LECTURE-

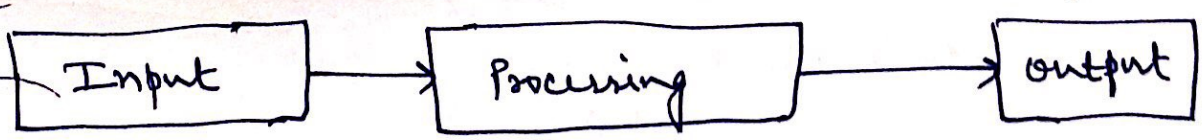
RCS-302 : Computer Organization and Architecture

Contents : Functional units of digital system and their interconnections.

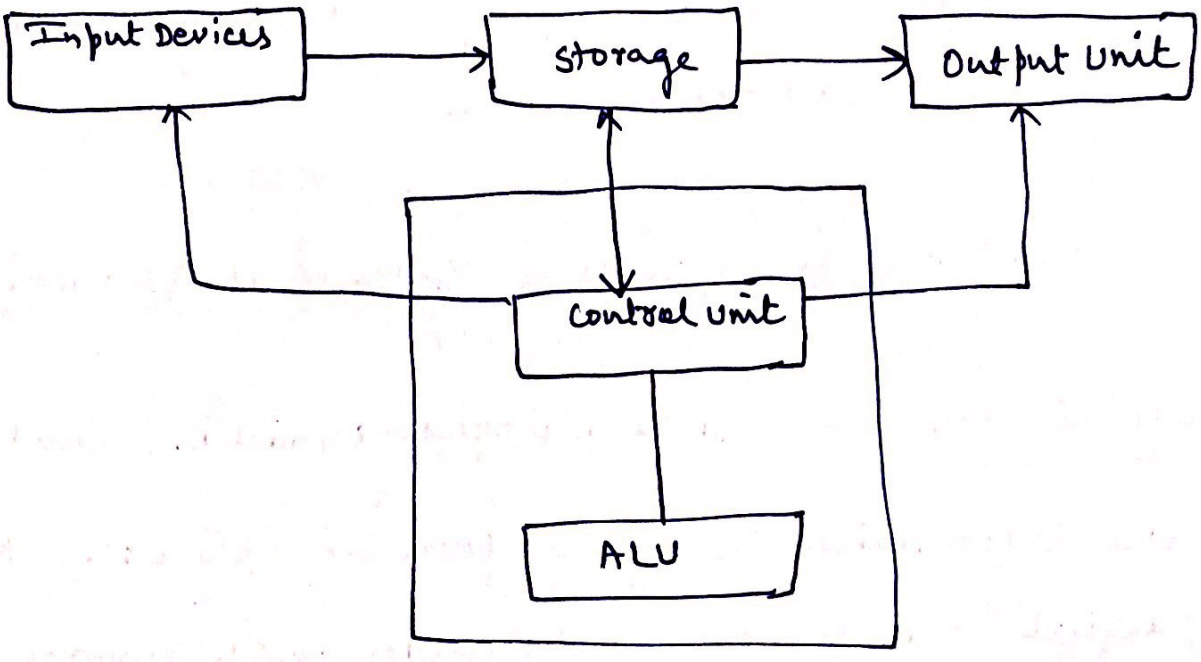
Functional units of digital system & their Interconnections

Digital computer : It is a processing machine that process the information in digital form i.e (0's & 1's). Means digital computer can only understand binary language (0's & 1's). If any analog quantity is to be processed, they must be converted into digital form before processing.

- The block diagram of a digital computer is shown in figure. whatever may be the type, size and capacity of the computer, it should have these five blocks.



Digital Computer Functional Unit



Block diagram

Functional units of computers:

- The computer consists of four main parts. These are as follows:
  1) central Processing unit
  ii) Memory
  iii) Input devices
  iv) Output devices

Central Processing Unit (CPU):

The CPU is the place where computations are performed. It is the brain and heart of a computer.

- CPU interprets instruction and process data contained in computer program.
- CPU has two components
  1. ALU (Arithmetic + Logic Unit)
  2. CU (Control Unit)

The ALU of the CPU performs the typical arithmetic operations such as addition, subtraction, multiplication and division. computers use the binary number system, to represent numbers. The binary number system has only two digits 0 and 1.

Control unit (CU) control all the operation in computer in computer. It is the controller of the system.

- It also control all the devices connected to CPU.
- It also control the flow of data from input devices to memory and memory to output devices.

Memory: Memory is the location where data and programs are stored while being processed by the CPU. Memory is the main storage unit in a computer. Memory consists of primary (main) memory and secondary memory. The data stored in main memory (RAM) is volatile and is erased as soon as the power supply is cut off. Therefore, secondary memory is used to store data. In secondary memory data is stored permanently.

Input Devices: Input device is the process of entering and translating incoming data into machine readable form.

- Any hardware item which attached to the main unit of a computer that comes the CPU is referred to as peripheral device.
- An input device is a peripheral device through which data are entered and transformed into machine readable form. Input devices are mainly used to communicate information between humans and computer.

Output Devices: An output device is a peripheral device that allows a computer to communicate information to humans or another machine by accepting data from computer and transforming them into a usable form. The output devices gives the desired result to the user. Exp: Monitor, Printer, plotter etc.

BUS: A group of wires connecting two or more devices and providing a path to perform communication is called bus.

- A bus that connect major computer component such as (CPU, Memory, I/O) is called system bus.
- A bus is a set of physical connections (cables, printed circuits, etc) which can be shared by multiple hardware components in order to communicate with one another. The purpose of buses is to reduce the number of "pathways" needed for communication between the components, by carrying out all communications over a single data channel.

Kinds of bus inside the system

There are three main bus groups or system bus can be separated into three functional groups

- Address Bus
- Data Bus
- Control Bus

Data Bus :- The data bus carries the data which is transferred throughout the system.

- It is bidirectional like ⟷

- Examples of data transfers
  - Program instructions being read from memory into MPU (microprocessor unit)
  - Data being sent from MPU to I/O port
  - Data being read from I/O port going to MPU
  - Results from MPU sent to memory
- There are called read and write operations.

~~UNIT~~

~~LECTURE~~

~~ECE-202 : Computer Organization and Architecture~~

~~Contents : Floating point arithmetic operation~~

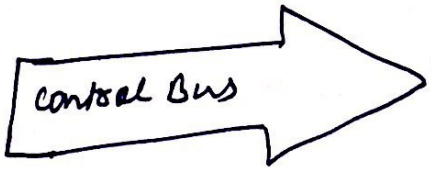
Address Bus : An address is a binary number that identifies a specific memory storage location or I/O port involved in a data transfer.

- The address bus is used to transmit the address of the location to the memory or the I/O port

- The address bus is unidirectional (one way): addresses are always issued by the MPU.

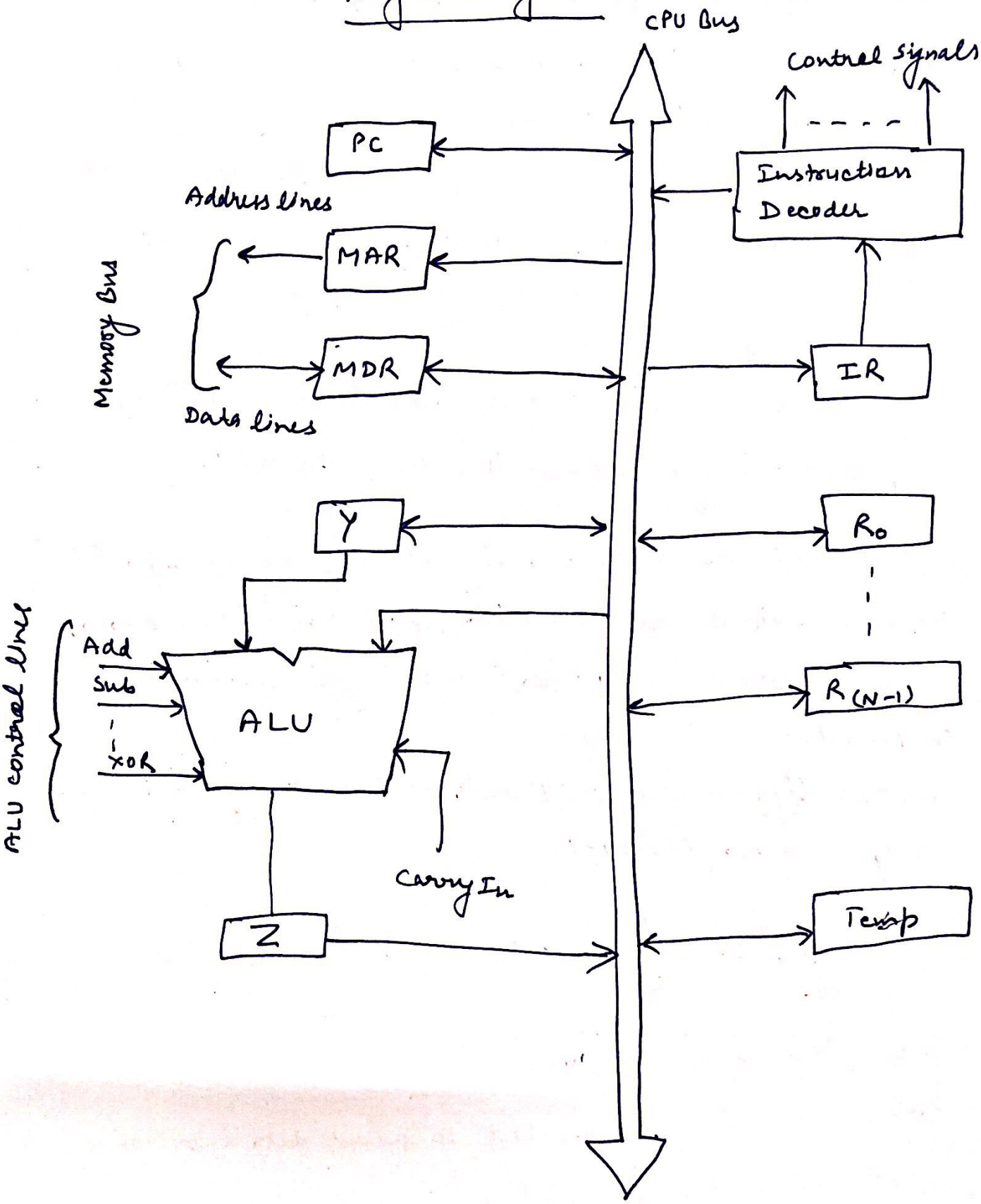
Control Bus : The control bus is another group of signals whose functions are to provide synchronization (timing control) between the MPU and the other system components.

- Control signals are unidirectional, and are mainly outputs from the MPU.



- Examples control signals
  - RD: read signal asserted to read data into MPU.
  - WR: write signal asserted to write data from MPU.

Single Bus System



LECTURE-

RCS-302 : Computer Organization and Architecture

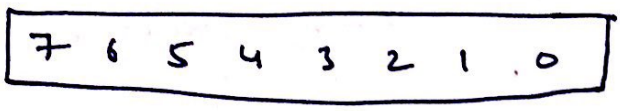
Content : Register, Bus, and Memory transfer, Processor organization.

