplexer, so it is always equal to 1. A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register automatically.

Mapping of Instruction

A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
→ The status bits for this type of branch are the bits in the operation code part of the instruction.
→ for example, Assume that the control memory has 128 words, requiring an address of seven bits. for each operation code there exists a microprogram routine in control memory that executes the instruction.

→ one simple mapping process that converts 4 bit operation code to a 7 bit address for control memory is shown below.



→ This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

→ If the routine needs more than four microinstructions, it can use address 1000000 through 1111111.

→ If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

→ one can extend this concept to a more general mapping rule by using a ROM to specify the ~~address~~ mapping function.

→ In this configuration, the bits of the instruction specify the address of a mapping ROM. The contents of the mapping ROM give the bits for the control address register.

→ In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.

→ The mapping concept provides flexibility for adding instructions for control memory as the need arises.

→ The mapping function is sometimes implemented by means of an integrated circuit called programmable logic device or PLD.

→ A PLD is similar to ROM in concept except that it uses AND and OR gates with internal electronic fuses.

→ The interconnection b/w i/ps, AND gates, OR gates and o/ps can be programmed in ROM. A mapping function that can be expressed in terms of Boolean expressions can be implemented conveniently with a PLD.

Subroutines :-

Subroutines are programs that are used by other routines to accomplish a particular task. A sub routine can be called from any point with in the main body of the microprogram.

→ Frequently, many microprograms contain identical section of code.

→ Microinstructions can be saved by employing subroutine that use common section of microcode.

→ for ex: the sequence of microoperations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions.

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

→ Microprogram that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return.

→ This may be accomplished by placing the incremented output from the control address register into a subroutine register and branching to the beginning of the subroutine.

→ The subroutine register can then become the source for transferring the address for the return to the main routine.

→ The best way to structure a register file that stores addresses for subroutines is to organise the registers in a last in first out (LIFO)

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.

Computer configuration:—

The block diagram of computer consist of two memory units;

→ main memory for storing instructions and data.

→ a control memory for storing the microprogram.

→ four registers are associated with the processor unit (main memory) and two with control unit (control memory).

→ The processor registers are 1) program counter (PC)
2) Address Register (AR)
3) Data Register (DR)
4) Accumulator Register (AC).

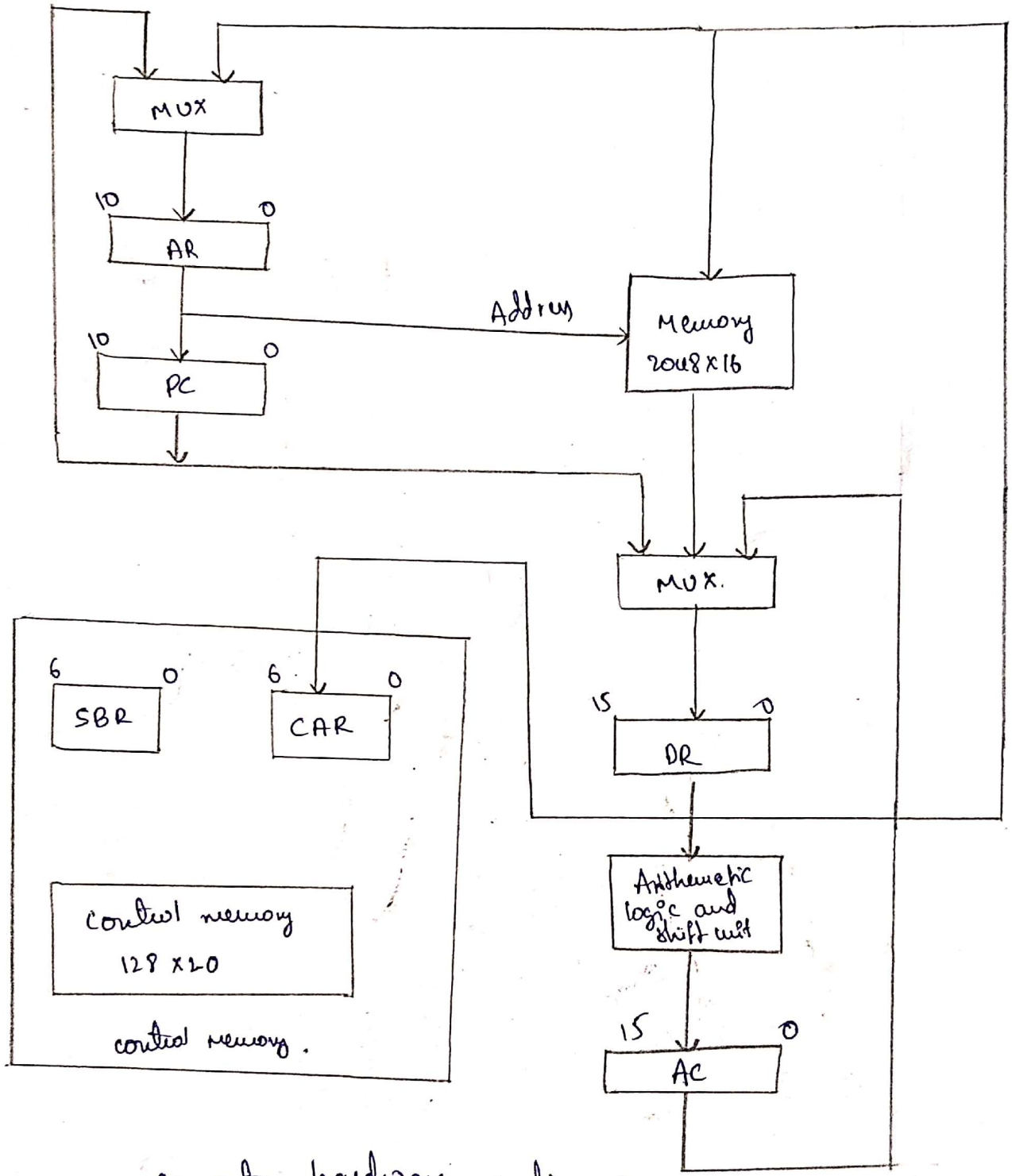
→ The function of these registers is similar to the basic computer

