COA

PAGE NO.:

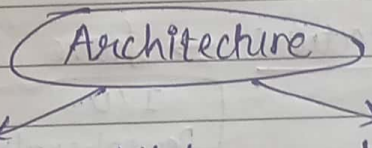
(COMPUTER ORGANIZATION AND ARCHITECTURE)

Computer Architecture
- Deals with functional behaviour of computer System
- Design Implementation for the various parts of computer.

Computer Organization
- Deals with ~~function~~ structural relationship.
- Operational attributes are linked together and contribute to realize the architechural specification

COMPUTER ARCHITECTURE

Defination : It is the design of computer, including their instruction sets, hardware components and system organization



Instruction Set Architechure (ISA)

Hardware System Architechure (HSA)

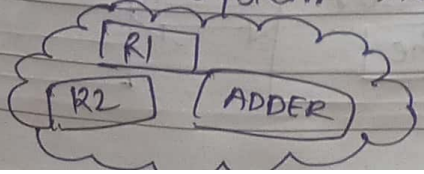
Eg.

Instruction Set Architechure (ISA)

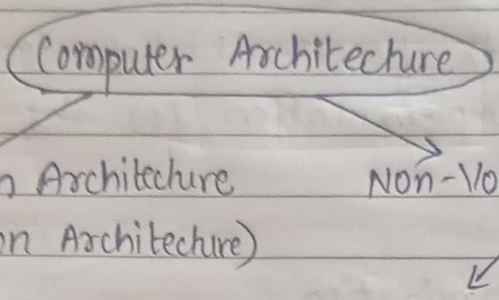
$X = 2 + 3$

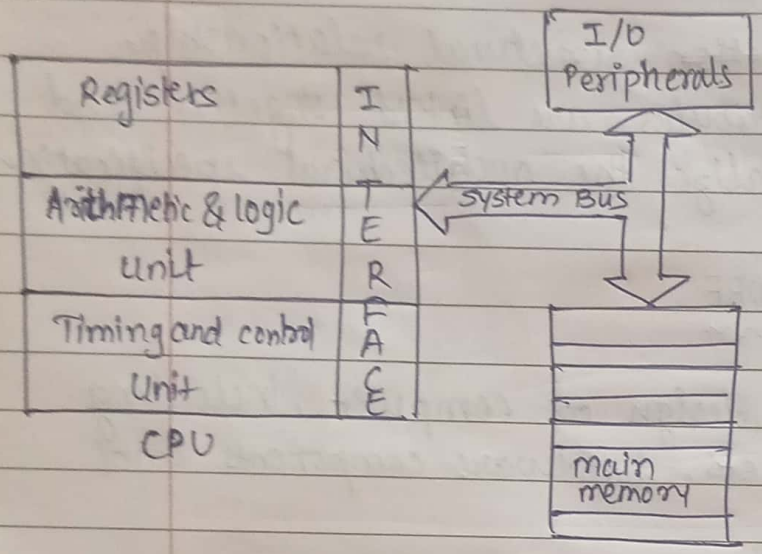
```
MOV R1, 02H
MOV R2, 03H
ADD R1, R2
STORE X, R1
```

Hardware System Architechure (HSA)

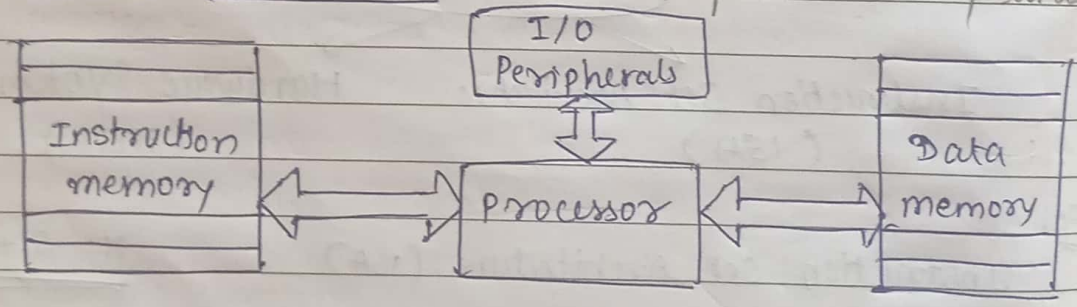


CLASSIFICATION OF COMPUTER ARCHITECTURE

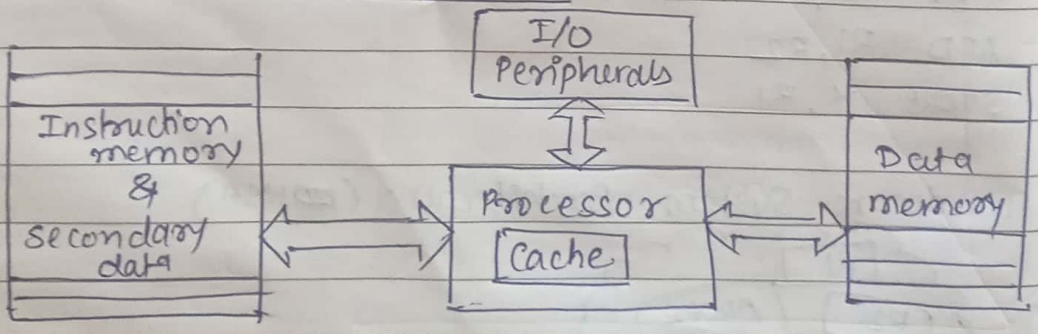




Harvard Architecture → have seperate memory unit



Modified Harvard Architecture



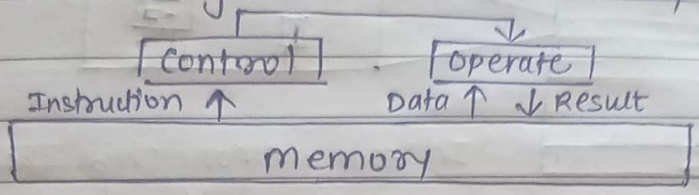
Flynn's Taxonomy

PAGE NO.:

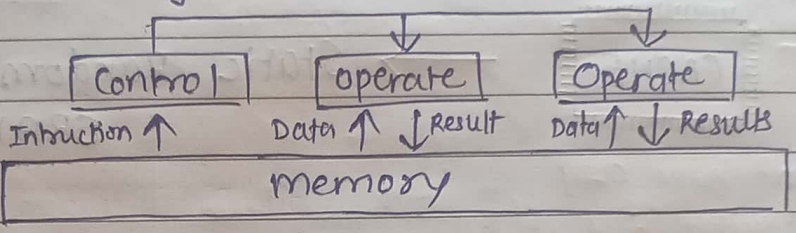
Control → CPU
Operate → ALU

Computer Architecture

→ SISD (Single Instruction stream, Single Data stream)

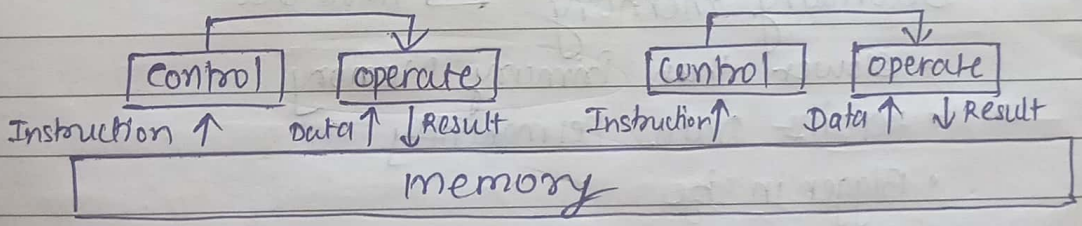


→ SIMD (Single Instruction stream, Multiple Data stream)

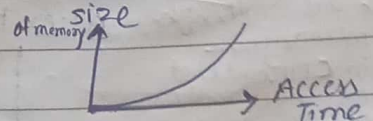


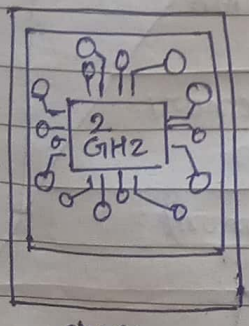
→ MISD (Multiple Instruction stream, single Data stream)

→ MIMD (Multiple Instruction stream, Multiple Data stream)



Memory is the faculty of the brain by which data or information is encoded, stored, and retrieved when needed.





2 GHz processor

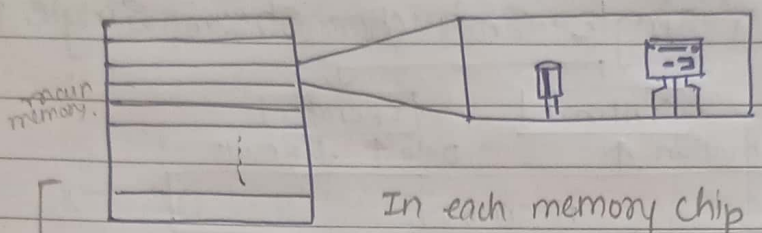
Frequency : 2 GHz

Time : $\frac{1}{\text{frequency}}$

$= \frac{1}{2 \times 10^9}$ sec $= \frac{1}{2} \times 10^{-9}$ sec

$= \frac{1}{2}$ nsec.

Primary memory ⟶ Dynamic Random Access memory
DRAM



In each memory chip their is a transistor with which a capacitor is associated

Both are volatile means they can only retain the data until the power is off.

CACHE ⟶ Static Random Access memory
SRAM

Don't have any capacitor but it is costly than main or primary memory

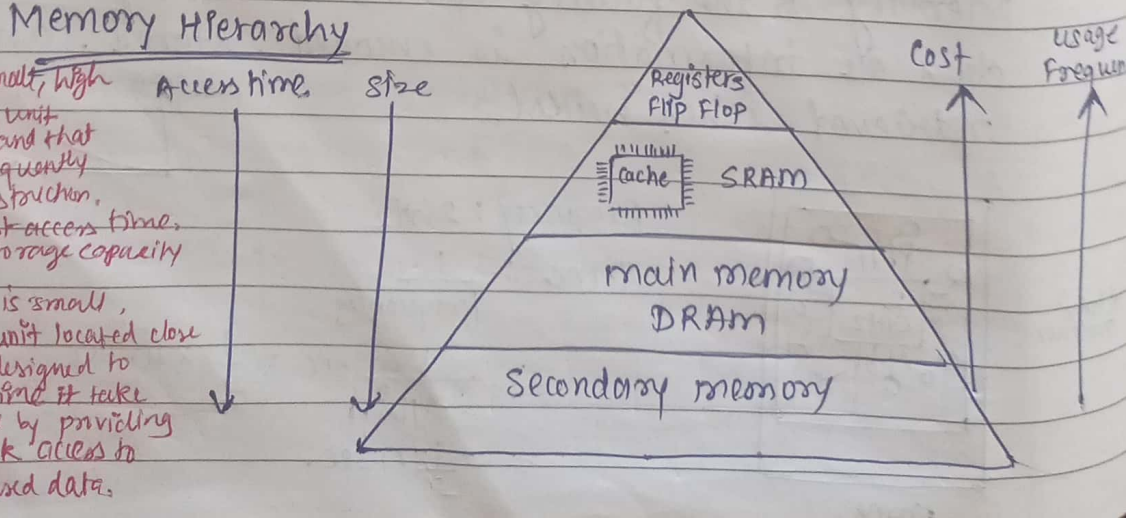
Secondary memory ⟶ Hard Disc.

- Slower than Primary Memory
- Retains Data Permanently
- Bigger in size
- Cost effective
- Semi Random accessibility

Memory Hierarchy

Register are small, high speed memory unit located in CPU and that store more frequently used data and instruction, and have fastest access time, and smallest storage capacity

Cache memory is small, fast memory unit located close to CPU. It is designed to minimize the time it take to access data by providing CPU with quick access to frequently used data.



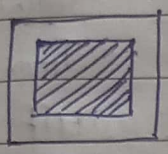
device through I/O processor. It is less expensive and with Auxiliary memory implemented by using Dynamic RAM. and larger in size. It is

secondary memory - They are used ~~to store~~ as backup storage. They ~~are cheaper than main memory~~ and large in size generally in few TB.

PAGE NO.:

Memory Interfacing

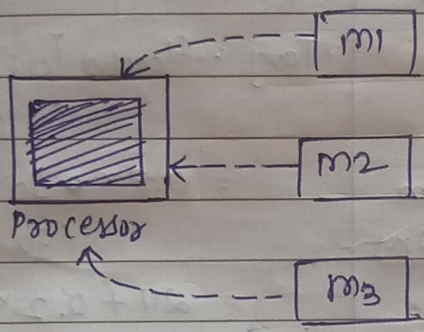
- Part of computer organization
- Deals with the way of connecting various levels of memory units to processor & I/O peripherals.



Processor

unit of measuring speed : MIPS (Million Instruction Per second)

Way 1.



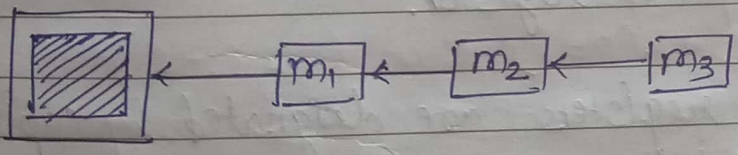
	Access Time	Hit Ratio
M_1	T_1	H_1
M_2	T_2	H_2
M_3	T_3	

$(H_1 \times T_1)$ Time taken to access M_1, if it a hit is M_1

Effective / Average memory Access Time (T_{avg}):

$$H_1T_1 + ((1-H_1) \times H_2)\, T_2 + ((1-H_1) \times (1-H_2))\, T_3$$

Way 2.



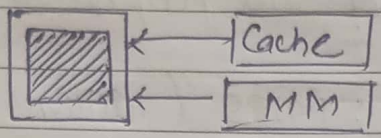
Access Time	T_1	T_2	T_3
Hit Ratio	H_1	H_2	

$$T_{avg} = H_1T_1 + ((1-H_1) \times H_2)(T_1+T_2) + ((1-H_1) \times (1-H_2))(T_1+T_2+T_3)$$

ISRO CS 2017

Q. A cache memory needs an access time of 30ns and main memory 150 ns, what is the average access time of CPU (assume hit ratio = 80%.)