Register : A register is a group of flip-flops to store the binary information. computer registers are designated by capital letters (sometimes followed by numarals) to denote the function of the register.

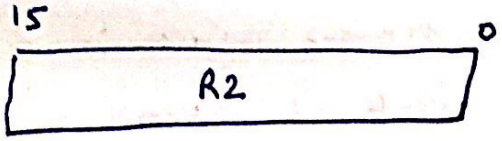
Exp: The register that holds an address for the memory unit is usually called a memory address rejister and is designated by the name MAR. others are PC (Program counter), R1 (Processor Register)

The individual flip flops in one n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and the increasing the numbers toward the left.

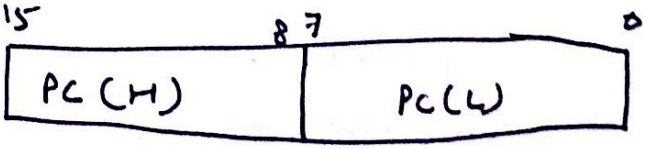
(a) Register R



(b) showing individual bits



(c) numbering of bits



(d) Divided into two parts.

In figure(d), bits 0 through 7 are assigned the symbol L (for Low byte) and bits 8 through 15 are assigned the symbol H (for High bytes). The name of 16-bit register is PC.

Register Transfer: Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement

$$R2 \leftarrow R1$$

denotes a transfer of the content of register R1 into register R2. By definition, the content of the source register R1 does not change after the transfer.

Normally, we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.

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

Where P is a control signal generated in the control section. It is sometimes convenient to separate the control variables from the

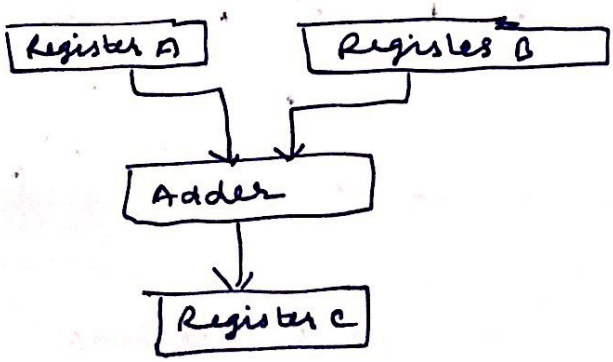
Basic symbol for Register transfer

Symbol	Description	Example
Letters	denotes a register	MAR, R2
Paranthesis ( )	denotes a part of a register	R2(0-7), R2(L)
Arrow ←	denotes transfer of information	R2 ← R1
comma ,	separates two micro operations	$R_2 \leftarrow R1$, $R_1 \leftarrow R2$

T: R2 ← R1, R1 ← R2

denotes an operation that exchange the contents of two registers during one common clock pulse provided that T=1. This simulation operation is possible with registers that have edge-triggered flip-flops.

* C ← A+B



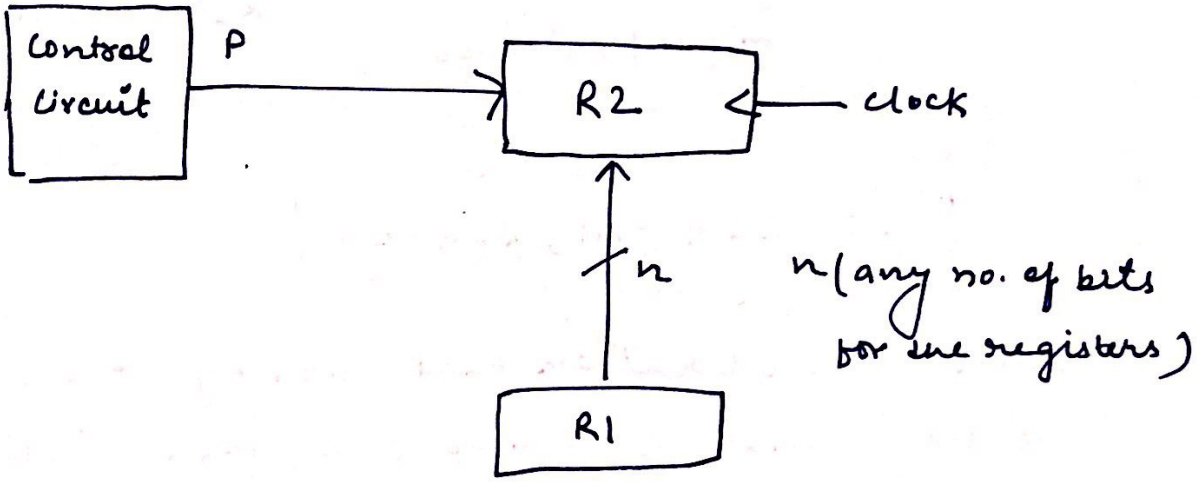
Hardware Implementation for addition micro-operation

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

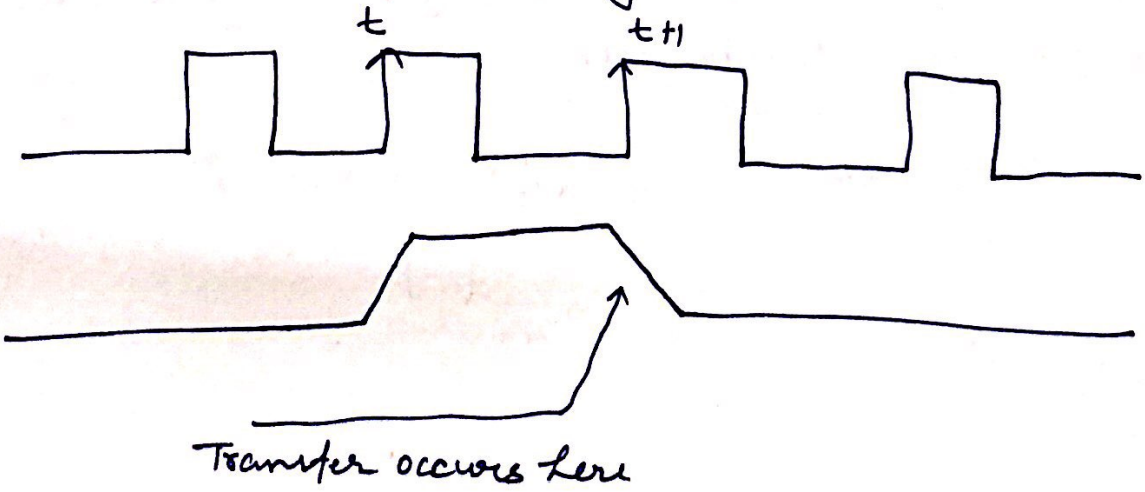
P: R2 ← R1

The control condition is terminated with a colon.

transfer from R1 to R2 when P=1

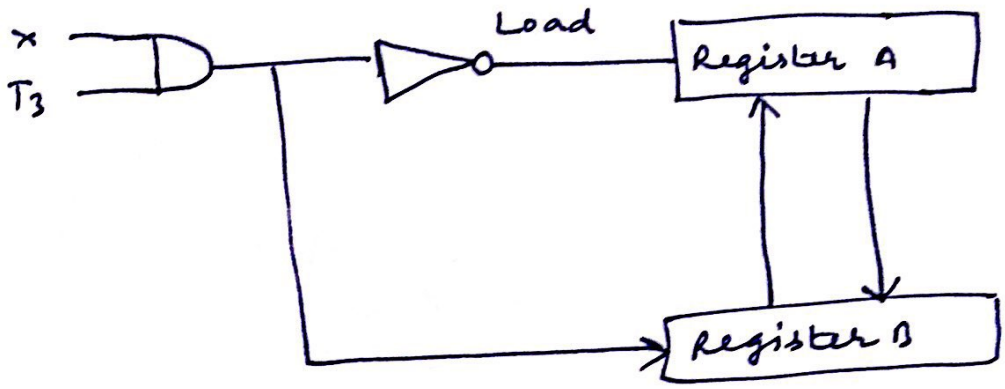


(a) Block diagram

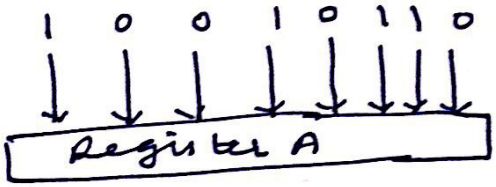


(b) Timing diagram

2. $xT_3 : A \leftarrow B,\ B \leftarrow A$

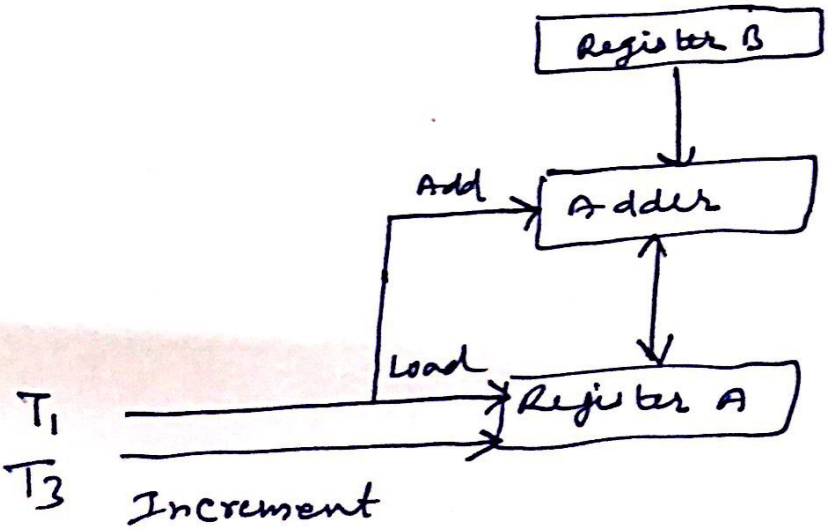


3. $X : A \leftarrow 10010110$



4. $T_1 : A \leftarrow A + B$

$T_3 : A \leftarrow A + 1$



Bus and Memory Transfers:

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 excessive. if separate lines are used between each register and all other registers in the system. A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system.

- **Common Bus System:**

A bus structure consists of a set of common lines, one for each bit of a register through which binary information is transferred one at a time.

Control signals determines which register is selected by the bus during each particular register transfer.

One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus.

The construction of a bus systems for four registers is shown in figure.

Each register has four bits, numbered 0 through 3. The bus consists of four 4×1 multiplexers each having four data inputs 0 through 3 and two selection inputs S_1 & S_0.