→ The control unit has a ~~CAR and~~ 1) control address Register (CAR)
2) subroutine Register (SBR).

→ The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

→ DR can receive information from AC, PC or memory. AR can receive information from PC or DR.

→ PC can receive information only from AR. (diagram)



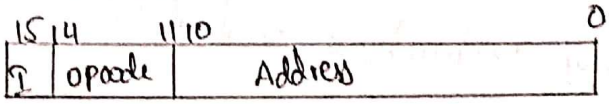
Computer hardware configuration.

→ The arithmetic, logic and shift unit performs microoperation with data from AC and DR. and place the result in AC.

Note: 1. Memory receives its address from AR.
2. Input data written to memory come from DR.
3. data read from memory can go only to DR.

→ The computer instruction format has 3 fields

1- bit field for
4-bit operation code (opcode)
11-bit address field.



(a) Instruction format

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 → effective address.

(b) four computer instructions.

Computer Instructions.

The list of four of the 16 possible memory reference instructions listed above.

→ The ADD instruction adds the content of the operand found in

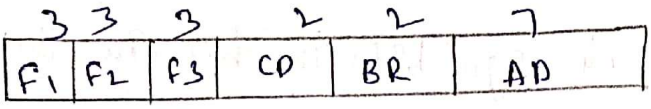
→ The BRANCH instruction causes a branch to the effective address if the operand in AC is negative.
→ The program proceeds with the next consecutive instruction if AC is not negative.
→ The AC is negative if its sign bit (the bit in the left most position of the register) is a 1.
→ The STORE instruction transfers the content of AC into the memory word specified by the effective address.
→ The EXCHANGE instruction swaps the data b/w AC and the memory word specified by the effective address.

Micro Instruction format

The microinstruction format for the control memory is shown below.
The 20 bit micro instruction are divided into four functional parts.
The three fields F_1, F_2, and F_3 specify microoperations for the computer
→ The CD field selects status bit conditions.
→ The BR field specifies the type of branch to be used.
→ The AD field contains a branch address.
→ The address field is seven bit wide, since control memory has $128 = 2^7$ words.
→ The microoperations are subdivided into three fields of three bits each.
→ The three bits in each field are encoded to specify seven distinct microoperations.

→ This gives a total of 21 microoperations, one from each field.

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



f_1, f_2, f_3: Microoperation fields.

CD: condition for branching.

BR: Branch field.

AD: address field.

Ex:

DR ← M[AR] with $f_2 = 100$

and PC ← PC+1 with $f_3 = 101$.

The nine bits of the microoperation fields will then be

000 100 101.

→ The CD field consist of two bits which are encoded to specify four status bit conditions.

CD	condition	Symbol	comments
00	→ always = 1	U	Unconditional Branch
01	→ DR(15)	I	Indirect address bit
10	→ AC(15)	S	Sign bit of AC
11	→ AC = 0	Z	zero value in AC.

→ the (BR) Branch field consist of two bits.

→ It is used, in conjunction with the address field AD, to choose the address of the next microinstruction.