Sol.



Access Time

$T_{cache} = 30ns$

$T_{MM} = 150ns$

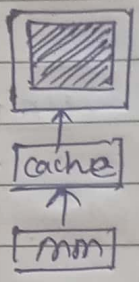
Hit Ratio

$H_{cache} = 80\% = 0.8$

$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}) T_{mm}$

$= 0.8 \times 30 + (1-0.8) \times 150 ns$

$= 24 + 0.2 \times 150 = 54 ns$



Access Time

$T_{cache} = 30ns$

$T_{mm} = 150ns$

Hit Ratio

$H_{cache} = 80\% =$

$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}$

$(T_{cache} + T_m$

$= 0.8 \times 30 + (1-0.8)$

$(30 + 150) ns$

$= 24 + 0.2 \times 180$

$= 60 ns$

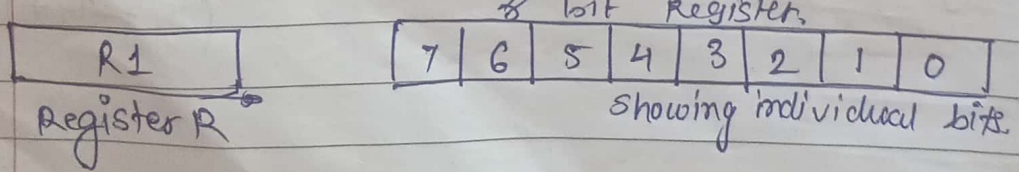
Register Transfer Language

→ Register - collection of individual flip flops connected in numbers to form different types of register.

→ Computer registers are designated by capital letters.

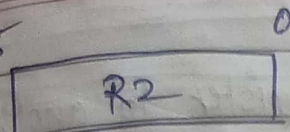
→ For example,

- MAR → Memory Address Register
- PC → Program Counter
- IR → Instruction Register
- R1 → Processor Register.

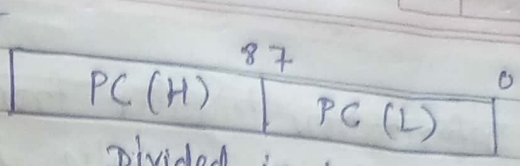


16 bit.

L → Lower
H → higher



Numbering of bits



Divided in two parts

Microoperations

- The operations executed on data stored in registers are called microoperations.
- A microoperation is an elementary operation performed on the information stored in one or many registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- Eg: shift, count, clear and load.

Register Transfer Language

→ The symbolic notation used to describe the microoperation transfer among registers is called register transfer language.

→ The term "register transfer" implies the availability of hardware logic circuit that can perform a stated microoperation and transfer the result of the operation to the same or another register.

→ A register transfer language is a system for expressing in symbolic form the microoperation sequence among the registers of a digital module.

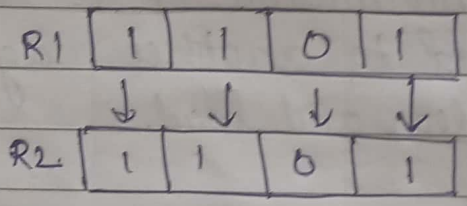
→ Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.

→ The statements

$$R2 \leftarrow R1$$

denotes a transfer of the content of register R1 into register R2.

previous content of R2 will be overrided by the new content of R1 but that ~~doesn't~~ doesn't mean that conent of R1 will be deleted



Register Transfer with Control function.

→ Normally, we want the transfer to occur only under a predetermined control condition using if-then statement,

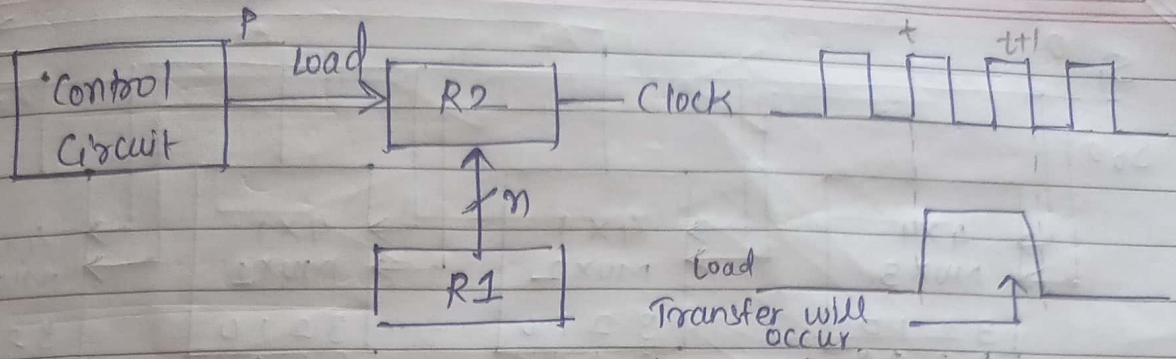
$$if(P=1)\ then\ (R2 \leftarrow R1)$$

where P is a control signal generated in the control section.

→ ~~When~~ A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

$$P: R2 \leftarrow R1$$

Note: Transfer will occur only when the signal goes low.



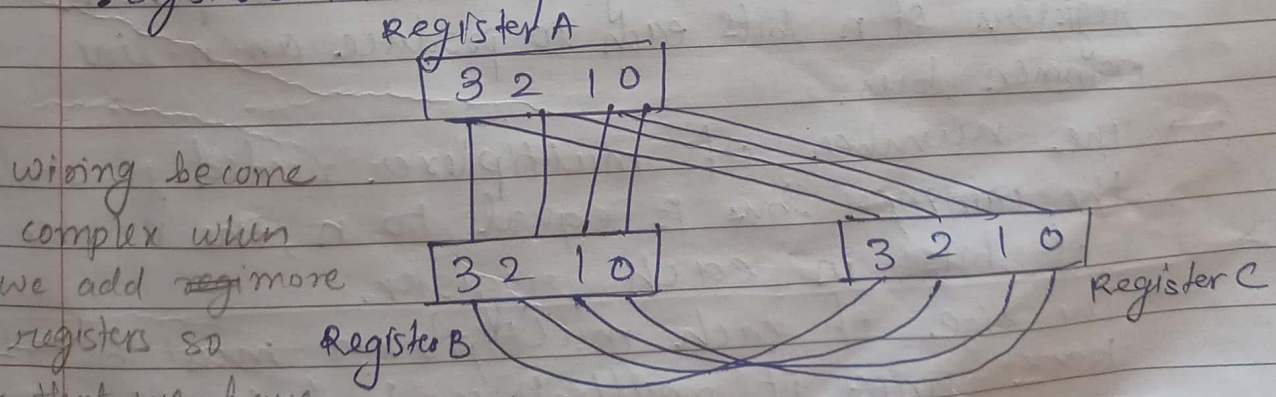
When control circuit will send signal P=1. And this 1 is connected to the load i/p of register R2 and that means when load is would be enable for R2 then content of R1 will be transfered to R2.

Bus and Memory Transfer

Common bus system for 4 registers

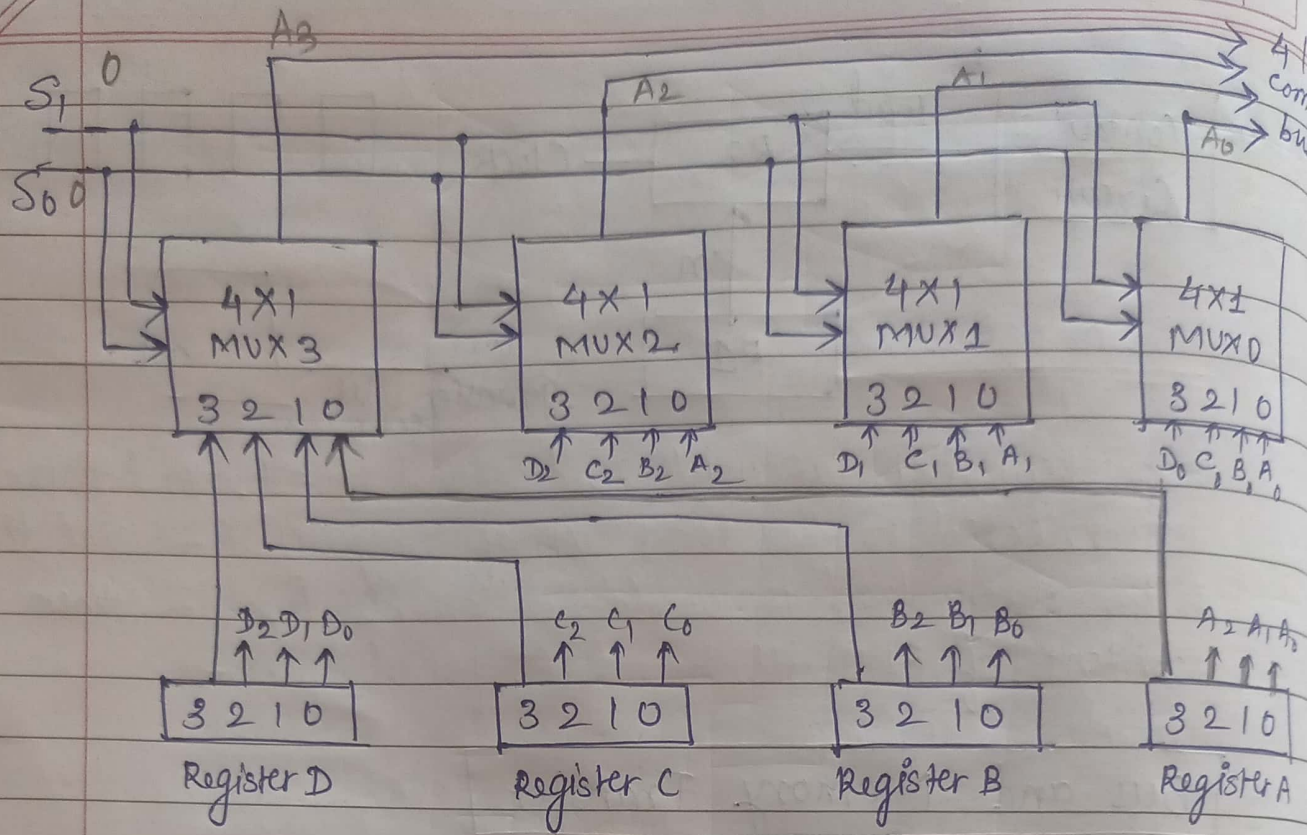
→ A typical digital computer has many registers and paths must be provided to transfer information from one register to another.

→ The number of wires will be exessive if seperate line are used between each register and all other registers in the system.



wiring become complex when we add more registers so. for that we have a solution common bus system

→ Common Bus system is more efficient scheme for transferring information in a system with many registers.



S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

→ In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus.

→ The number of multiplexer needed to construct the ~~common~~ bus is equal to n, the number of bits in each register.

→ The size of each multiplexer must be k×1 since it multiplexes k data lines

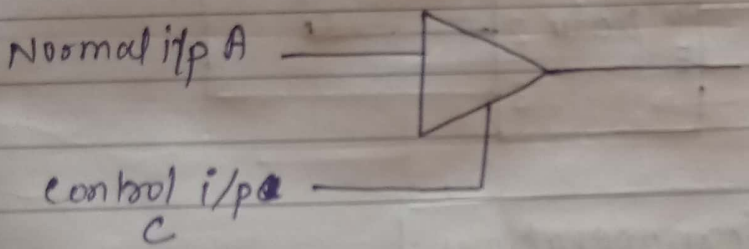
→ For eg. a common bus of 8 register of 16 bits requires

Multiplexers – 16 of (8×1)

Select lines – 3

Tri-State Buffer

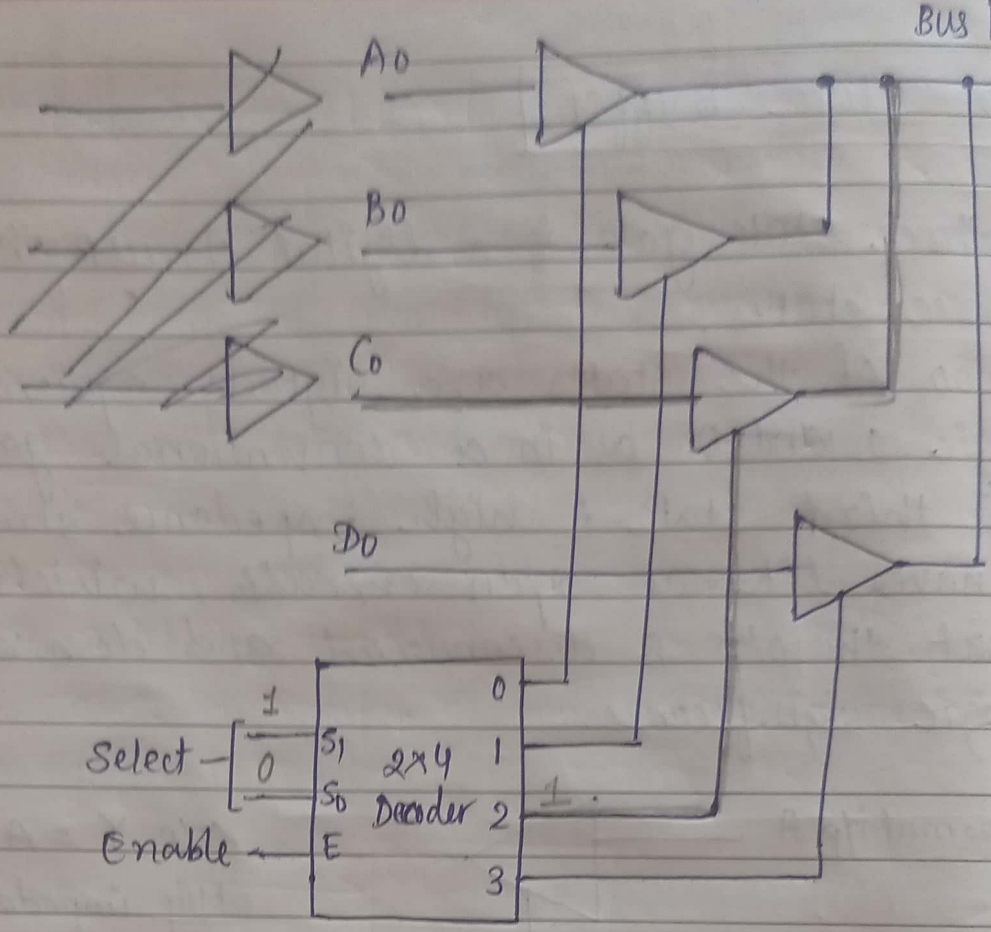
→ A three state gate is a digital circuit that exhibits three states.
→ Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
→ The third state is high impedance state which behaves like an open circuit, which means that the o/p is disconnected and does not have logic significance.