Bus selection: The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers

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

When $S_1S_0 = 00$, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

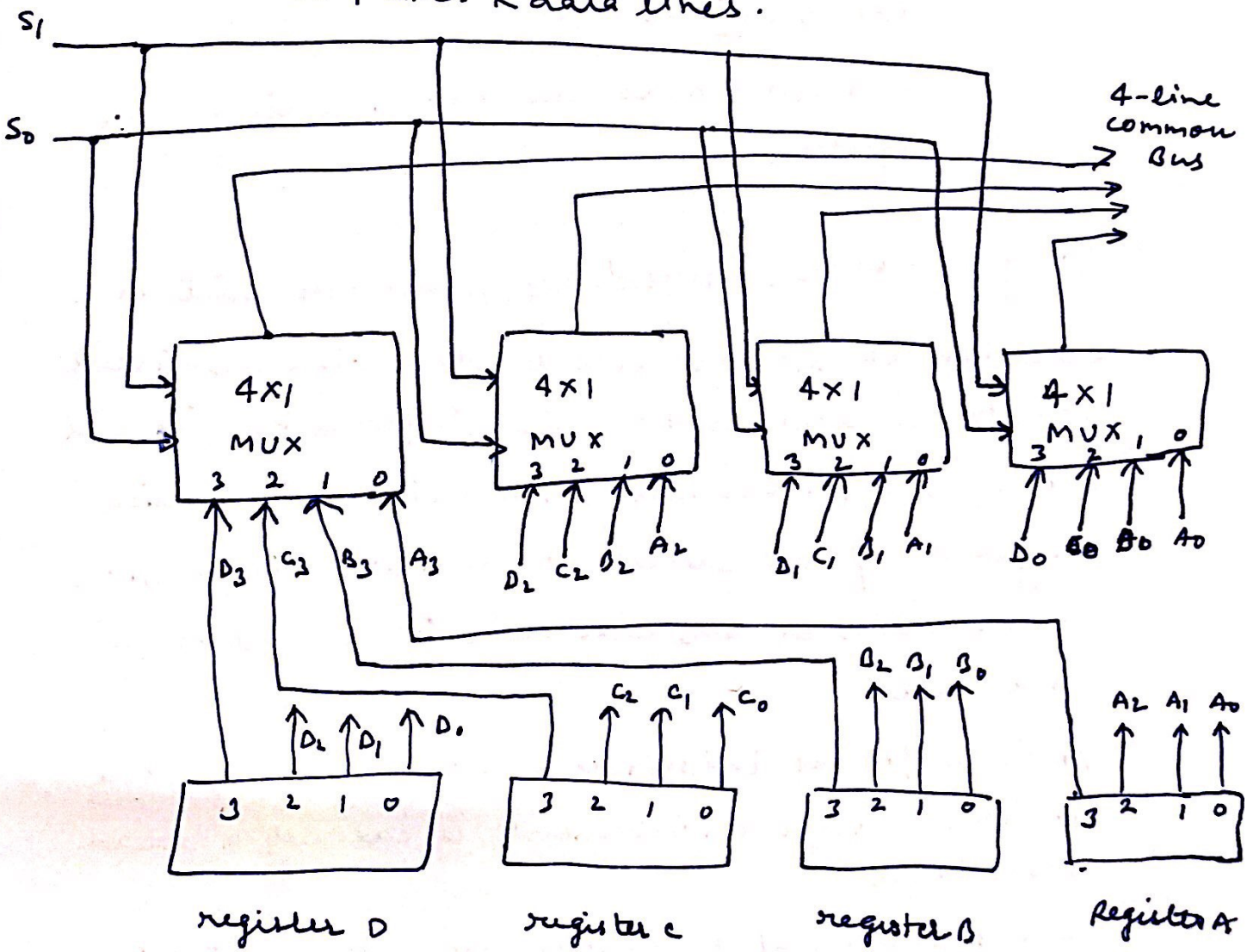
Similarly, register B is selected if $S_1S_0 = 01$ and so on.

General Rules :

1). A bus system will multiplex k- registers of n- bits each to produce an n- line common bus.

2). The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.

3). The size of each multiplexer must be K X 1. since it multiplexes K data lines.



K register = 4
Common Bus = 4
no. of multiplexers = 4, 4 bits in each register

Bus system for four registers

- A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.

(a) How many selection inputs are there in each multiplexers?

(b) What size of multiplexers are needed

(c) How many multiplexers are there in the bus.

Ans: (a) 4-selection lines to select one of 16 registers

(b) 16x1 multiplexers

(c) 32 multiplexers one for each bit of the registers.

Memory Transfers: Basically, A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage. The memory stores binary information in groups of bits called words. A word in memory is an entity of bits that move in and out of storage as a unit.

A transfer of information from a memory word to the outside environment is called a read 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.

Memory Read: A memory unit that receives the address from a register called the address register symbolized by AR. The data are transferred to another register called the data register symbolized by DR.

The read operation can be stated as follows:

read : DR ← M[AR]

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

Memory write: The write operation transfers 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. The write operation can be stated as follows:

write : M[AR] ← R1

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

Processor Organization :

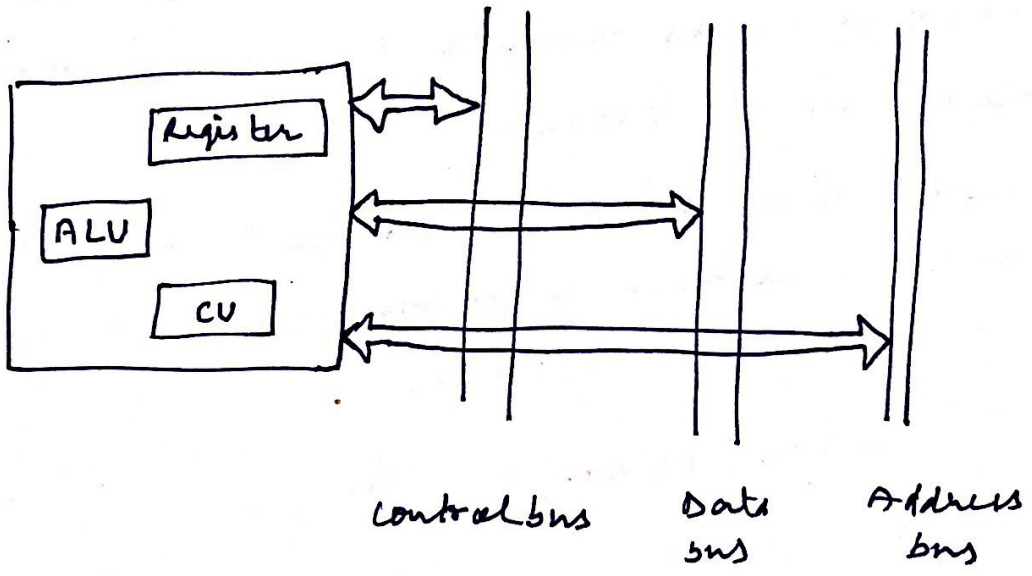
The processor is a main part of a computer; CPU is sometimes referred to as data sub system. The execution unit (Data Processing Unit) contains a set of registers for storing data and an arithmetic and logic unit for execution of arithmetic and logical operations.

The Processor has three main parts listed below :

1- Register – store data & instructions

2- ALU – Performs arithmetic & logic functions

3- CU – generates sequence of control signals



Block diagram of Processor Organization

Arithmetic Microoperation :

A microoperation is an elementry operation performed with the data stored in registers.
The microoperations most often encountered in digital computers are classified into four categories:

1- Register transfer microoperations transfer binary information from one register to another.

2. Arithmetic microoperations perform arithmetic operations on numeric data stored in registers.

3. Logic microoperations perform bit manipulation operations on non-numeric data stored in registers.

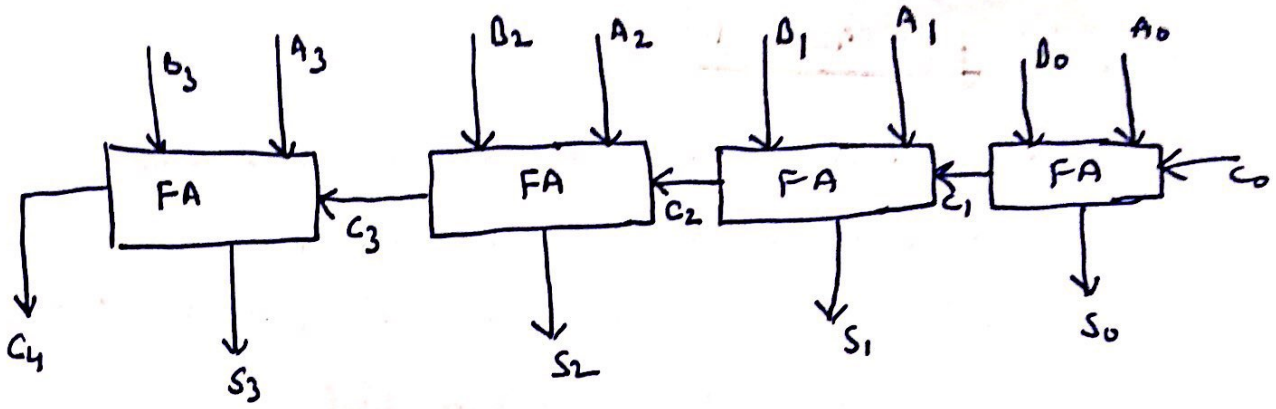
4. Shift microoperation perform shift operations on data stored in registers.

	Symbolic designation	Description
(1)	$R3 \leftarrow R1 + R2$	content of R1 plus R2 transferred to R3
(2)	$R3 \leftarrow R1 - R2$	content of R1 minus R2 transferred to R3
(3)	$R2 \leftarrow \overline{R2}$	complement of R2 (1's complement)
(4)	$R2 \leftarrow \overline{R2} + 1$	2's complement the content of R2 (negate)
(5)	$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (subtraction)
(6)	$R1 \leftarrow R1 + 1$	increment the contents of R1 by one
(7)	$R1 \leftarrow R1 - 1$	decrement the content of R1 by one

Binary Adder: To implement the add microoperation with hardware, we need the registers that holds the data and the digital component that performs the arithmetic addition. The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full adder.