→ The control i/p determines the o/p state. When the control i/p C is equal to 1, the o/p is enabled and the gate behaves like any conventional buffer, with the o/p equal to the normal i/p.
→ When the control i/p C is 0, the o/p is disabled and the gate goes to a high impedance state, regardless of the value in the normal i/p.

Application

Common bus system using decoder and tri state buffer.



Arithmetic Microoperations

→ Arithmetic microoperations perform arithmetic operations on numeric data stored in registers

Add Microoperation
$R3 \leftarrow R1 + R2$

Subtract Microoperation
$R3 \leftarrow R1 - R2$
$R3 \leftarrow R1 + \overline{R2} + 1$

Symbolic Designation	Description
$R3 \leftarrow R1 + R2$	Content of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Content of R1 minus R2 transferred to R3

$R2 \leftarrow \overline{R2}$	Complement to contents of R2 (1's complement)
$R2 \leftarrow \overline{R2}+1$	2's complement of the content of R2 (negate)
$R3 \leftarrow R1+\overline{R2}+1$	R1 plus the 1's complement of R2 (subtraction)
$R1 \leftarrow R1+1$	Increment the content of R1 by 1
$R1 \leftarrow R1-1$	Decrement the content of R1 by one

Memory Transfer

→ Read Operation: The transfer of information from a memory word to the outside environment is called a read operation.

→ Write Operation: The transfer of new information to be stored into the memory is called a write operation.

→ A memory word will be symbolized by the letter M.

→ It is necessary to specify the address of M when writing memory transfer operation. M[Address]

→ This will be done by enclosing the address in square brackets following the letter M.

→ Consider a memory unit that receives the address from a register, called the address register, symbolized by AR

→ The data are transfered to another register, called the data register, symbolized by DR. The read operation can be stated as follows:

Read : DR ← M[AR]

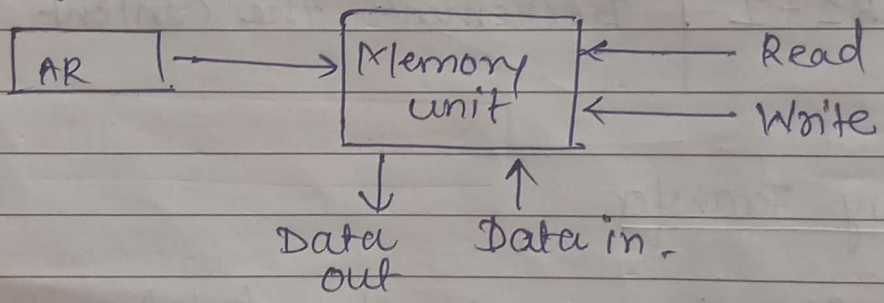
→ This causes transfer of information into DR from the memory word M selected by the address in AR.

→ The write operation transfer the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.

→ Write operation can be stated symbolically as follows:

Write : M[AR] ← R1

→ This causes a transfer of information from R1 into memory word M selected by address AR.



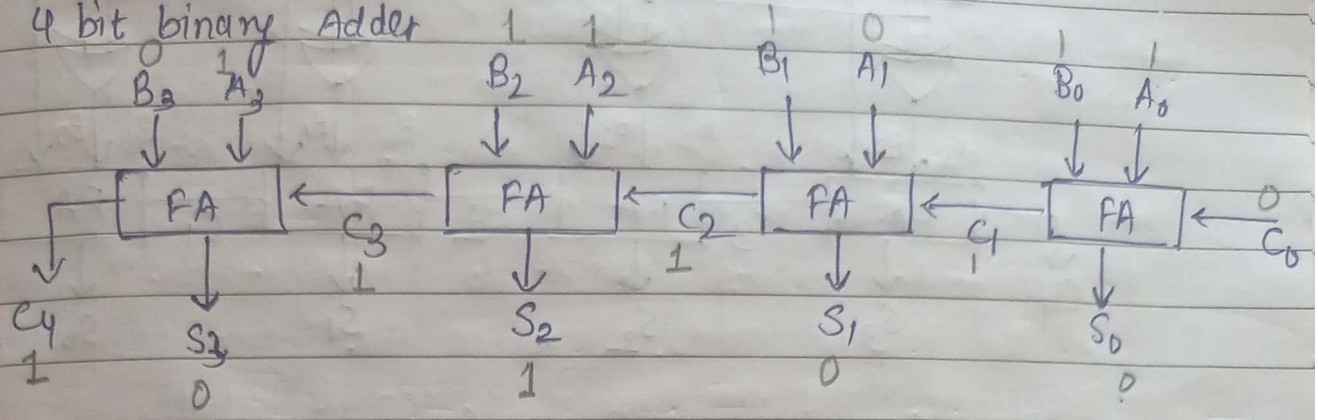
Binary Adder

→ The digital circuit that generates the arithmetic sum of two binary numbers of any length is called binary adder.

→ Eg.

	C	1	1	1	0
	R1	1	1	0	1
	R2	+ 0	1	1	1
	sum	1	0	1	0	0

4 bit binary Adder



→ The binary adder is constructed with full adder circuit connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder.

Binary Adder Subtractor

→ The subtraction of binary number can be done most conveniently by means of complements

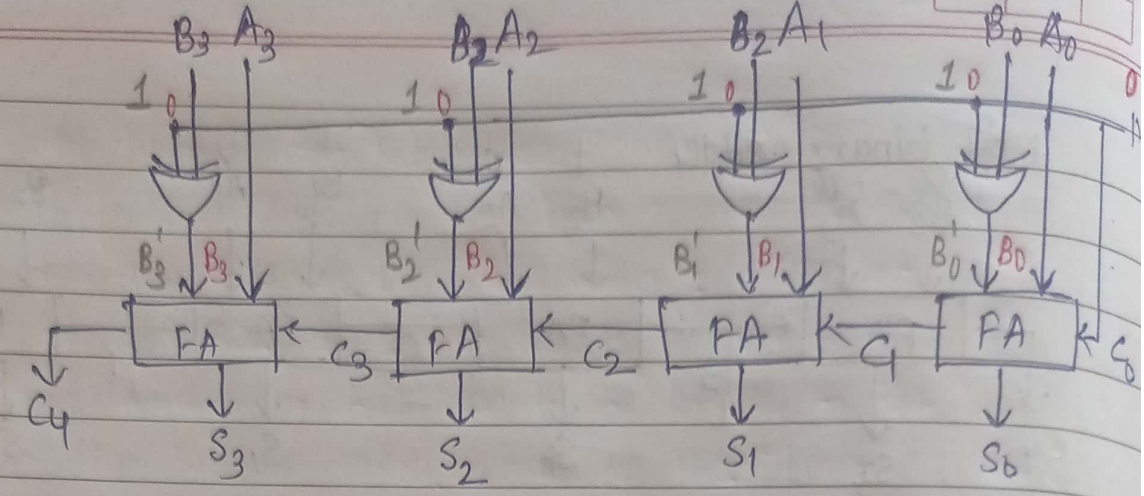
→ Remember that the subtractor A−B can be done by taking the 2's complement of B and adding it to A.

→ The 2's complement can be obtained ~~and ad~~ by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverter and a one can be added to the sum through the input carry.

→ The addition and subtraction operation can be combined into common circuit by including an exclusive OR gate with each full adder.

→ The mode input M controls the operation.
When M=0 the circuit is an Adder
M=1 the circuit becomes a Subtractor

M = 0 : A + B (Addition)
M = 1 : A + $\bar{B}$ + 1 (Subtraction)



For unsigned number,

If A >= B , then A - B

If A < B , then B - A

For signed numbers,

Result is A - B , provided that there is no overf

4 bit Binary Incrementar

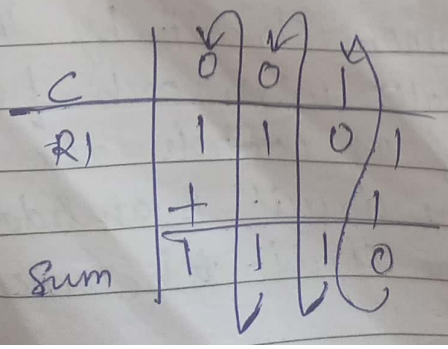
→ The increment micro operation adds one to a number in a register.

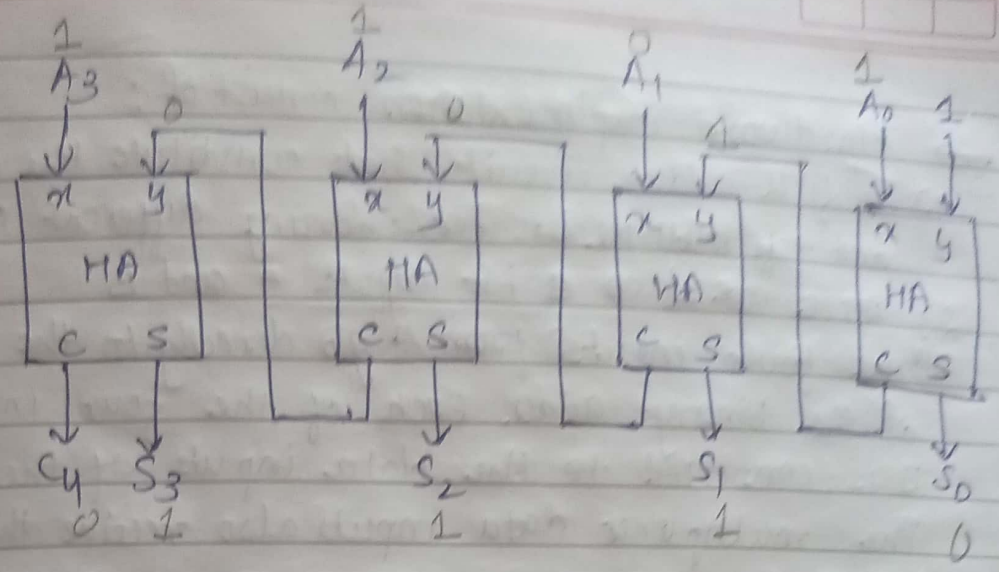
→ for eg, if a 4 bit register has a binary value 0110, it will go to 0111 after it is incremented.

$$\begin{array}{r} 0\ 1\ 1\ 0 \\ +\quad\ \ 1 \\ \hline 0\ 1\ 1\ 1 \end{array}$$

→ ~~Fffe diagr~~

Eg.





→ One of the input to the least significant half adder (HA) is connected to the logic-1 and the other input is connected to the least significant bit of the number to the incremented.

→ The output carry from one half adder is connected to one of the inputs of the next higher order half adder.

→ The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3.

→ ~~The output carry~~ C_4 will be 1

4 Bit Arithmetic Circuit

→ The arithmetic micro-operations can be implemented in one composite arithmetic circuit.

→ The basic component of an arithmetic circuit is the parallel adder.

→ By controlling the data inputs to the adder, it is possible to obtain different type of arithmetic operations.

→ The output of binary adder is calculated from arithmetic sum

$$D = A + Y + C_{in}$$

A → fixed input
Y → multiplexed i/p.

→ Hardware implementation consist of:

1.) 4 full adder circuits that constitute the 4 bit adder and 4 multiplexers for choosing different operations

2.) There are two 4 bit inputs A and B
The four inputs from A go directly to the x inputs of the binary adder. Each of the four inputs from B is connected to the data inputs of the multiplexer. The multiplexer data inputs also receive the complement of B.

3.) The other two data inputs are connected to logic 0 and logic -1. Logic 0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0.

4.) The four multiplexers are ~~connected~~ controlled by two selection inputs, S_1 and S_0

5.) The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.

6.) 4 bit output $D_0 \dots D_3$

Decrement using 2's complement

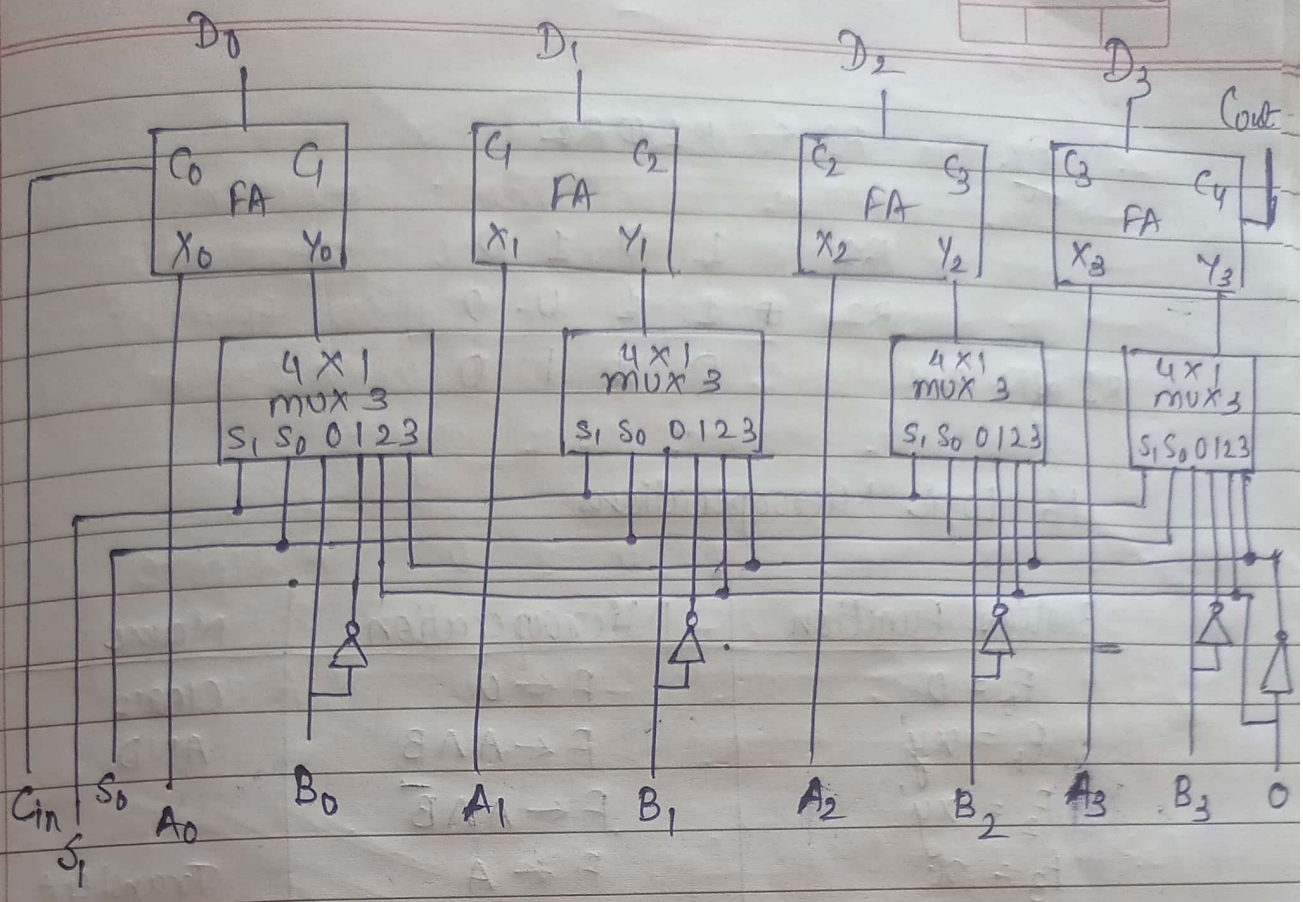
$$\begin{array}{r} 1101 \\ - \quad 1 \\ \hline 1100 \end{array} \xrightarrow{\text{2's complement}}$$

$$\begin{array}{r} 1110 \\ 1101 \\ +\,1111 \\ \hline 1\,1100 \end{array}$$

Discard carry

$D = A + Y + C_{in}$

Decrement = $A + 1111 + 0$



S_1	S_0	Cin	Y	$D = A + Y + Cin$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B'	$D = A + B'$	Subtract with borrow
0	1	1	B'	$D = A + B' + 1$	Subtract
1	0	0	0	$D = A$	Transfer
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Logic Microoperations

→ Logic micro operations specify binary operations for strings of bits stored in register.

→ These operations consider each bit of the register separately and treat them as binary variable.

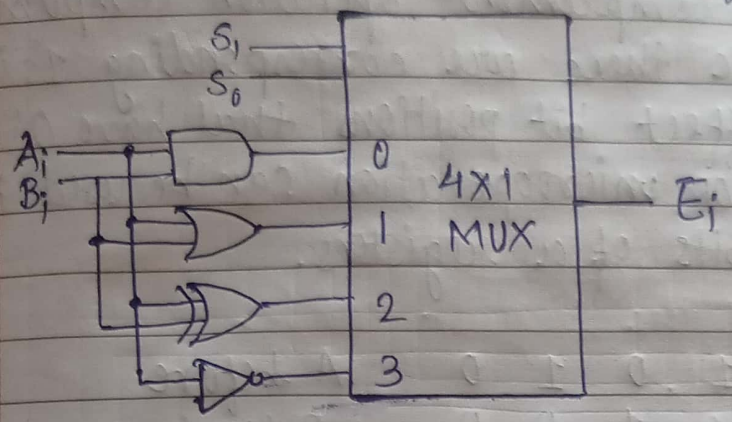
Eg. $R3 \leftarrow R1 \oplus R2$

R1	1	0	1	0
R2 ⊕	1	1	0	0
R1 after P=1	0	1	1	0

16 Logic Microoperations

Boolean Function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive - NOR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware implementation of logic circuit



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

Application of logic Microoperations

1. Selective set operations
   - The selective-set operation set to 1 the bits in register A where there are corresponding 1's in register B
   - It does not affect bit positions that have 0's in B
   - The OR microoperation can be used to selectively set bits of a register

Where ever their are 1 in B register corresponding bit in A register will become 1

```
  1 0 1 0    A before
  1 1 0 0    B (logic operand)
  ---------
  1 1 1 0    A after.
```

(2.) Selective complement Operation

→ The selective - complement operation complements bits in A where there are corresponding 1's in B.
→ It does not affect bit positions that have 0's in B.
→ The exclusive OR microoperation can be used to selectively set bits of a registers.

Where ever their are one in B the corresponding bit of A will be complemented

```
1 0 1 0   A before
1 1 0 0   B (logic operand)
---------
0 1 1 0   A after
```

(3.) Selective clear Operation

→ The selective clear operation clear to 0 the bits in A only where there are corresponding 1's in B.
→ It does not affect bit positions that have 0's in B.
→ The corresponding logic microoperation is $A \leftarrow A \wedge B'$

Wherever their are 1 in B the corresponding bit of A will be 0

```
1 0 1 0   A before
1 1 0 0   B (logic operand)
---------
0 0 1 0   A after.
```

(4.) Mask operation

→ The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are ~~com~~ corresponding 0's in B.
→ The mask operation is ~~And~~ an AND microoperation

where their are 0 in B corresponding bit of A will become 0

```
1 0 1 0   A before
1 1 0 0   B (logic operand)
---------
1 0 0 0   A after
```

(5.) Insert Operation

→ The insert operation insert a new value into a group of bits

→ This is done by first masking and then ORing them will required value.

→ The mask operation is an AND microoperation and the insert operation is an OR microoperation

→ Consider an example of inserting # 1001 in place of 0110.

wherever we want to insert new bits their we need to perform masking. So, for masking we would be putting zero where we want to perform replace. So rest would be one.

Mask

```
A   0110   1010
B   0000   1111
    ----------
A   0000   1010
```

Whereever their are zero in B corresponding A will become zero.

Insert

```
A   0000   1010
B   1001   0000
    ----------
A   1001   1010
```

We have to perform ORing So, o/p of stage one is to be OR with the group of bit you want to replace and rest bit is to kept 0.

(6.) Clear Operation

→ The clear operation compares the words in A and B and produces an all 0's result. if the two number are equal.

→ This operation is achieved by an exclusive-OR microoperation

```
1 0 1 0   A
1 0 1 0   B
---------
0 0 0 0   A ← A ⊕ B
```

If both bits are same then it becomes 0.

Shift Microoperation

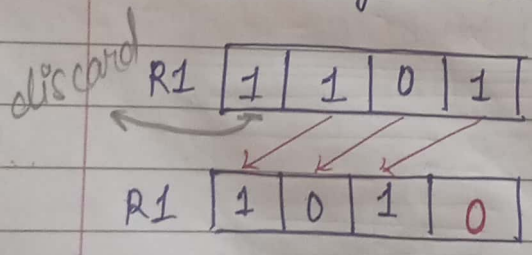
→ Shift microoperations are used for serial transfer of data.
→ Used in conjuction with arithmetic, logic and other data processing operations
→ The content of the register can be shifted to the left or the right.
→ The first flip-flop receives its binary information from the serial input.
→ The information transferred through the serial input determines the type of shift.

Types of shift

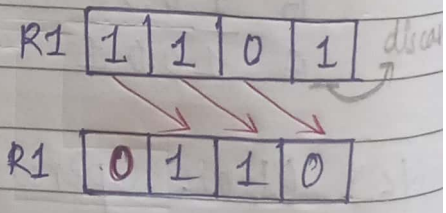
(1.) Logical shift

→ A logical shift is one that transfers 0 through the serial input

shl - logical shift left



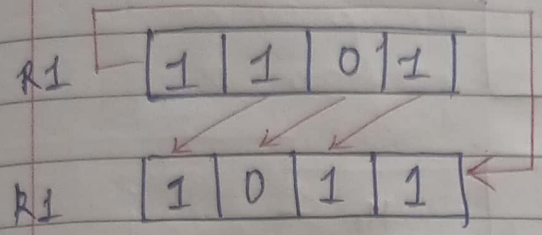
shr - logical shift right



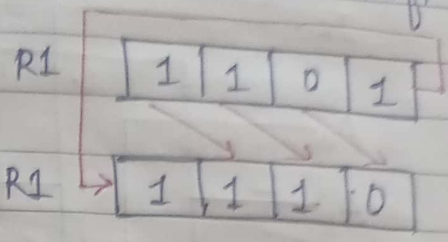
(2) Circular shift

→ A circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information
→ This is accomplished by connecting the serial o/p of the shift register to its serial input.

cil - circular shift left



cir - circular shift right



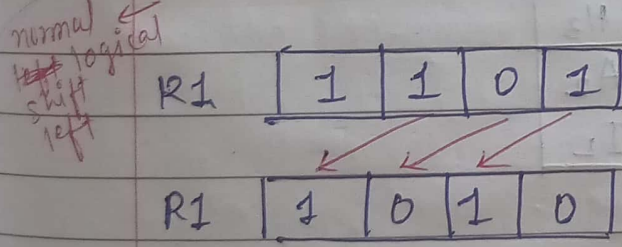
(3.) Arithmetic shift

→ A arithmetic shift is a micro-operation that shifts a signed binary number to the left or right

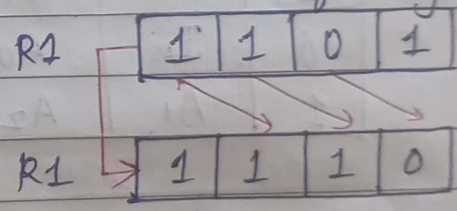
→ An arithmetic shift left multiplies a signed binary number by 2.

→ An arithmetic shift - right - divides the number by 2

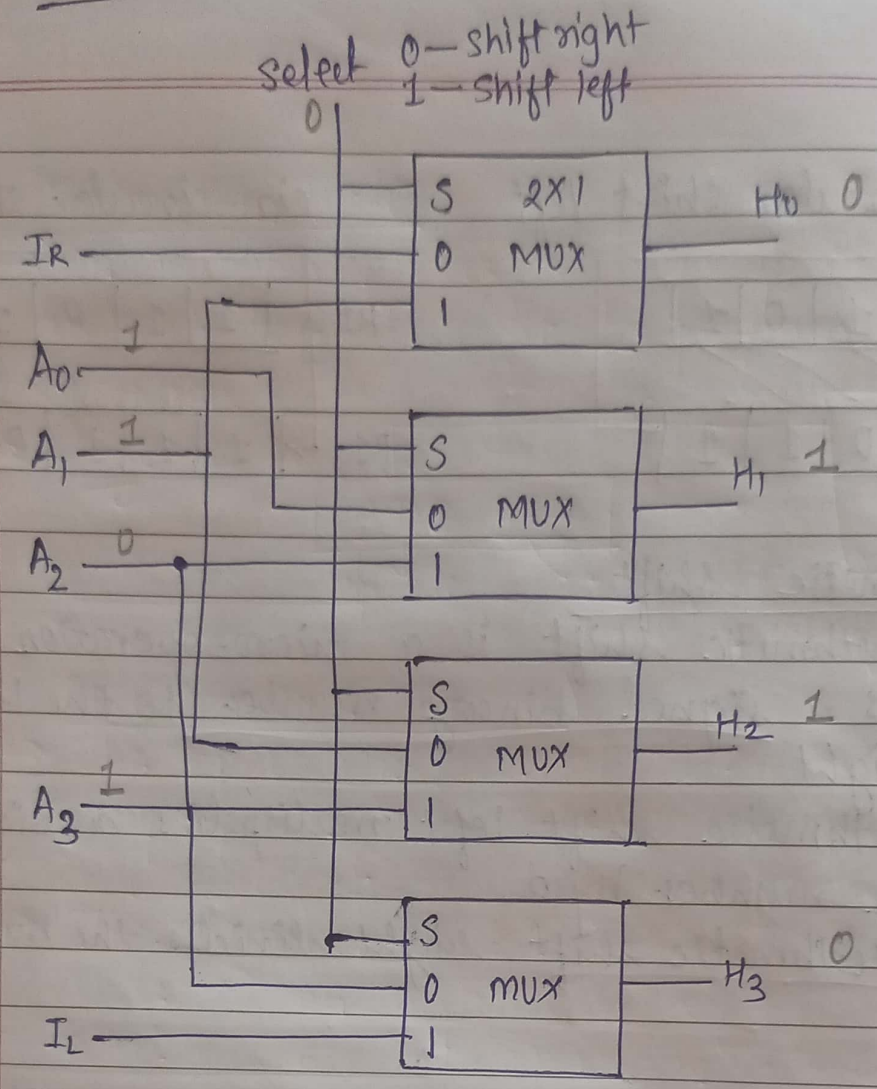
ashl - arithmetic shift left



ashr - arithmetic shift right.