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

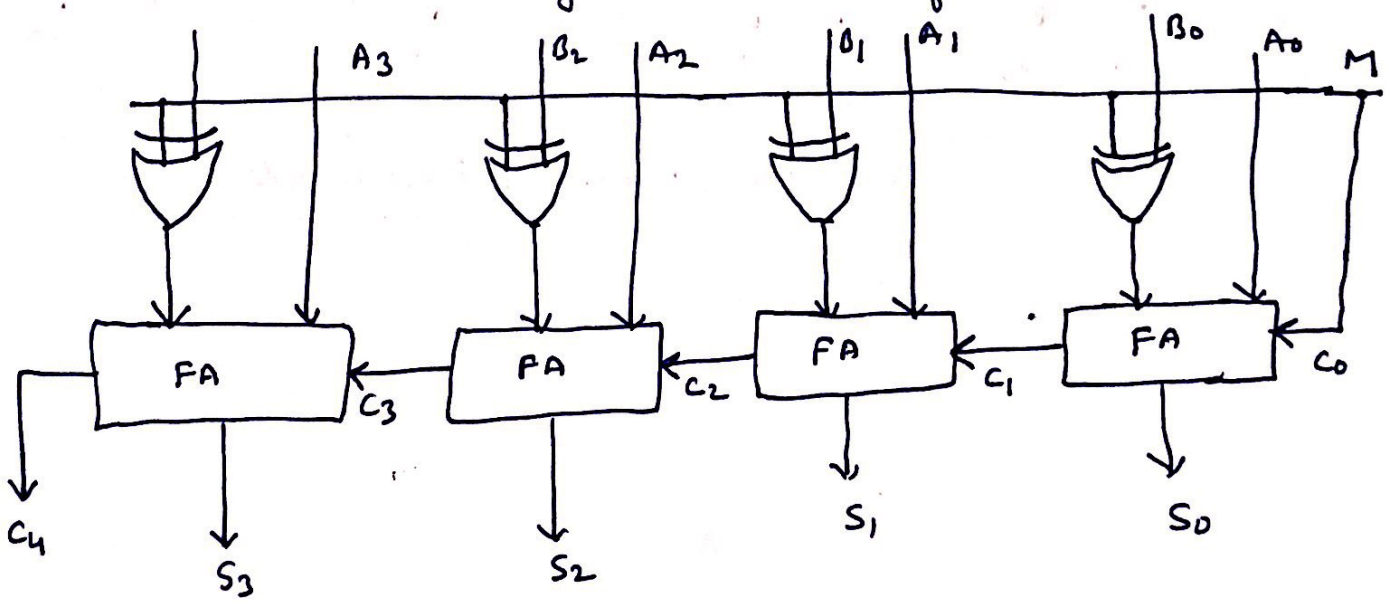
The binary adder is constructed with full adder circuits connected in cascade with the output carry from one full adder to the input carry of the next full adder.



4-bit binary adder

Binary Adder-Subtractor

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



4-bit adder-subtractor

B	A	
0	0	0
0	1	1 A
1	0	1
1	1	0 $\bar{A}$

Binary Incrementer

The increment microoperation adds one to a number in a register. for example - if a 4-bit register has a binary value 0110 it will go to 0111 after it is incremented.

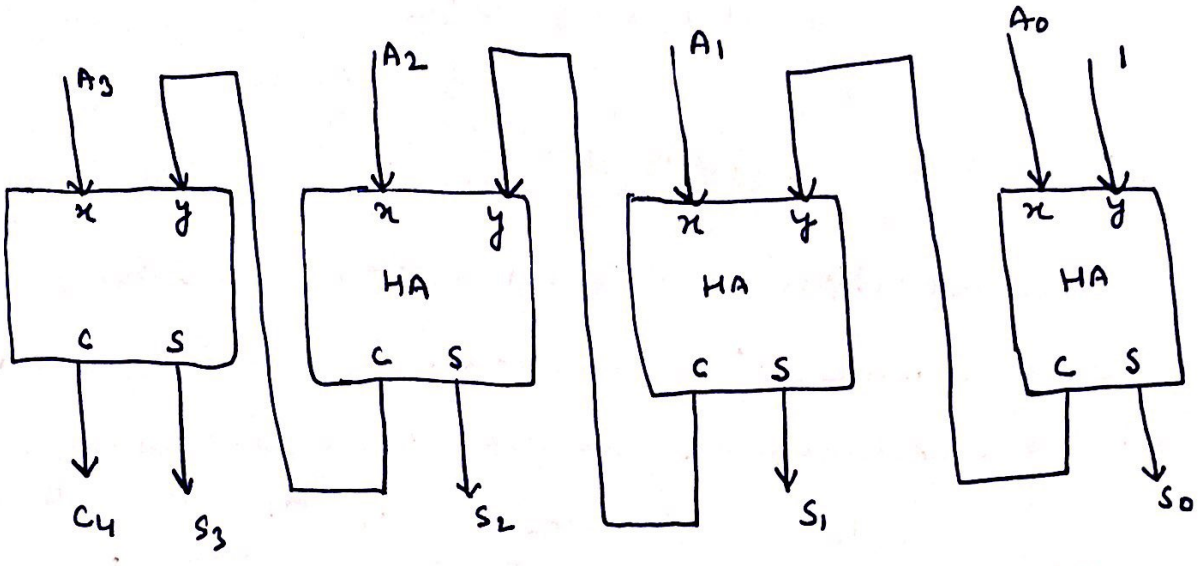


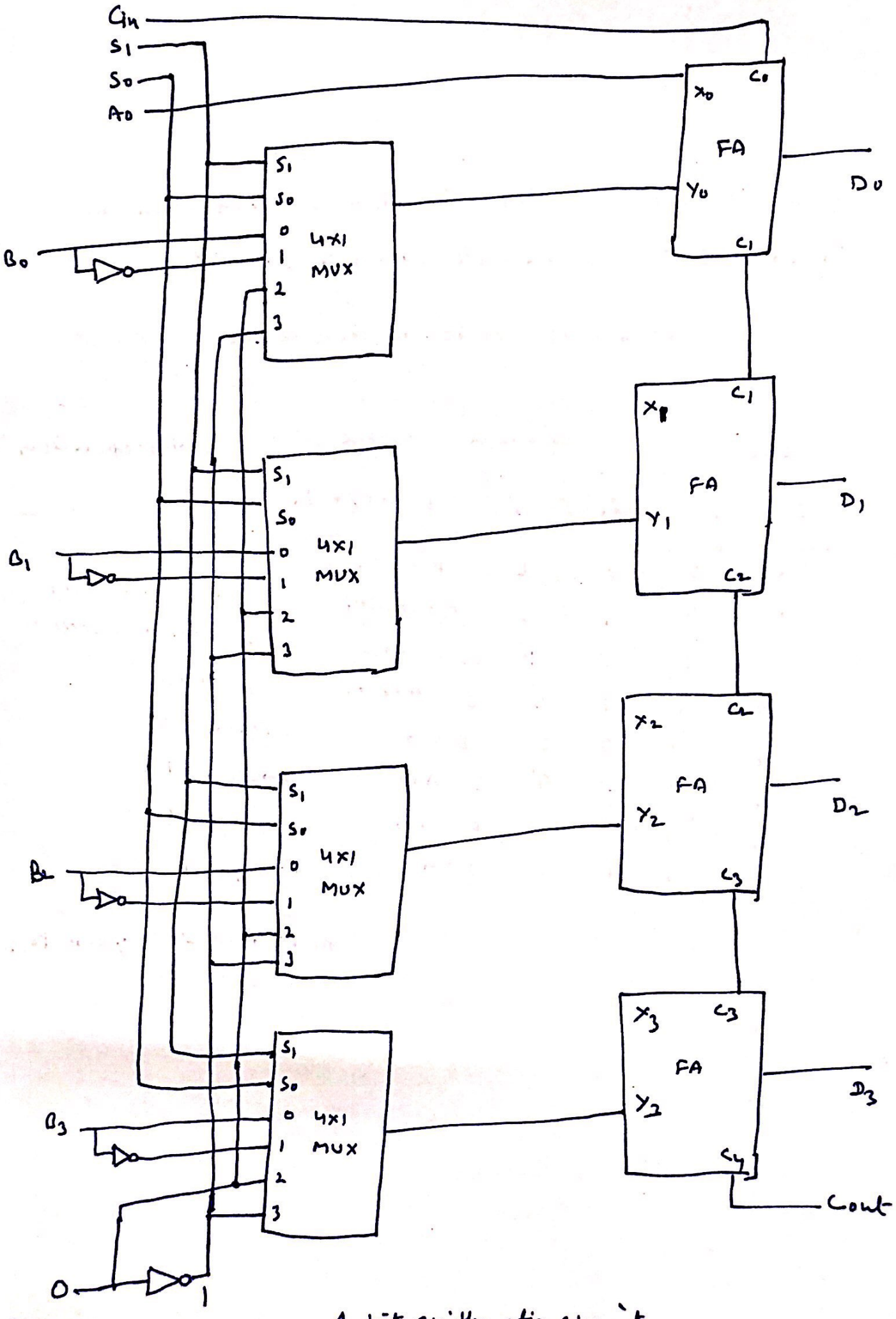
Fig: 4-bit binary incrementor

Arithmetic Circuit : Arithmetic microoperations can be implemented in one composite arithmetic circuits.

Arithmetic Circuit Function Table

Select			input	output	microoperation
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	subtract
1	0	0	0	$D = A$	Transfer A
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

All 1's mean 2's complement of 1.



4-bit arithmetic circuit

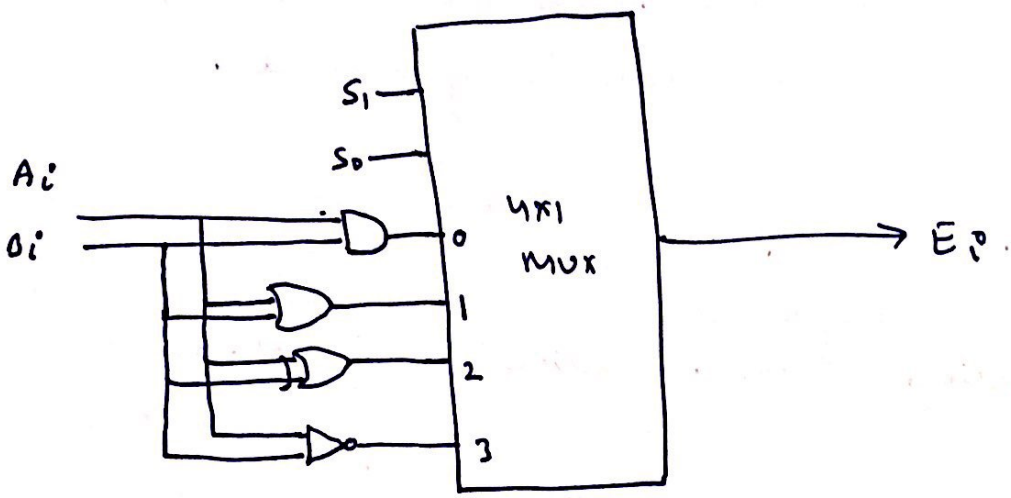
Logic Microoperation: Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variable.

For exp: The exclusive-OR microoperation with the content of two registers R1 and R2 is symbolized by the statement.

P: R1 ← R1 ⊕ R2

	Boolean Function	Microoperation	Name
(1)	$F_0 = 0$	$F \leftarrow 0$	clear
(2)	$F_1 = xy$	$F \leftarrow A \wedge B$	AND
(3)	$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
(4)	$F_3 = x$	$F \leftarrow A$	Transfer A
(5)	$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
(6)	$F_5 = y$	$F \leftarrow B$	Transfer B
(7)	$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
(8)	$F_7 = x + y$	$F \leftarrow A \vee B$	OR
(9)	$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
(10)	$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
(11)	$F_{10} = y'$	$F \leftarrow \bar{B}$	complement B
(12)	$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
(13)	$F_{12} = x'$	$F \leftarrow \bar{A}$	complement A
(14)	$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
(15)	$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
(16)	$F_{15} = 1$	$F \leftarrow$ all 1's	set to all 1's.

Hardware Implementation : The Hardware Implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the logic function.