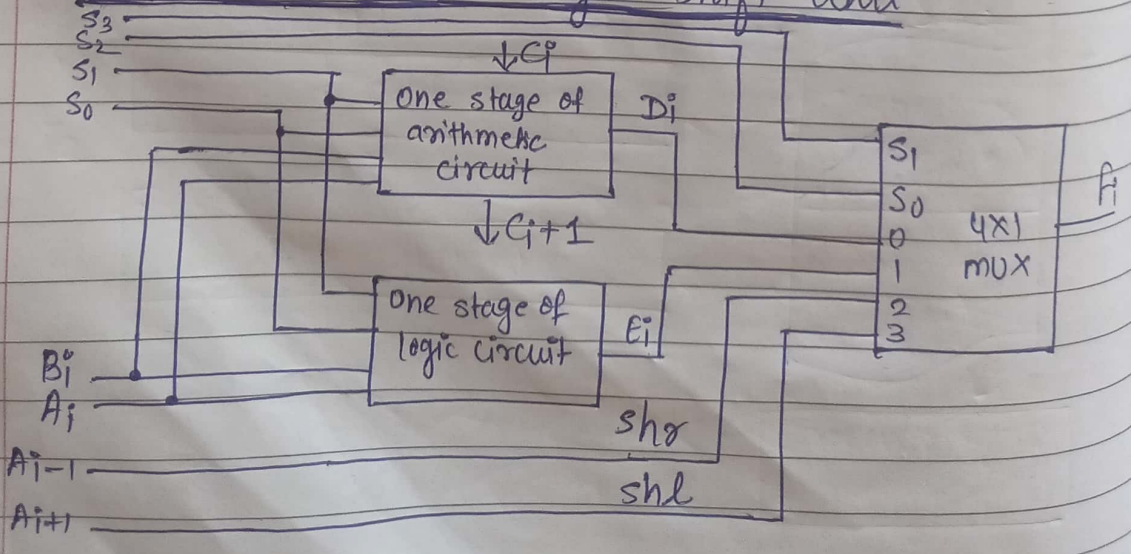


4 bit combinational circuit shifter



S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

4 bit Arithmetic Logic Shift Unit



S_3	S_2	S_1	S_0	Cin	Operation	Function
0	0	1	1	1	F = A	Transfer A
0	1	0	0	x	F = A ∧ B	AND
0	1	0	1	x	F = A ∨ B	OR
0	1	1	0	x	F = A ⊕ B	XOR
0	1	1	1	x	F = A'	Complement A
1	0	x	x	x	F = shr A	shift right A into F
1	1	x	x	x	F = shl A	Shift left A into F

MODULE - 2

Instruction Code

- **Program**

→ A program is a set of instructions that specify the operations, operands and the sequence by which processing has to occur.

- **Computer Instruction**

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

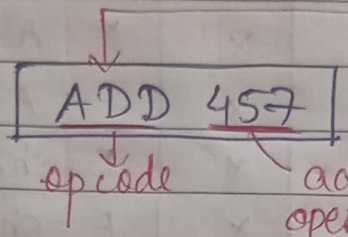
→ The computer reads each instruction from memory and place it in a control register.

→ The control ~~register~~ then interepts the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.

Instruction Code

→ An instruction code is a group of bits that instruct the computer to perform a specific operation.

→ Eg.



Operation code (Opcode)

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

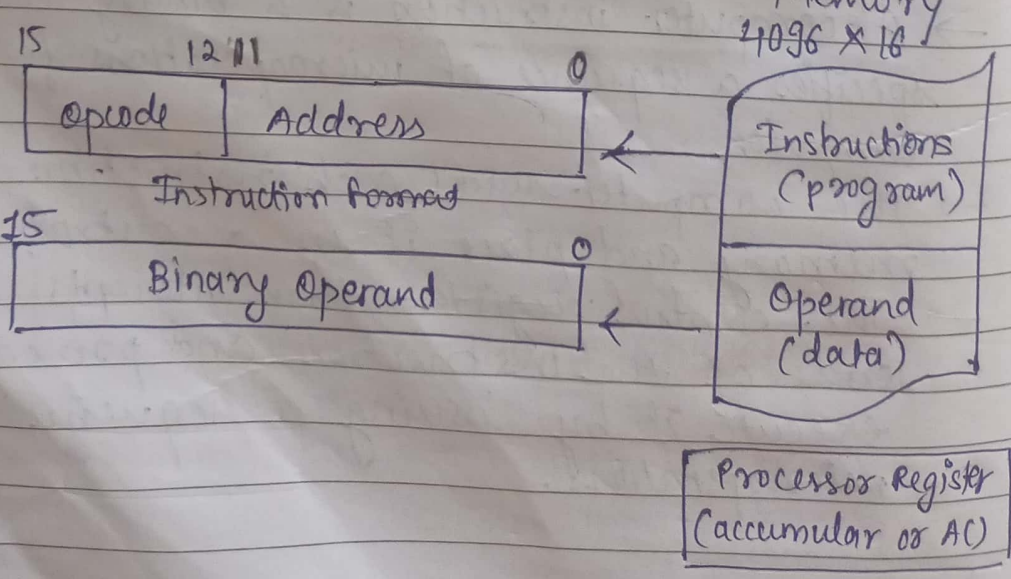
→ The number of bits required for the operation code of an instruction depends on the total number of operation available in the computer.

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

Stored Program Organization

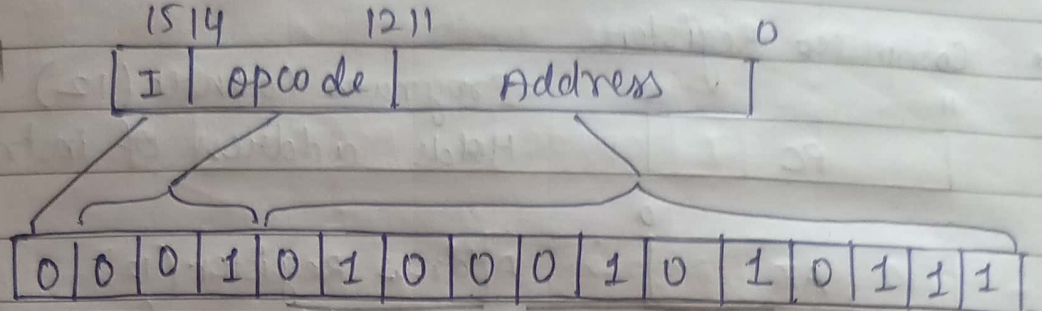
4096 → 2^{12}

So too point each memory location we require 12 bit



Instruction format of basic computer

I bit stand for direct and indirect



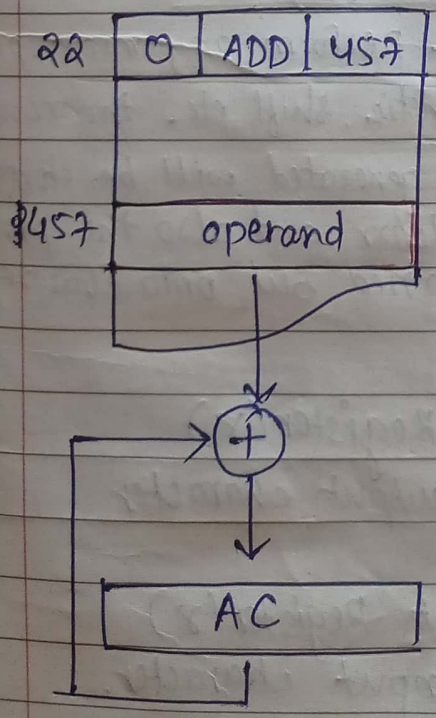
Add Instruction - ADD 457

Direct and Indirect Addressing of memory

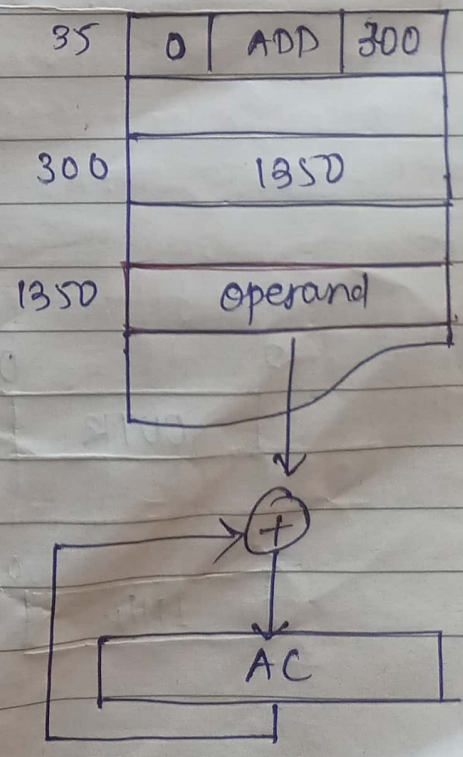
→ If the second part of an instruction format specifies the address of an operand, the instruction is said to have a direct address.

→ In Indirect address, the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found

Direct Addressing



Indirect addressing

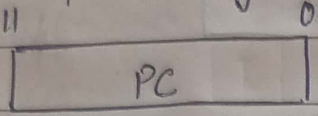


depend on any of the signal

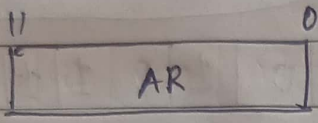
PAGE NO.:

Common Bus System

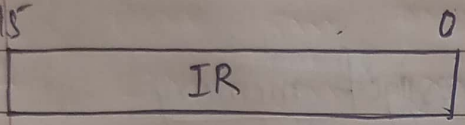
- Computer Registers



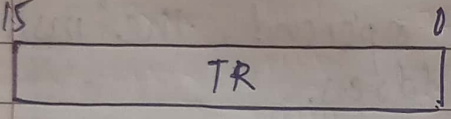
Program Counter (12)
Holds address of instruction



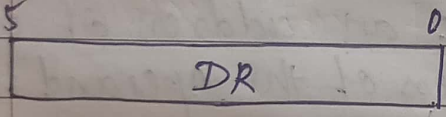
Address Register (12)
Holds address of memory



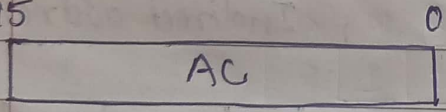
Instruction Register (16)
Holds instruction code



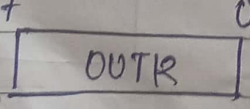
Temporary Register (16)
Holds temporary data



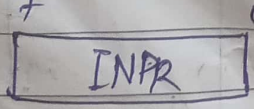
Data Register (16)
Holds memory operand



Accumulator (16) or Processor Register
when we perform any microoperation like arithmetic, shift etc. the result which is getting generated will be stored into the accumulator and also the processing will be carried out onto this register only.



Output Register (8)
Holds output character

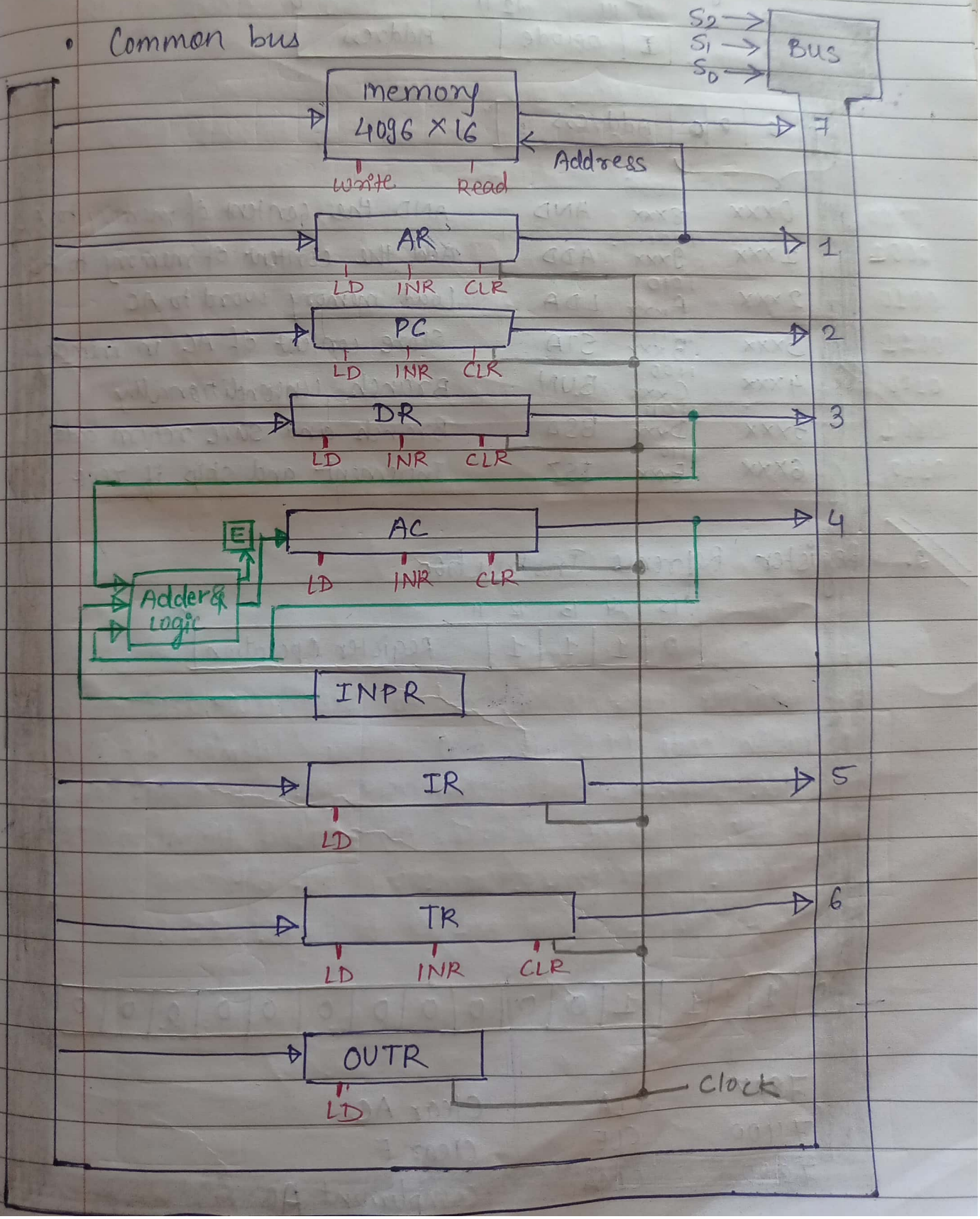


Input Register (8)
Holds input character.

Load (LD) → It will allow data to store into the register
Increment (INR) → it will increment the content of corresponding register means if the signal is high on INR the content of any of these register would be incremented by 1.
Clear (CLR) → clear signal will clear the data of the particular signal. that means it will make all the bits to be zero. so it will reset the register.

Memory
4096 words
16 bits per word

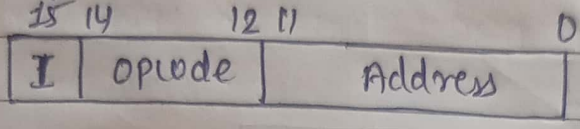
- Common bus

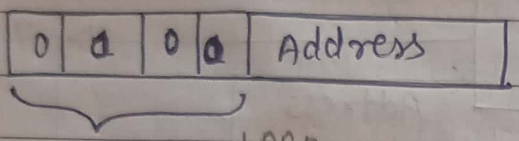


Memory Reference Instruction uses 12 bits to specify an address and one bit to specify the addressing mode. I. I is equal to 0 for direct address and to 1 for indirect address.

Types of Computer Instruction

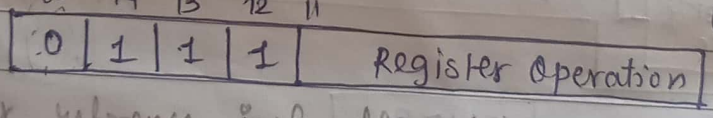
1. Memory Refrence Instruction



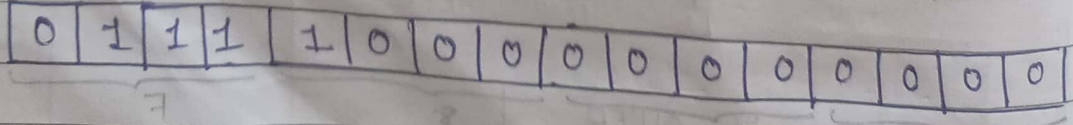


0000	0xxx	8xxx	AND	AND the content of memory to AC
0001	1xxx	9xxx	ADD	Add the content of memory to AC
0010	2xxx	Axxx	LDA	Load memory word to AC
0011	3xxx	Bxxx	STA	Store content of AC in memory
0100	4xxx	Cxxx	BUN	Branch unconditionally
0101	5xxx	Dxxx	BSA	Branch and save return address
0110	6xxx	Exxx	ISZ	Increment and skip if zero

2. Register Refrence Instruction



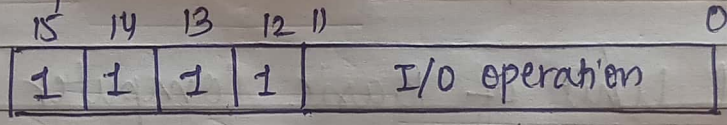
The register refrence instruction are recognized by the operation code 111 with a 0 in the leftmost bit. A register-refrence instruction specifies an operation on or a test of the AC register. An operand from memory is not needed therefore, the other 12 bits are used to specify the operation or test to be executed.



7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC

7100	CME	Complement E
7080	CIR	Circulate right AC and E
7040	CIL	Circulate left AC and E
7020	INC	Increment AC
7010	SPA	Skip next instruction if AC is positive
7008	SNA	Skip next instruction if AC is negative
7004	SZA	Skip next instruction if AC is zero
7002	SZE	Skip next instruction if E is zero
7001	HLT	Halt computer

3. Input - Output Instruction



An input output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bit are used to specify the type of input - output operation or test performed.

1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0

F800	INP	Input character to AC
F400	OUT	Output character from AC
F200	SKI	skip on input flag
F100	SKO	skip on output flag
F080	ION	Interrupt on
F040	IOF	Interrupt off

- Instruction set completeness

→ Instruction set is said to be complete if it includes sufficient number of instruction in each of the following categories:

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

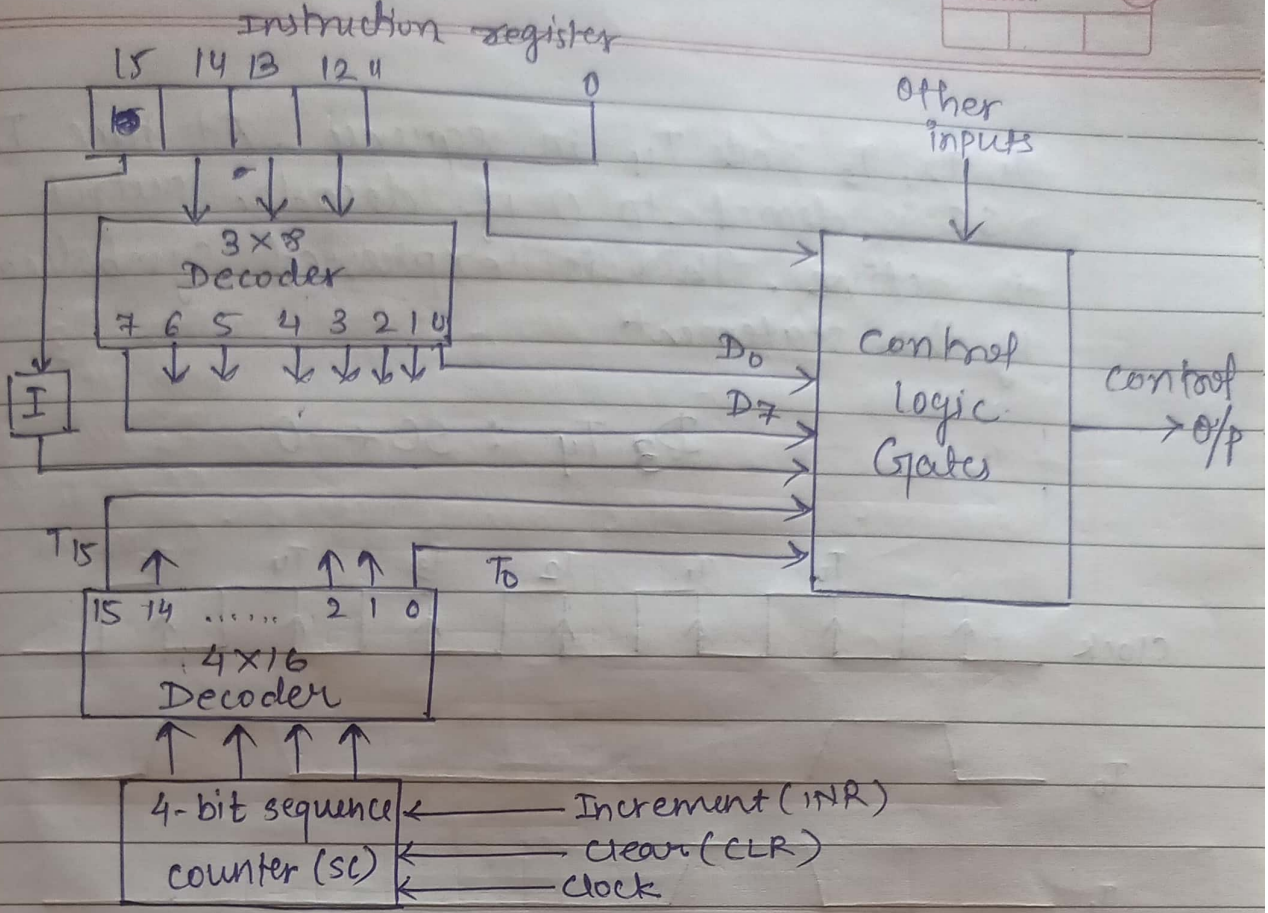
Control Unit of Basic Computer

- Control organization with timing diagram

→ Components of control unit
1. Two decoders
2. A sequence counter
3. Control logic gates

→ An instruction read from memory is placed in the instruction register (IR). In control unit the IR is divided into three parts: I bits, the operation code (12-14) bits, and bits 0 through 11.

→ The operation code in bits 12 through 14 are decoded with a 3×8 decoder.



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

→ The eight outputs of the decoder are designated by the symbols D_0 through D_7. Bit 0 through 11 are applied to the control logic gates. The 4 bits sequence control can count in binary from 0 through 15. The output of counter are decoded into 16 timing signals T_0 through T_{15}.

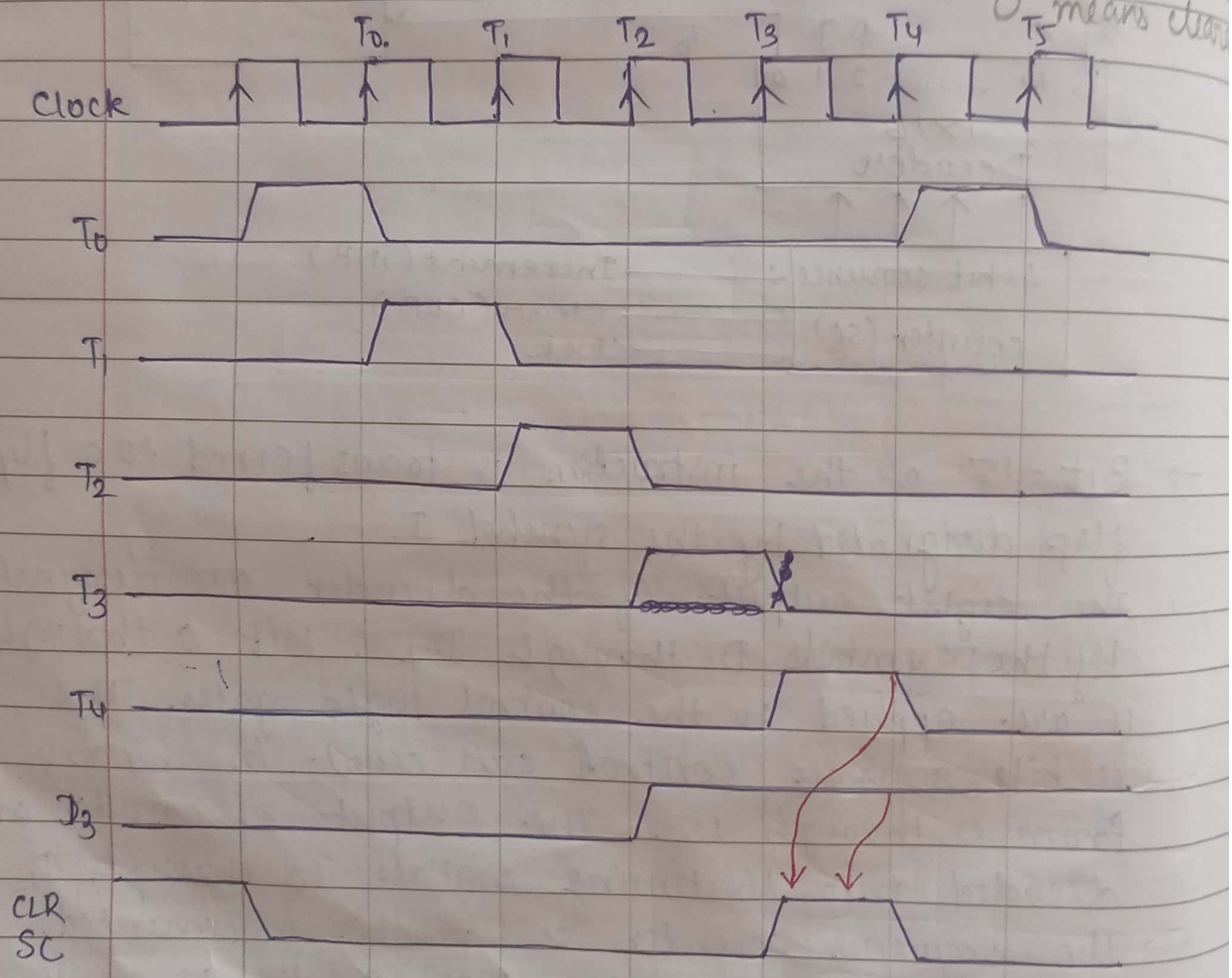
→ The sequence counter SC can be incremented or clear synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of 4×16 decoder. Once in a while, the counter is cleared to 0, causing the next timing signal to be T_0.

→ As eg, consider the case where SC is incremented to provide timing signals T_0,

T_1, T_2, T_3 and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement

$$D_3 T_4 : SC \leftarrow 0$$

when D_3 and
both are 1 at
D_3 and T_4 the
SC is made to
0 means clear



→ The last three waveform shows how SC is cleared when $D_3 T_4 = 1$ output D_3 from the operation decoder becomes active at the end of timing signal T_2. When timing signal T_4 becomes active, the output of the AND gate that implements the control function $D_3 T_4$ 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 consists of a sequence of instruction. 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

→ After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction

→ This process continue unless a HALT instruction is encountered.