(a) Logic diagram

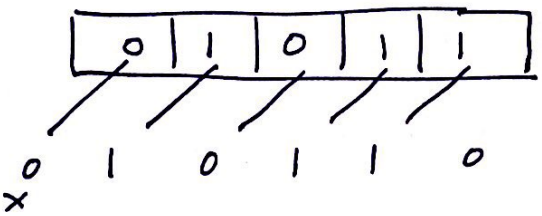
S_1	S_0	output	operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \cup B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	complement

(b) Functional Table

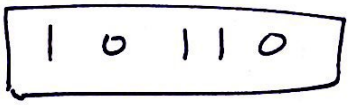
Shift Microoperations : Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic logic. The information transferred through the serial input determines the type of shifts.

- logical shift
- Circular shift
- arithmetic shift

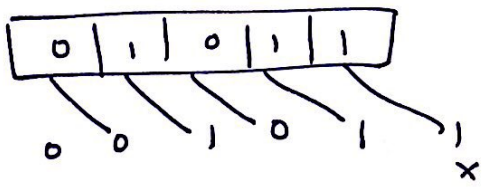
- Logical shift : A logical shift is one that transfers 0 through the serial shift.

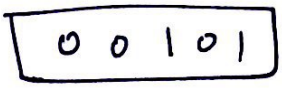


shl :



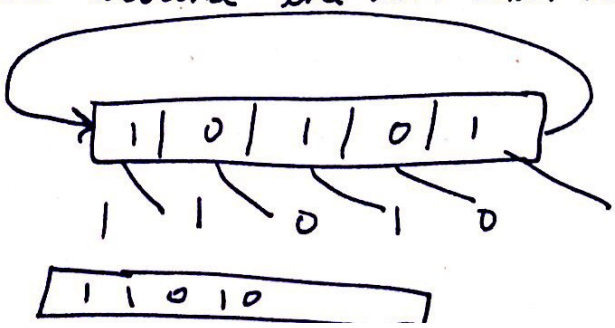
shr :



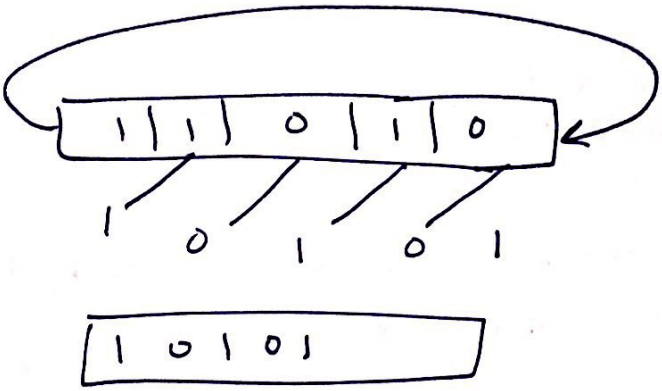


- Circular shift : The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.

cil :

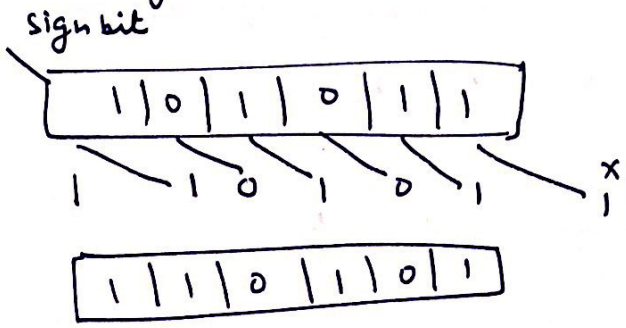


cir:

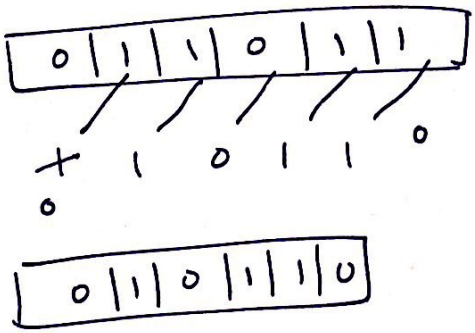


iii] Arithmetic shift: An arithmetic shift is a microoperation that shifts a signed binary number to the left or right

ashr:



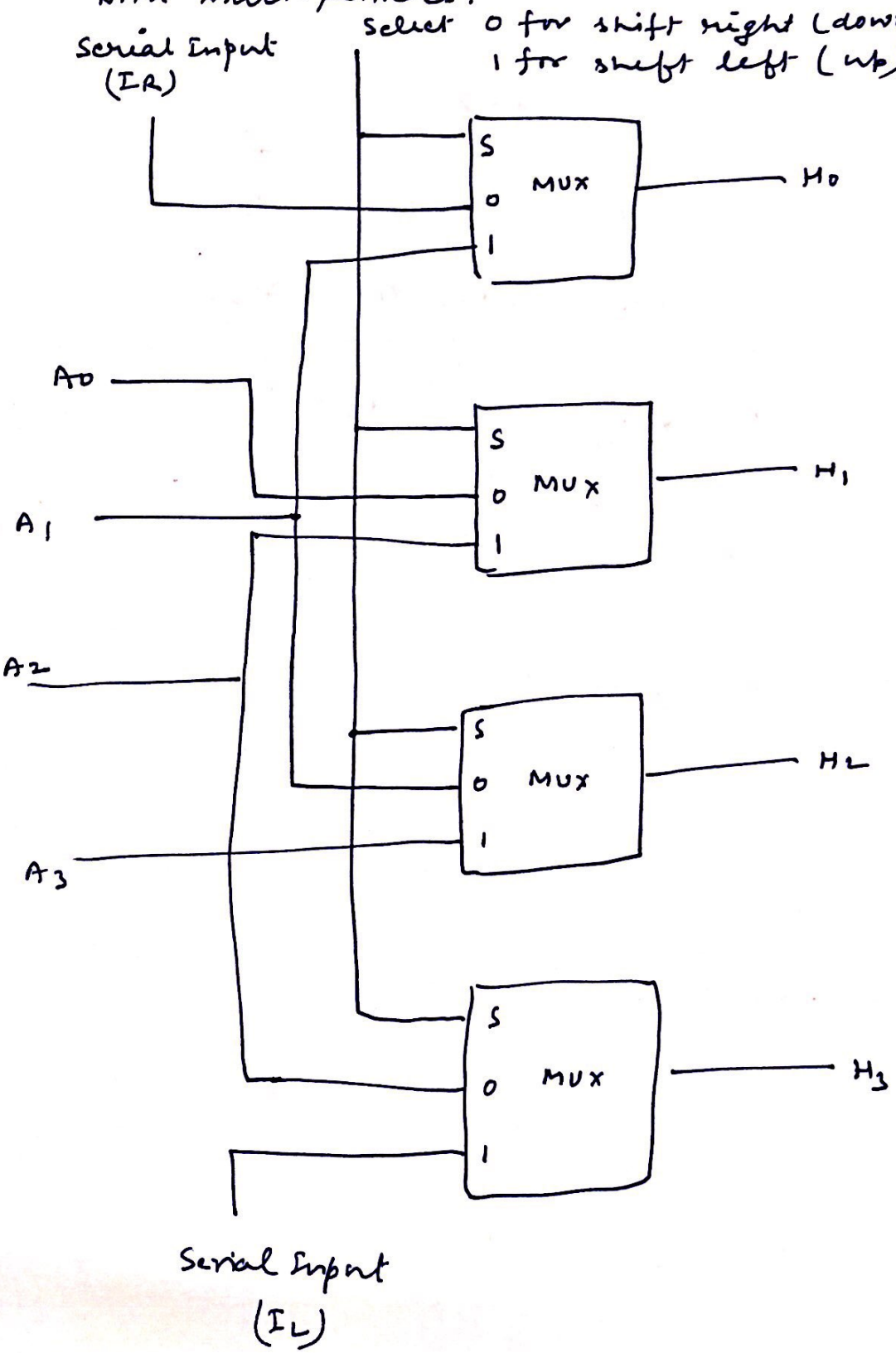
ashl



Hardware Implementation

A possible choice for a shift unit would be bidirectional shift register with parallel load. Information can be transferred to the register in parallel and then shifted to the right or left

Shifter: A combinational circuit can be constructed with multiplexers.



4-bit combinational circuit shifter

Functional Table

Select-	output			
S	M_0	M_1	M_2	M_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

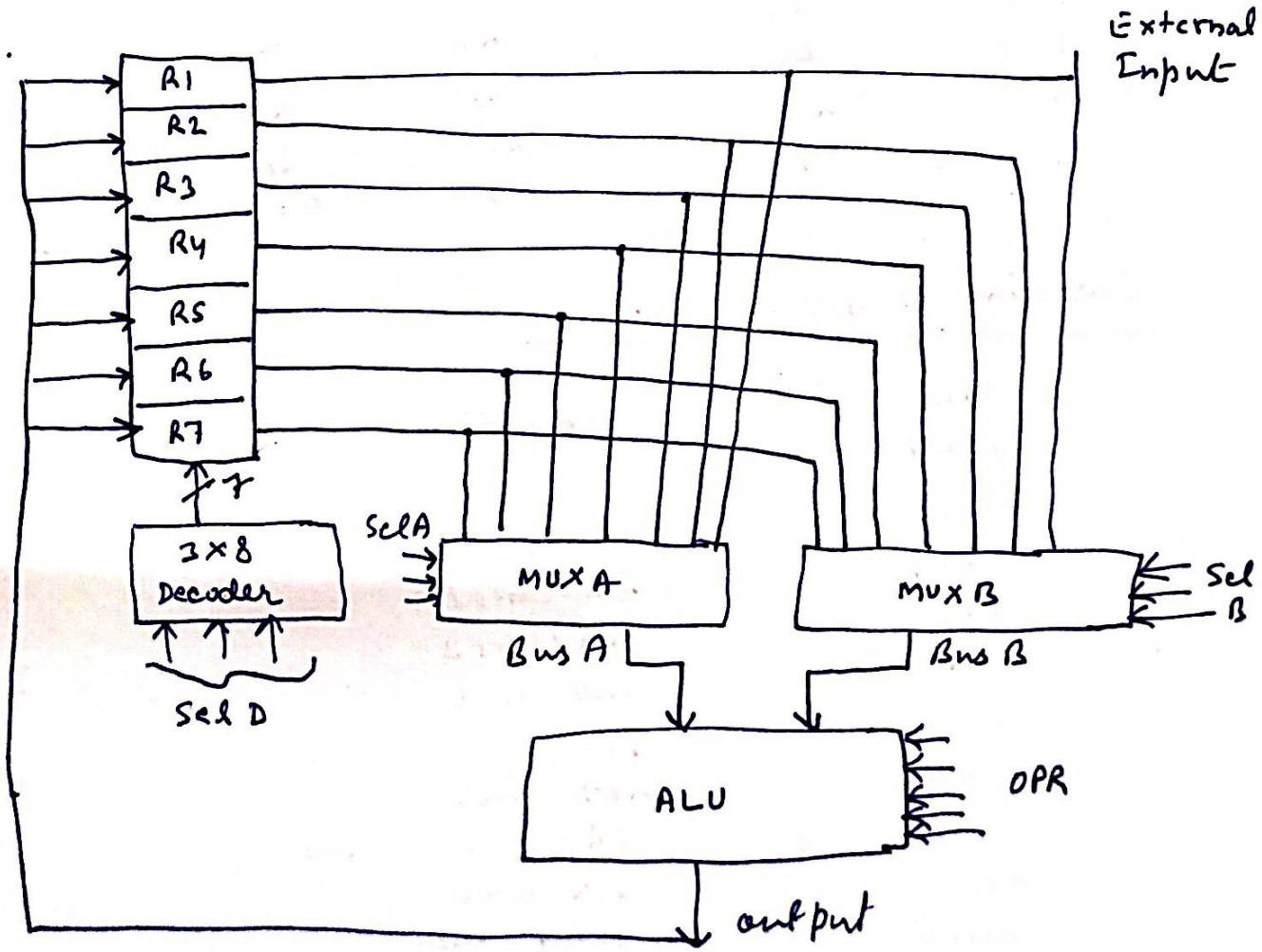
LECTURE-

Content : General Register organization and stack organisation

Organization & Architecture of CPU

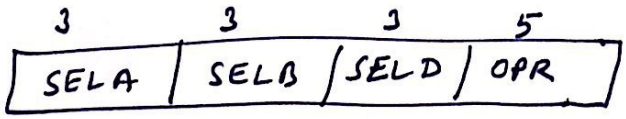
General Register Organization :

The register set stores intermediate data used during the execution of the instructions. If a large number of registers are included in processor unit, we need common bus system for the general register organization. In this organization, the registers communicate with each other not only for direct data transfers, but also while performing various microoperations.



Block diagram

Control word : It is a group of binary bits that are assigned to perform the specified operation.



Encoding of Register selection :

Binary code	SELA	SELB	SELD
000	External Input	Ext Input	—
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Encoding of ALU operations

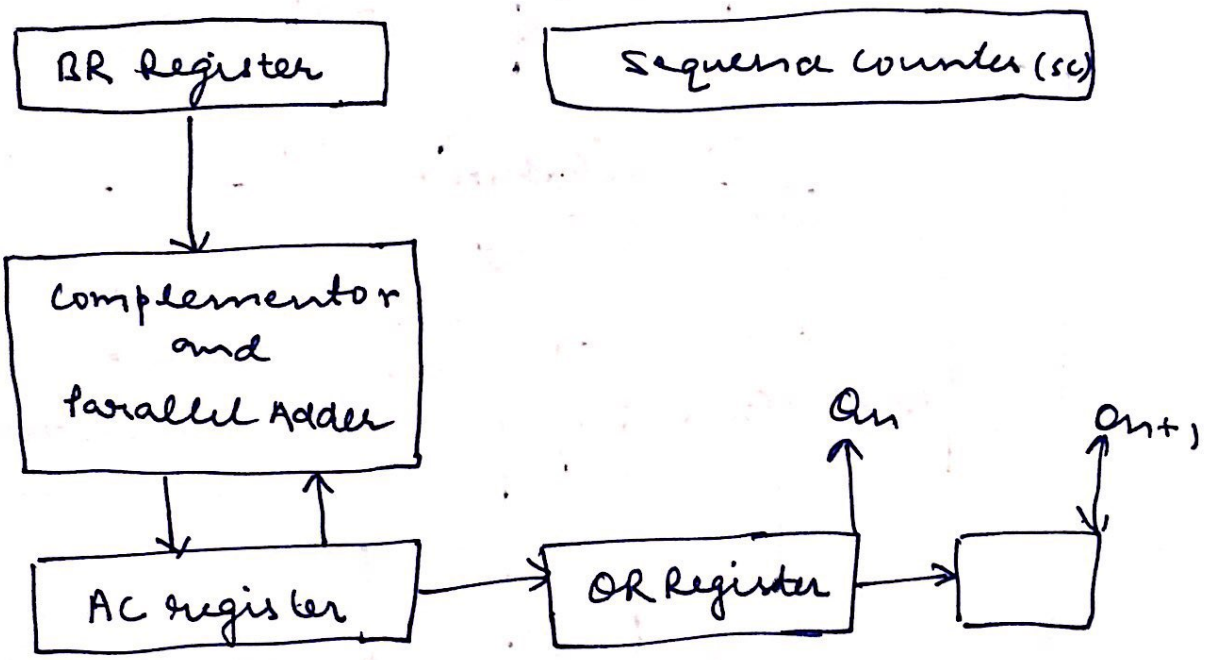
OPR Select-	operation
00000	Transfer A
00001	Increment A
00010	Add (A+B)
00101	Sub (A-B)
00110	Dec A
01000	AND (A&B)
01010	OR (A+B)
01100	XOR (A+B)
01110	~~Shift~~ Comp A
10000	Shift right A
11000	Shift left A

Lecture

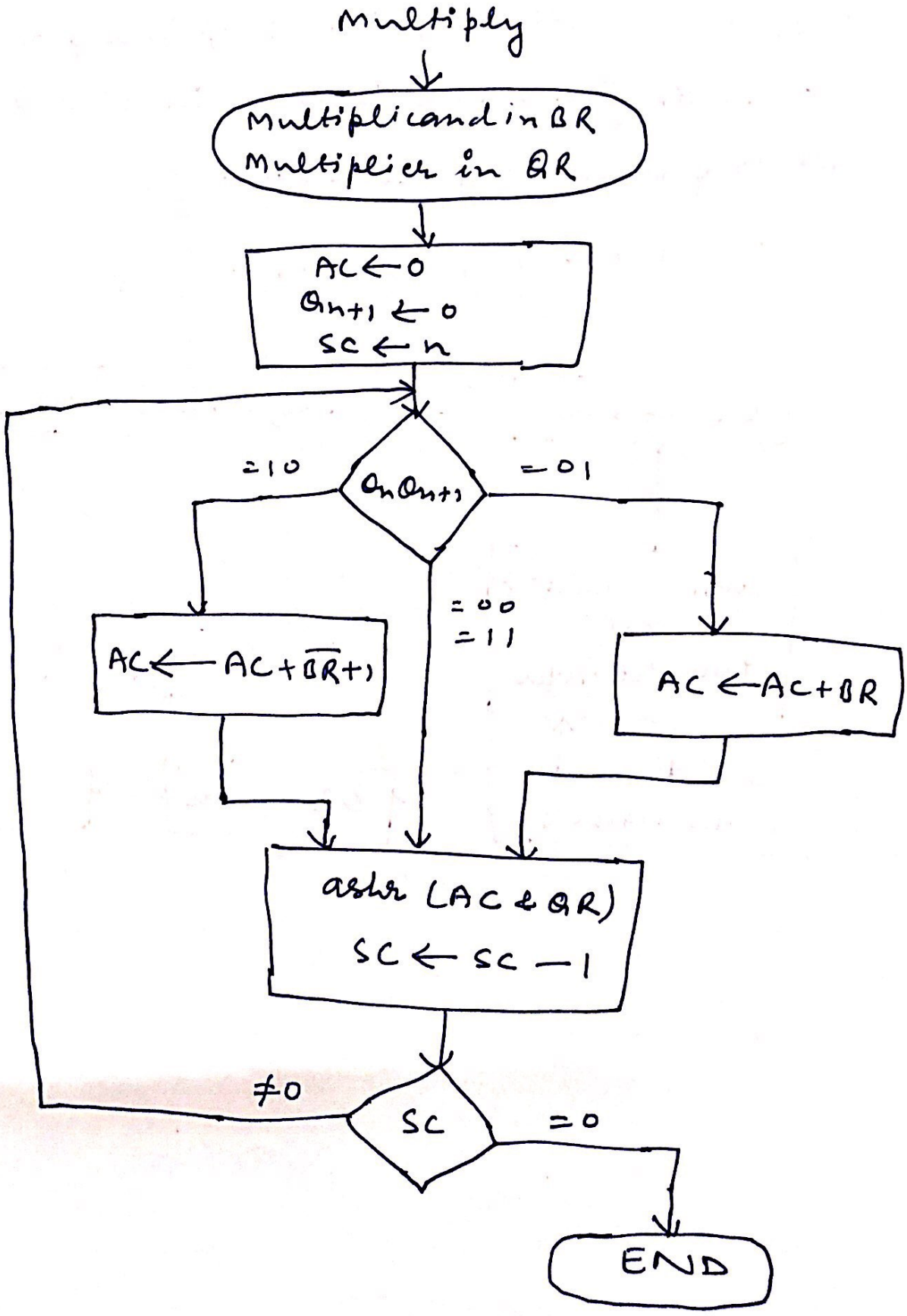
Content : Booth Algorithm

Booth Multiplication Algorithm : It is a procedure for multiplying binary integers in signed 2's complement representation.

Hardware for Booth Algorithm :



Booth algorithm for multiplication of signed 2's complement :



Case-1: 5X4:

	Binary	Binary with sign	2's complement
Multiplicand (BR) = 5	101	0101	
Multiplier (QR) = 4	100	0100	

Q_n	Q_{n+1}	BR = 0101 $\overline{BR}+1 = 1011$	AC	QR	Q_{n+1}	SC
	.	Initial	0000	0100	0	4
0	0	ashr	0000	0010	0	3
0	0	ashr	0000	0001	0	2
1	0	Subtract BR	1011			
			1011			
		ashr	1101	1000	1	1
0	1	add BR	0101			
			0010			
		ashr	0001	0100	0	0

result: 0001 0100 (20)

Case 2 : 5 × -4 :

	Binary	Binary with sign	2's complement
Multiplicand : BR = 5	101	0101	
Multiplier : QR = -4	100	0100	⇒ 1011 + 1 ⇒ 1100

Q_n	Q_{n+1}	BR = 0101 $\overline{BR}+1$ = 1011	AC	QR	Q_{n+1}	SC
		initial	0000	1100	0	4
0	0	ashr	0000	0110	0	3
0	0	ashr	0000	0011	0	2
1	0	Subtract BR	1011 1011			
		ashr	1101	1001	1	1
1	1	ashr	1110	1100	1	0

result : 11101100 (2's complement of 20)

Case 3 : -5 × 4

Carry look Ahead Adder :