Fetch & Decode

- PC is loaded with the address of the first instruction in the program.
- The microoperations for fetch and decode phases are as follows:

T_0: AR ← PC

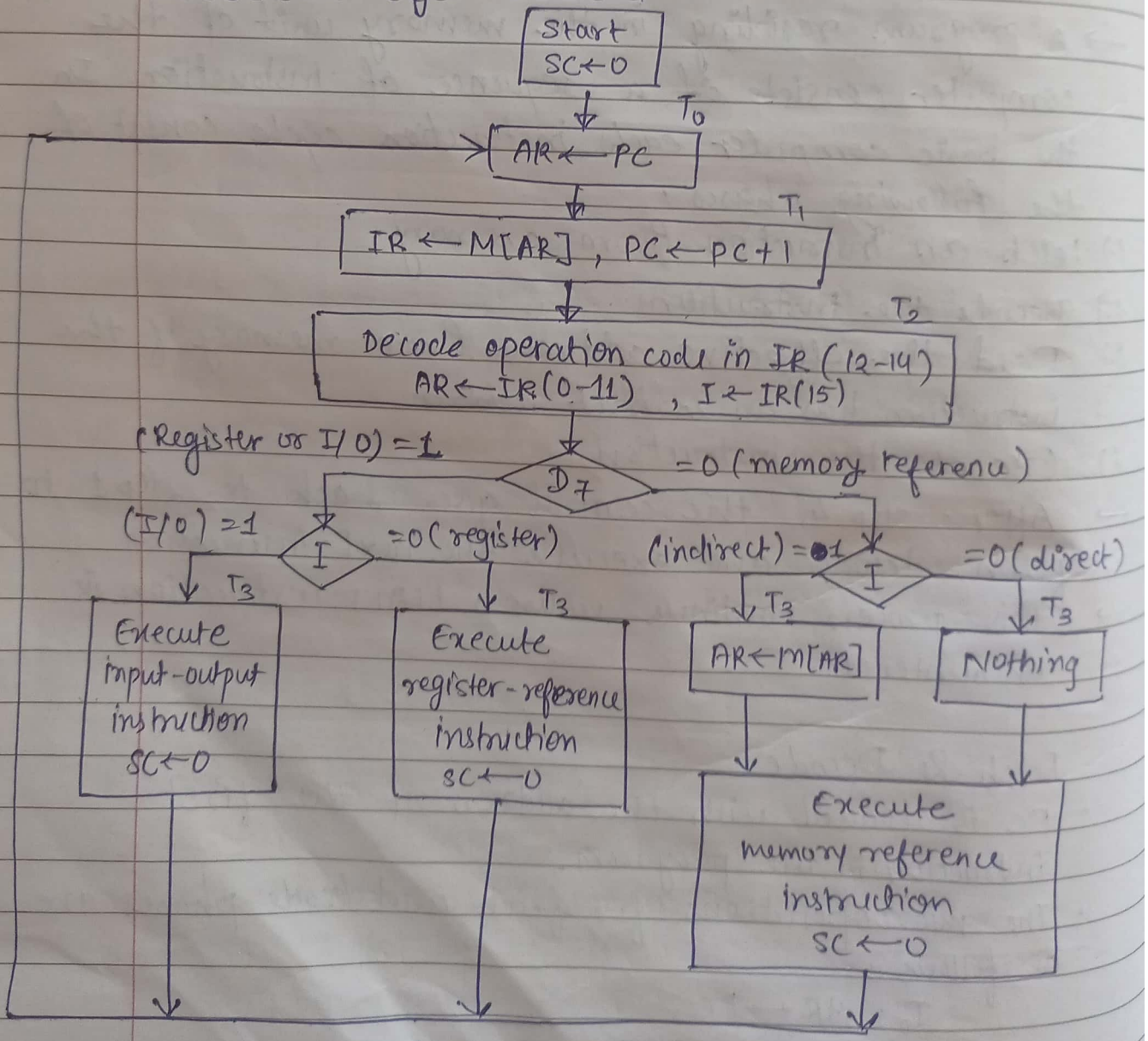
T_1: IR ← M[AR], PC ← PC + 1

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

Instruction Cycle

- Determine the type of instruction
  - → During time T_3, the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-output instruction.
  - → If $D_7 = 1$ then instruction must be register reference or input-output else memory reference instruction.

Instruction Cycle Flowchart.



→ The flowchart present an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding

→ If $D_7 = 1$, the instruction must be register-reference or input-output type. If $D_7 = 0$, the operation code must be one of the other seven values 110, specifying a memory-reference instruction. Control then inspects the value of the first bit of the instruction, which 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 three instruction type are subdivided into four seperate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

$D'7\,I\,T3$: $AR \leftarrow M[AR]$

$D'7\,I'\,T3$: Nothing

$D7\,I'\,T3$: Execute a register-reference instruction

$D7\,I\,T3$: 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 $D'7\,I\,T3 = 1$, so that the execution of the memory reference instruction can be ~~cour~~ continued with timing variable T4.

→ A register-reference or input-output instruction can be executed with the click associated with timing signal T3. After the instruction is executed, SC is

cleared to 0 and control returns to the fetch phase with $T_0 = 1$. SC is either incremented or cleared to 0 with every positive clock transition.

Register Reference Instruction

→ When the register reference instruction is decoded, D_7 is set to 1.

CLA	rB_{11}	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}	$E \leftarrow 0$	Clear E
CMA	rB_9	$AC \leftarrow AC'$	Complement AC
CME	rB_8	$E \leftarrow E'$	Complement E
CIR	rB_7	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	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 AC is positive
SNA	rB_3	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if AC is negative
SZA	rB_2	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is zero
SZE	rB_1	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E is zero
HLT	rB_0	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt Computer

Memory Reference Instruction

1. AND : 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.

$D_0 T_4 : DR \leftarrow M[AR]$

$D_0 T_5 : AC \leftarrow AC \wedge DR, SC \leftarrow 0$

2. ADD : ADD to AC

→ This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip flop.

$D_1 T_4$: $DR \leftarrow M[AR]$

$D_1 T_5$: $AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow 0$

3. LDA : Load to AC

→ This instruction transfers the memory word specified by the effecive address to AC.

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

$D_2 T_5$: $AC \leftarrow DR$, $SC \leftarrow 0$

4. STA : Store AC

→ This instruction stores the content of AC into the memory word specified by the effective address

$D_3 T_4$: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

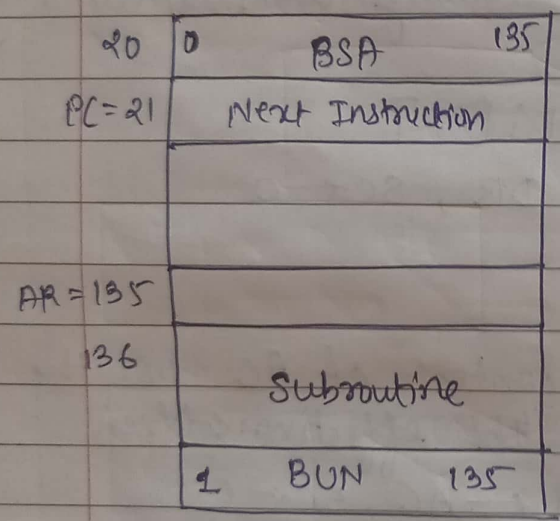
5. BUN : Branch Unconditionally

This instruction transfer the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out-of sequence and the program branches (or jumps) unconditionally.

$D_4 T_4$: $PC \leftarrow AR$, $SC \leftarrow 0$

6. BSA: Branch and Save Return Address
This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the program called a subroutine or procedure. When executed, then 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.

$D_5 T_4$: M[AR] ← PC, AR ← AR + 1.
$D_5 T_5$: PC ← AR, SC ← 0.



memory, PC and AR at time T_4

7. ISZ : Increment and Skip If Zero.
These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. 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

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 $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

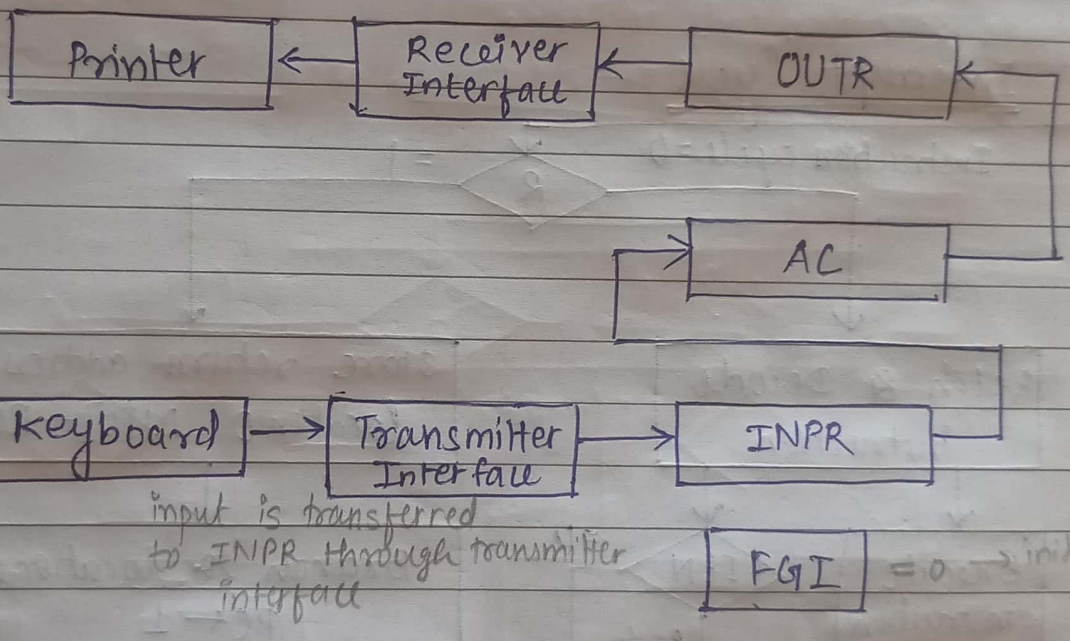
Input - Output Instructions

Input-Output terminal

Serial communication interface

Computer registers and flip flops

FGO = 1



→ Data from INPR can only be ~~transfered~~ transfered to ~~regis~~ AC when FGI is 1. So

→ When FGO is 1 content of AC is transfered to OUTR register.

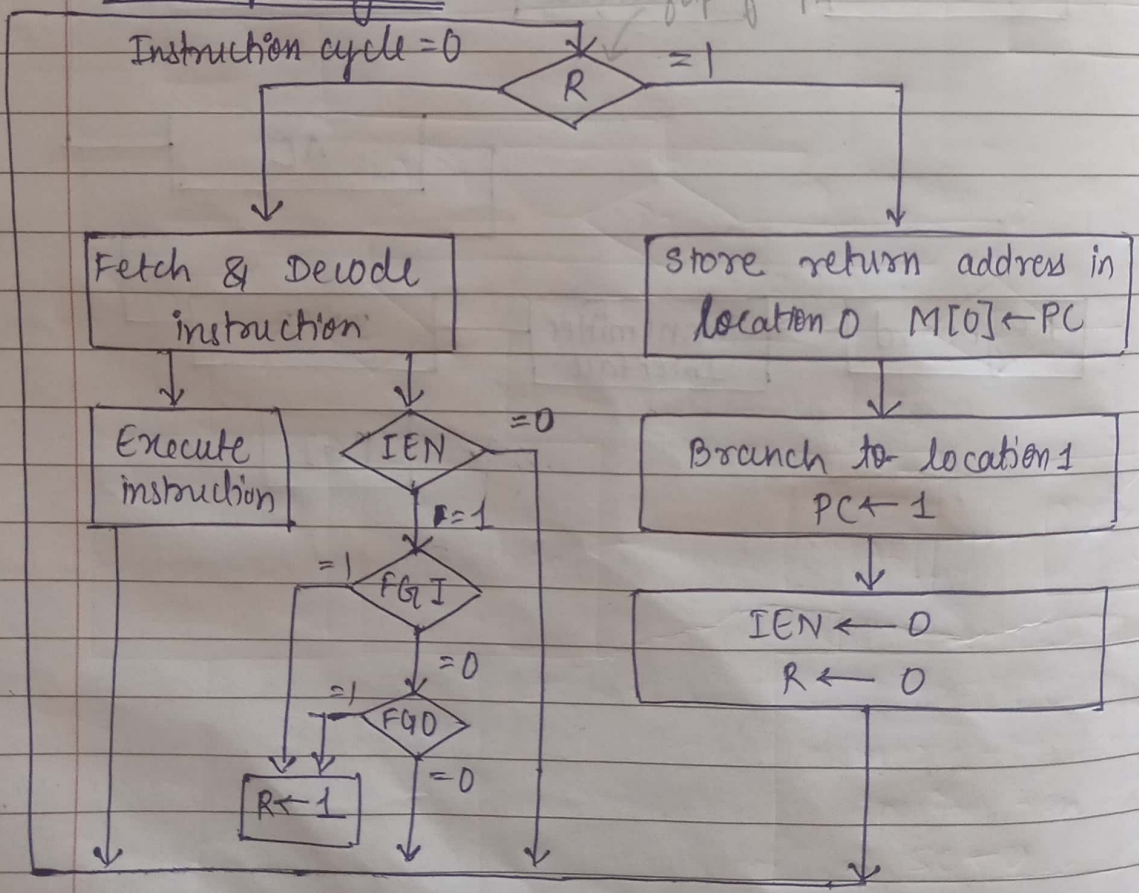
→ When FGO is set 0, OUTR register will send its data to receiver interface and through receiver interface it will send its data to printer.

$D_7 I T_3 = p$ (common to all input-output instruct
$IR(i) = B_i$ [bit in IR (6-11) that specifies the oper

INP	pB_{11}	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input Character
OUT	pB_{10}	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	Output Character
SKI	pB_9	If (FGI=1) then ($PC \leftarrow PC+1$)	Skip on input fla
SKO	pB_8	If (FGO=0) then ($PC \leftarrow PC+1$)	Skip on output f
ION	pB_7	$IEN \leftarrow 1$	Interrupt enable
IOF	pB_6	$IEN \leftarrow 0$	Interrupt enable

Interrupt Cycle

flip flop



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

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

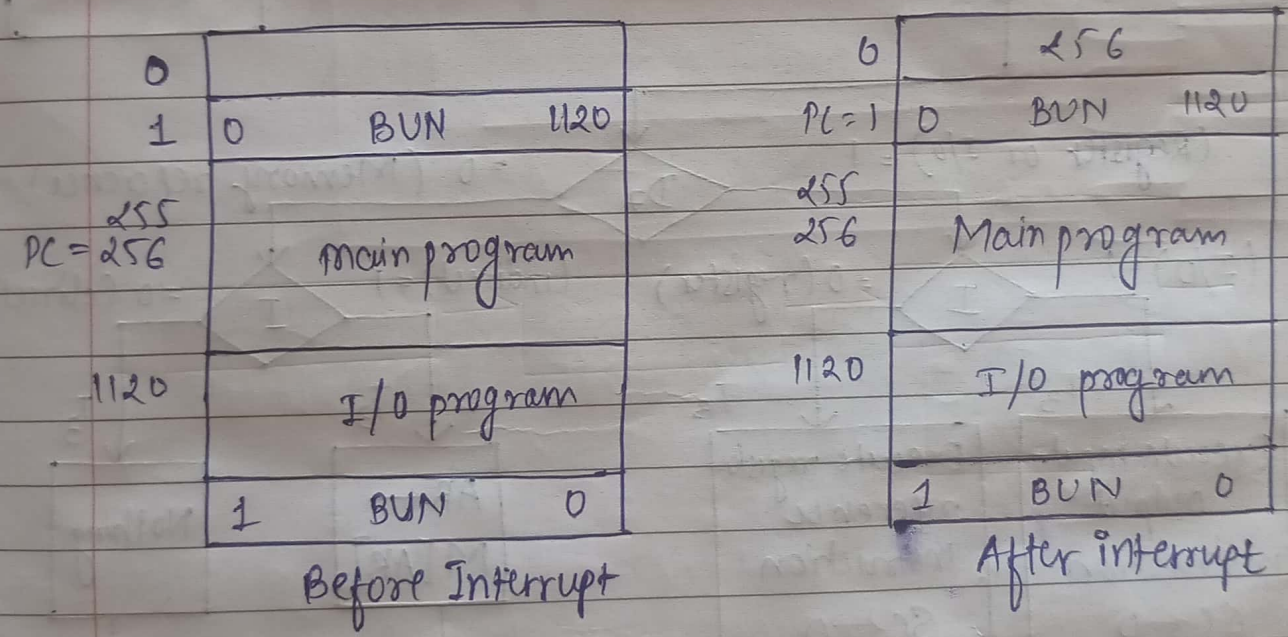
→ Therefore the interrupt cycle statement are:

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

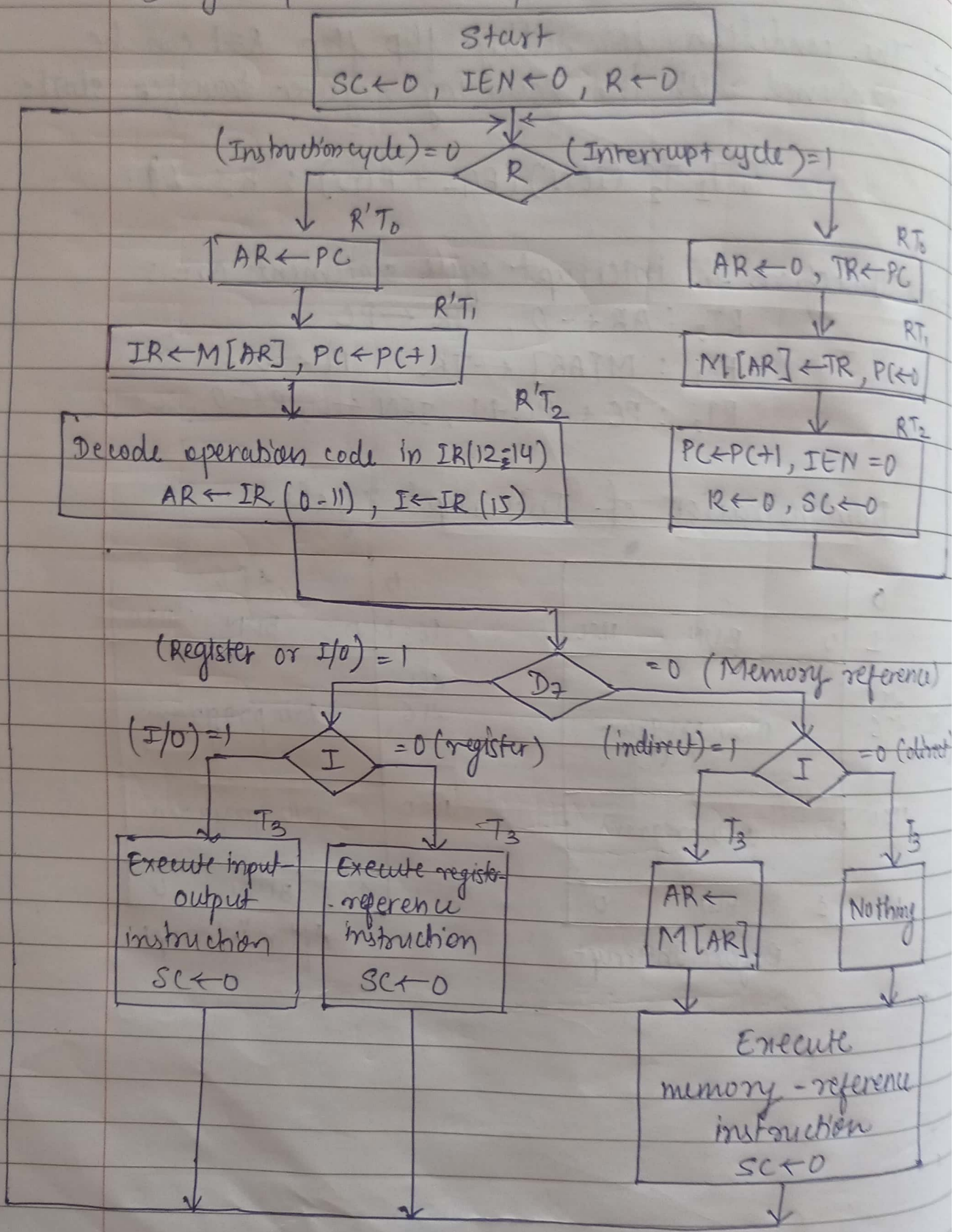
$RT_1 : M[AR] \leftarrow TR, \quad PC \leftarrow 0$

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

Demonstration of Interrupt cycle



Design of Basic Computer

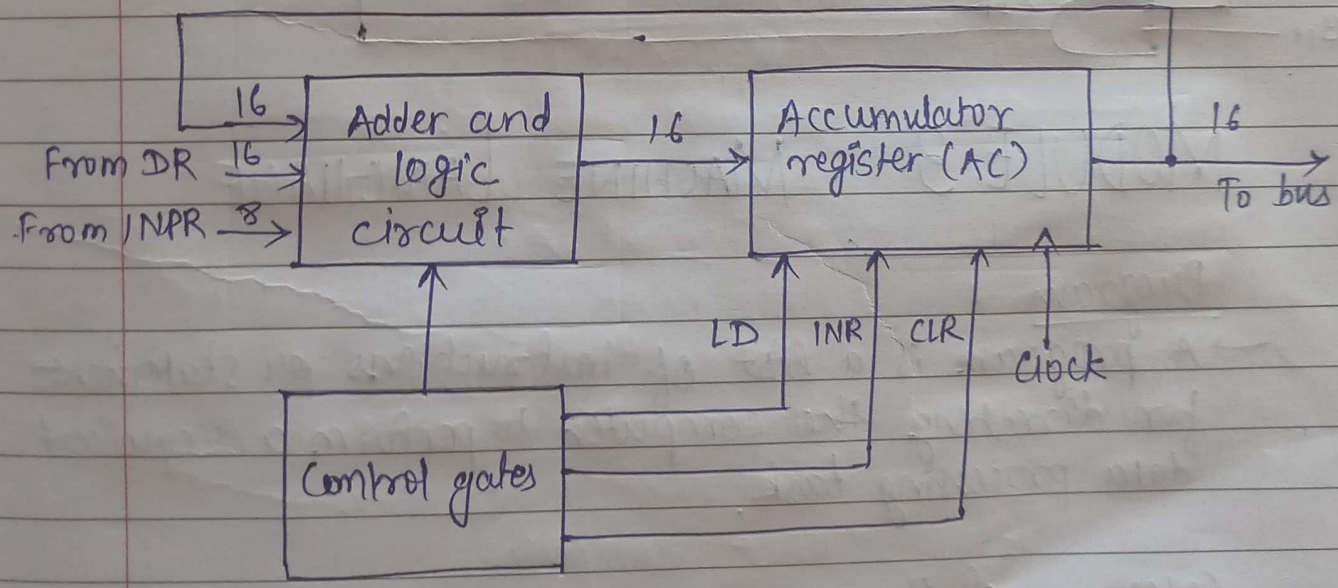


Design of Accumulator Logic

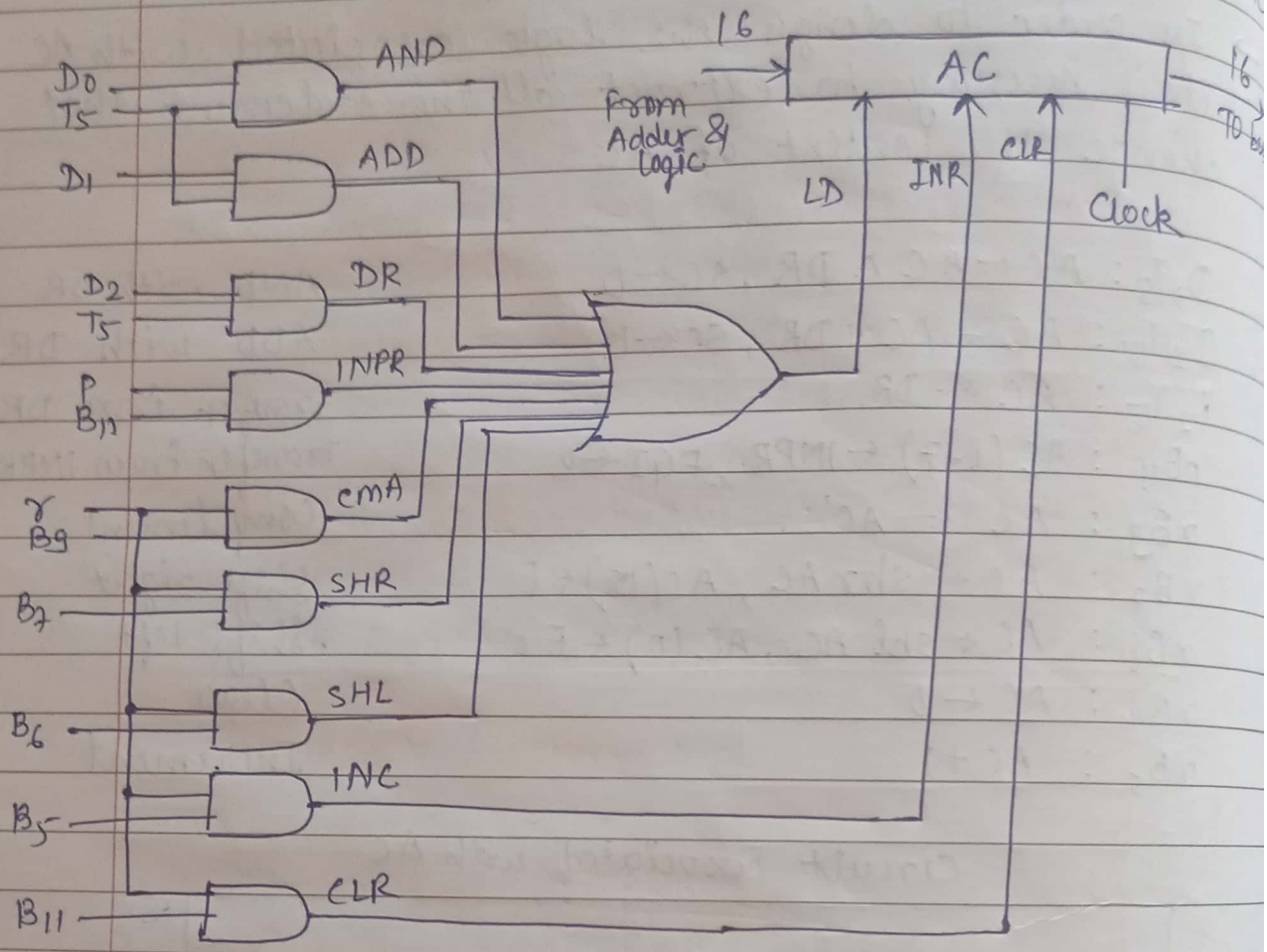
→ In order to design the logic associated with AC it is necessary to extract all the statement that change the content of AC

D_0T_5 : $AC \leftarrow AC \wedge DR, SC \leftarrow 0$	AND with DR
D_1T_5 : $AC \leftarrow AC + DR, SC \leftarrow 0$	ADD with DR
D_2T_5 : $AC \leftarrow DR$	Transfer from DR
pB_{11} : $AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Transfer from INPR
rB_9 : $AC \leftarrow AC'$	Complement
rB_7 : $AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6 : $AC \leftarrow shl\ AC, AC(0) \leftarrow E$	Shift left
rB_{11} : $AC \leftarrow 0$	Clear
rB_5 : $AC + 1$	Increment

Circuit associated with AC



Gate structure for controlling LD, INR and CLR of AC



MODULE 3 MACHINE LANGUAGE

Program

→ A program is a list of instructions or statements for directing the computer to perform a required data processing task.

Categories of Program.

- Binary code
- Octal or hexadecimal code
- Symbolic code
- High level programming language

Assembly Language

Assembler

- An assembler is a program that accepts a symbolic language program and produce its binary machine language equivalent.
- The input symbolic program is called the source program and the resulting binary program is called the object program.
- The assembler is a program that operates on character strings and produce an equivalent binary interpretation.

Pseudo Instruction

- A pseudo instruction is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation.

Symbol	Information for the Assembler
ORG N	Hexadecimal number N is the memory location for the instruction or operand listed in the following line.
END	Denotes the end of symbolic program
DEC N	Signed decimal number N to be converted to binary.
HEX N	Hexadecimal number N to be converted to binary

Memory Refrence Instruction

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

Register Refrence Instruction

7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC
7100	CME	Complement E
7080	CIR	circuit right AC and E
7040	CIL	~~cirq~~ circuit left AC and E
7020	INC	Increment AC
7010	SPA	Skip next instruction if AC is +ve
7008	SNA	Skip next instruction if AC is -ve
7004	SZA	Skip next instuction if AC is zero
7002	SZE	Skip next instruction if E is zero
7001	HLT	Halt computer.

Input Output Instruction

F800	INP	I/p character to AC
F400	OUT	O/p character from AC
F200	SKI	Skip on input flag

F100	SKO	Skip on output flag.
F080	ION	Interrupt on
F040	IOF	Interrupt off

Converting Assembly Code to Binary

code for LDA = 2

Location	Instruction	Comment
000	2 LDA 004	Load first operand into AC
001	ADD 005	Add second operand to AC
002	STA 006	Store sum in location 006
003	HLT	Halt computer
004	0053 ← Assembly code	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

↓ Hexa code:

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

Binary →

Location	Instruction Code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

First Pass of an assembler