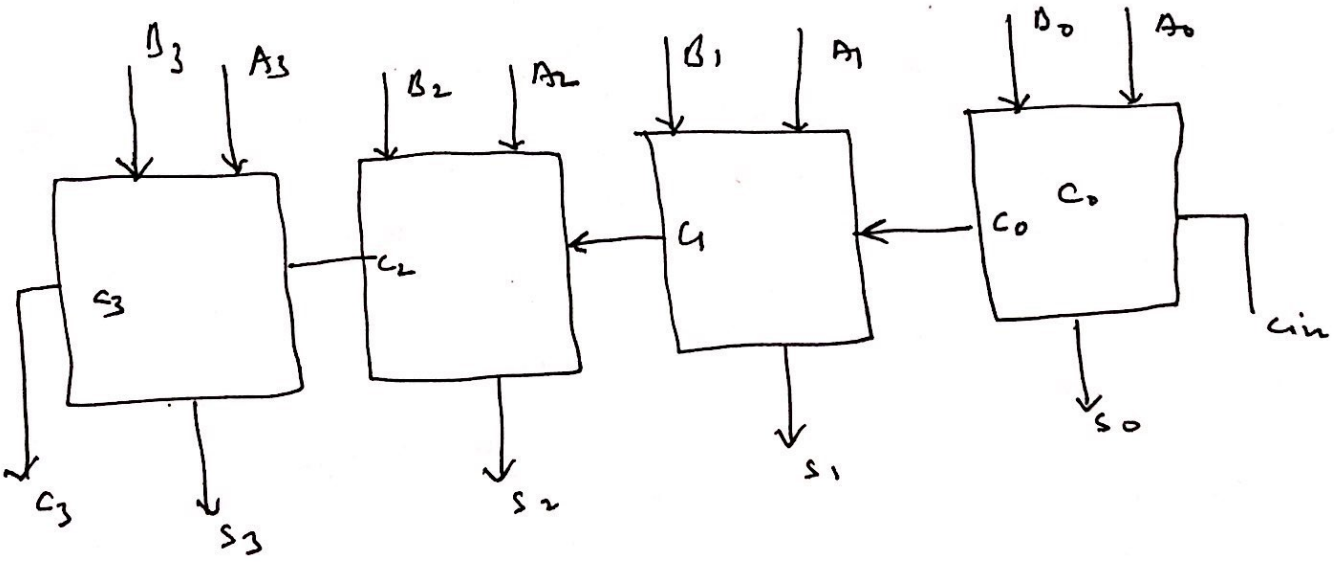
A	B	cin	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

carry generator — carry Propagate

$$C_o = A \cdot B + (A \oplus B) \cdot C_{in}$$

$$C_o = G + P C_{in}$$

$$C_i = G_i + P_i C_{i-1}$$



LECTURE

Content : Array Multiplier,

Array Multiplier : Checking the bits of the multiplier one at a time and performing partial products is a sequential operation that requires a sequence of add and shift micro operations.
The multiplication of two binary numbers can be done with one micro operation by means of a combinational circuit that forms the product bits all at once.

for J multiplier bits
K multiplicand bits
J x K AND gates
(J-1) K bit adders to produce a product of J+K bits

b_1 b_0

a_1 a_0

$a_0 b_1$ $a_0 b_0$

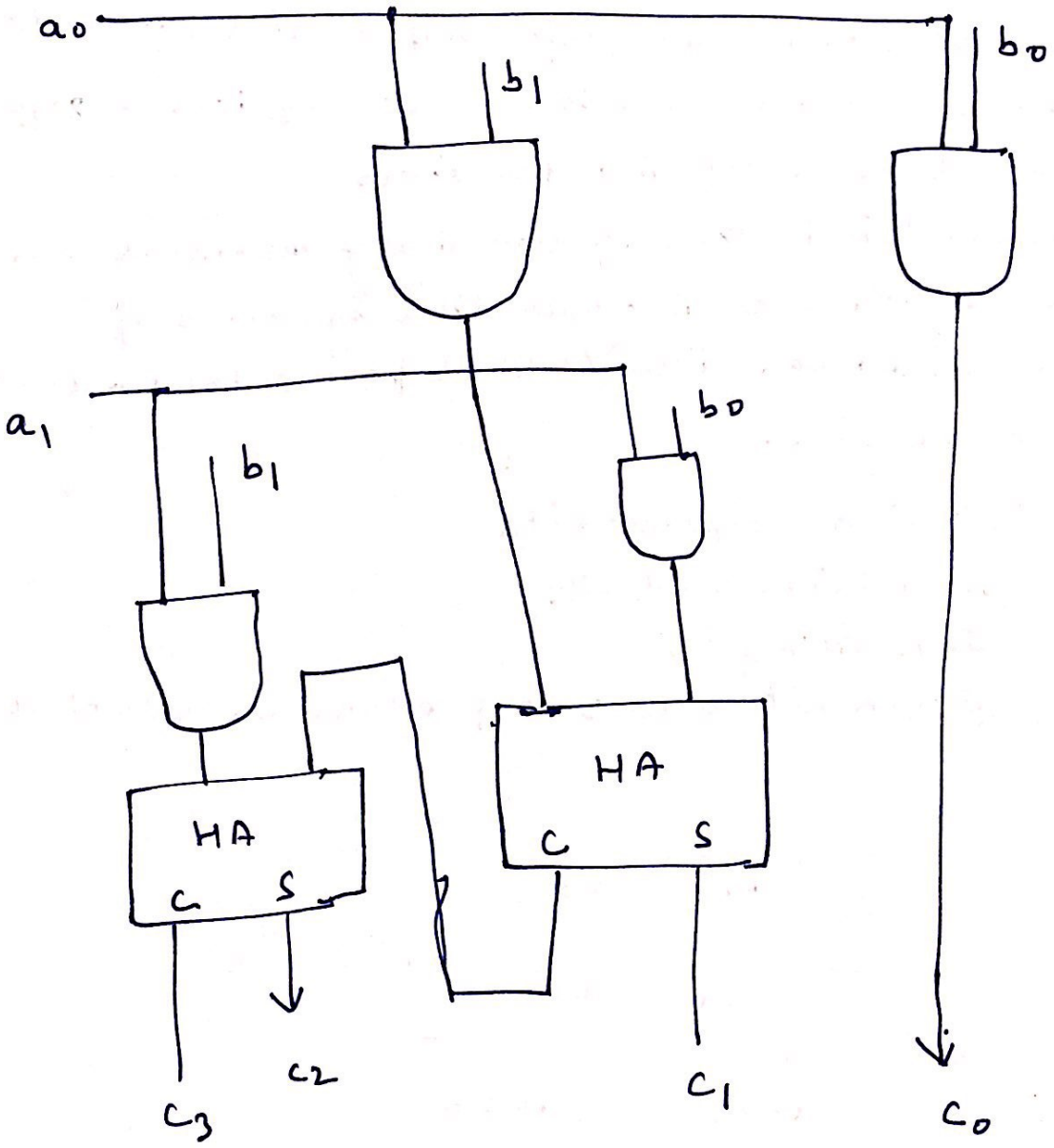
$a_1 b_1$ $a_1 b_0$

C_3 C_2 C_1 C_0

J=2 J*K means 2*2 = 4 gates

K=2 (J-1) K bit adders

(J-1) K bit adders to product a product of (J+K) bits



2-bit by 2-bit array Multiplier

Exp.

K=4

J=3

For Exp:

	SEL A	SEL B	SEL D	opr
R1 ← R2 + R3	010	011	001	00010
R6 ← R6 + 1	110	000	110	00001

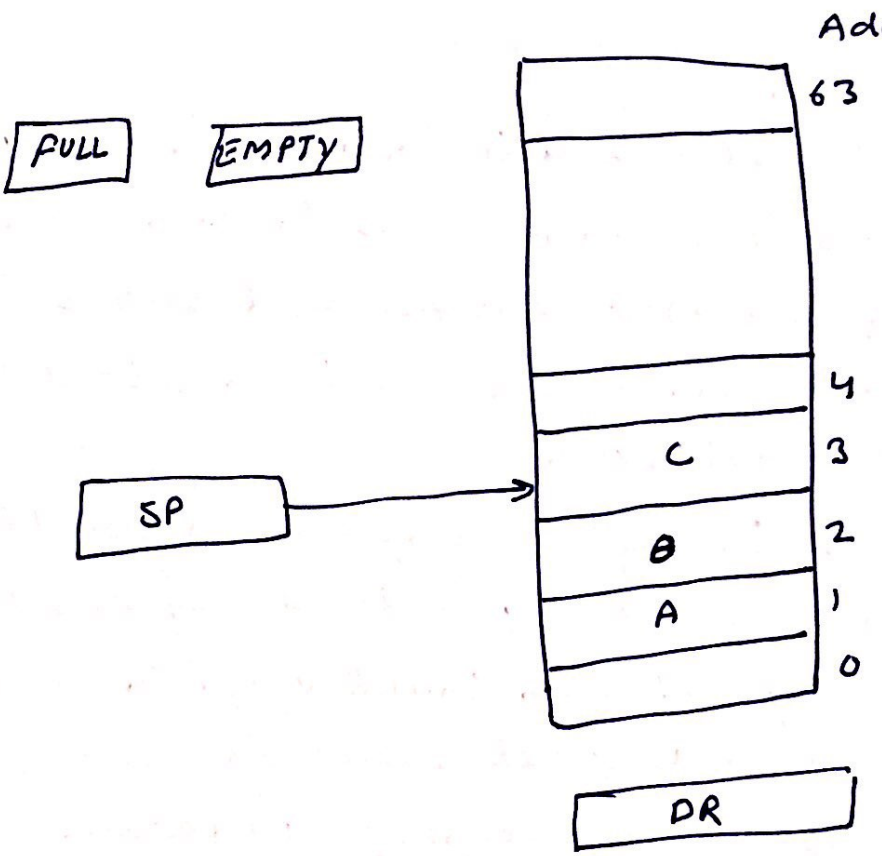
Stack Organization :

A useful feature that is included in the CPU of most computers is a stack or last in, first out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push because it can be thought of as the result of pushing a new item on top. The operation of deletion is called pop because it can be thought of as the result of removing one item so that the stack pops up. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer-register.

Register Stack : A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.



Block diagram of a 64-word stack

This figure shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack : A, B and C in that order. Item C is on top of the stack so that the content of SP is now 3.

Initially, SP is cleared to 0, EMPTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and stack is marked empty and not full.

Push:

SP ← SP+1 Increment stack Pointer

M[SP] ← DR write item on top of the stack

if (SP = 0) then (FULL ← 1) check if stack is FULL

EMPTY ← 0 Mark the stack not empty

Pop:

A new item is deleted from the stack if the stack is not empty.

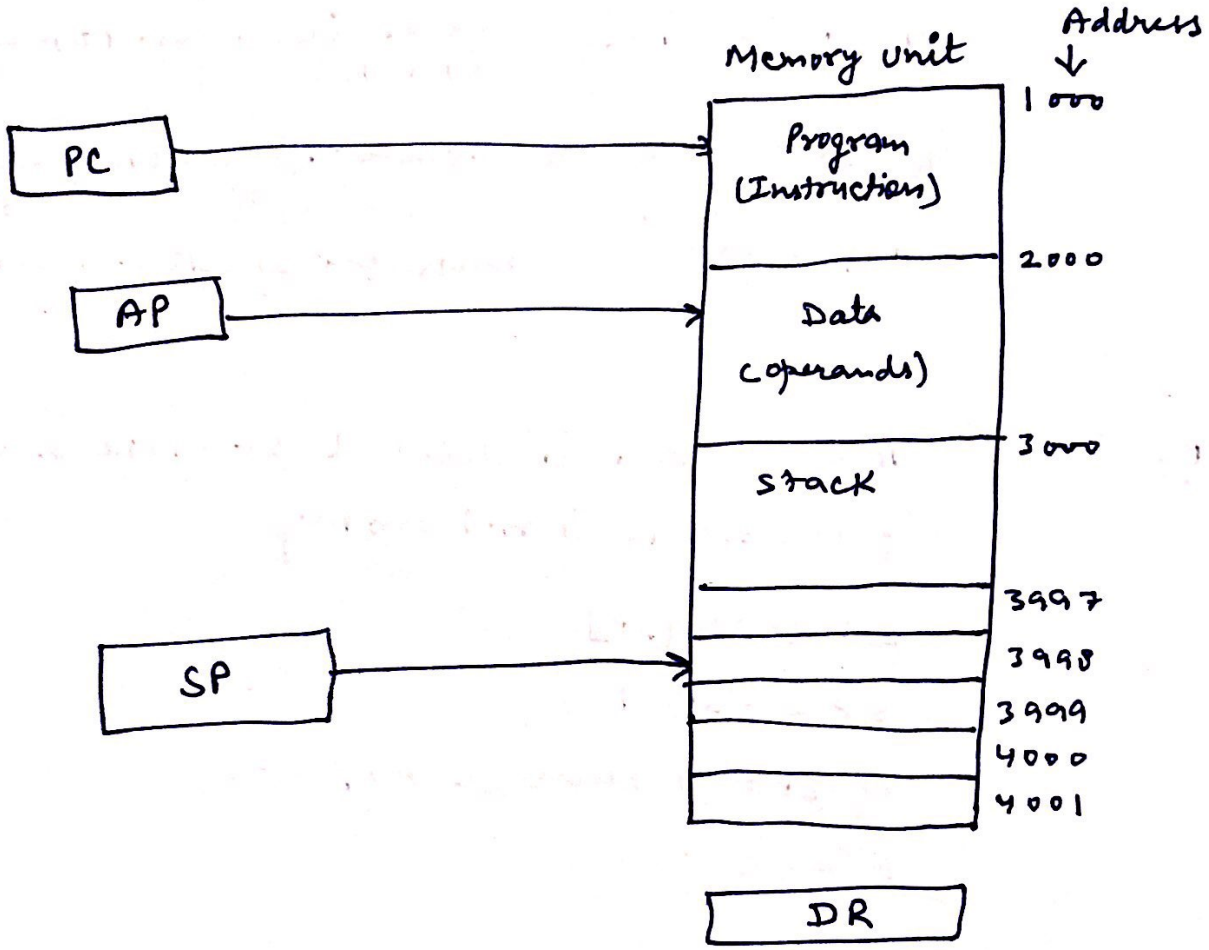
DR ← M[SP]

SP ← SP-1

if (SP = 0) then (EMPTY ← 1)

FULL ← 0

Memory Stack : A stack can exist as a stand-alone unit as in above figure or can be implemented of a stack In the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.



Computer memory with programs, data & stack segments

This figure shows a portion of computer memory partitioned into three segments: program, data and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data.

The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.

We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

push:

$$SP \leftarrow SP - 1$$

$$M[SP] \leftarrow DR$$

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

pop:

$$DR \leftarrow M[SP]$$

$$SP \leftarrow SP + 1$$

Reverse Polish Notation

A stack organization is very effective for evaluating arithmetic expressions.

A + B	Infix Notation
+ A B	Prefix or Polish Notation
A B +	Postfix or Reverse Polish Notation

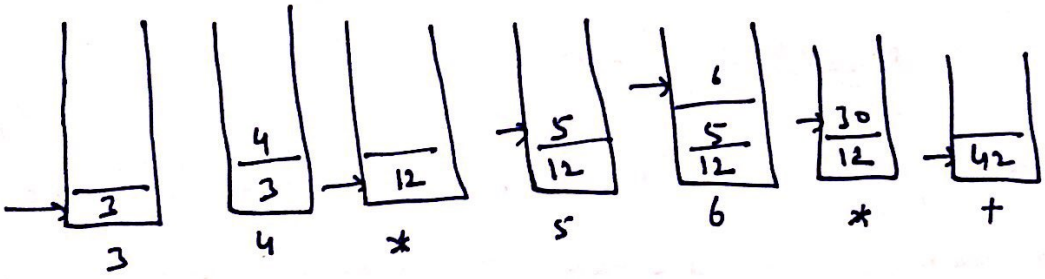
like A * B + C * D

RPN A B * C D * +

(3 * 4) + (5 * 6)

RPN 3 4 * + 5 6 * +

3 4 * 5 6 * +



Stack operation to evaluate (3 * 4) + (5 * 6)

Instruction Formats : The format of an instruction is usually depicted in a rectangular box symbolizing the bits of instruction as they appear in memory words or in control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction format are :

1- An operation code field that specifies the operation to be performed.

2- An address field that designates a memory address or a processor register.

3- A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three type of CPU organization.

1- Single Accumulator organization.
2- General register organization
3. Stack organization

Three-Address Instructions

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

```
ADD   R1, A, B    R1 ← M[A] + M[B]
ADD   R2, C, D    R2 ← M[C] + M[D]
MUL   X, R1, R2   M[X] ← R1 * R2
```

Two-Address Instructions

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

```
MOV   R1, A       R1 ← M[A]
ADD   R1, B       R1 ← R1 + M[B]
MOV   R2, C       R2 ← M[C]
ADD   R2, D       R2 ← R2 + M[D]
MUL   R1, R2      R1 ← R1 * R2
MOV   X, R1       M[X] ← R1
```

One-Address Instruction (Single Accumulator organization)

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

```
LOAD   A    AC ← M[A]
ADD    B    AC ← AC + M[B]
STORE  T    M[T] ← AC
LOAD   C    AC ← M[C]
ADD    D    AC ← AC + M[D]
MUL    T    AC ← AC * M[T]
STORE  X    M[X] ← AC
```

Zero-Address Instructions (Stack organization)

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

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

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M[X] ← TOS

LECTURE

Content : Addressing Modes & Look Ahead Carry Adders

Addressing Modes : The Technique of specifying the address of the data is known as addressing modes.

or

The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced

Advantage of having various addressing modes

- To give programming versatility to the user
- To reduce the number of bits in the addressing field of the instruction.

- **Implied Mode** : In this mode the operand are specified implicitly in the definition of the instruction. Instruction related to accumulator.

  Ex: RAL : Rotate left the content of accumulator left

  CMA : complement the content of accumulator.

- **Immediate Mode** : In this the operand is given in the instruction itself.

  Ex: MVI A, 06 :- Move 06 to accumulator

  ADI 05 : Add 05 to content of accumulator

- Register Addressing mode: In this the operand are in registers that reside within the CPU.

Ex: ADD B : Add the content of Register B with the content of accumulator.

MOV A,B : Transfer the content of register B to accumulator.

- Register Indirect addressing mode: In this the instruction specifies a register in the CPU, whose contents give the address of the operand in the memory.

Ex: LXI X, 2400H : Load H-L with 2400H.

MOV A,M : Move the content of memory location, whose address is in H-L pair (i.e 2400H) to the accumulator.

LXI H, 2100H : Load HL with 2100H.

ADD M :

Auto Increment mode: It is similar to the register indirect mode except the register is incremented after its value is used to access memory.

- Auto decrement Mode : It is similar to the register Indirect mode except that the register is decremented before its value is used to access memory.

Direct Address Mode : In this the effective address is equal to the address part of the instruction.

Indirect Address Mode : In this the address field of the instruction gives the address where the effective address is stored in the memory

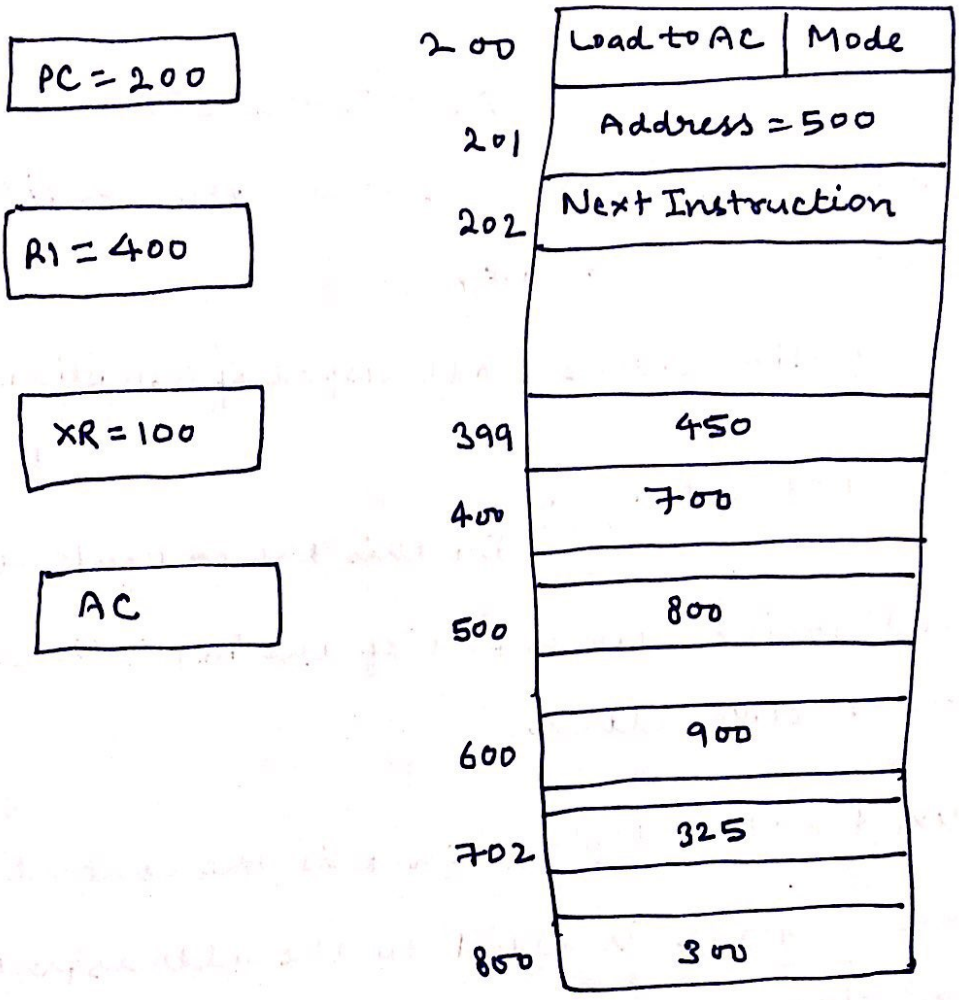
effective address = address part of Instruction + content of CPU register

Relative Address Mode : In this the content of PC is added to the address part of the instruction to obtain the effective address.

Indexed Address Mode : In this the content of the Index register is added to the address part of the instruction to obtain the effective address.

Base Register Addressing Mode : In this the content of base register is added to the address part of instruction to obtain the effective register.

Example of Addressing Mode



Addressing Mode	Effective Address	Content of AC
Direct Address	500	800
Immediate Operand	201	500
Indirect Address	800	300
Relative Address	702 (500+202)	325
Indexed Address	600 (XR+500)	900
Register	—	400
Register Indirect	400	700
Autoincrement	400	700
Autodecrement	299	450