UNIT-1 -

Date

① # STRUCTURE OF DESKTOP COMPUTER

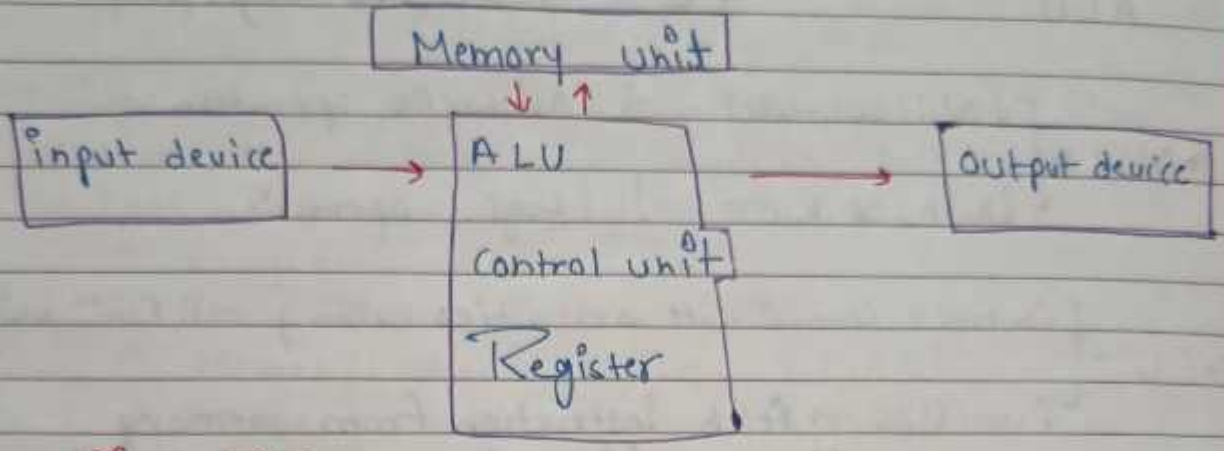


Fig- CPU

memory → Primary memory [RAM]
→ Secondary memory

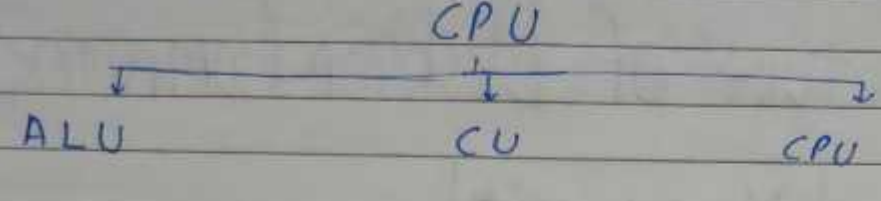
→ Primary memory are fast, small & expensive. They store program & active data. Temperory Storage.

→ Secondary memory are large, slow & cheap.

eg- Magnetic tape, Magnetic disks.

CPU
- Brain of Computer
- It controls all internal & external device
- Controls the memory usage
- Controls the sequence of operation.
- It perform arithmetic & logical operation.

Date _____



- ALU - [Odd, Substract ...] Arithmetic operation
  [AND, OR, NOT ...] Logic operation
- CU - Control & Co-ordinate activities among all Fun. unit
  Function → fetch instruction from memory
  decode [identify the operation]
  Execute instruction
- CPU Register - It are generally of 2 type.
  a) Special Purpose Register
  b) General " "
- Register - A group of Flip-Flop used to store data. (a word) {1 FlipFlop can store 1 bit of data (0/1)}
  → It is high speed temporary storage space for holding data addresses & instructions during Processing the instruction.
  → Unlike memory, registers are directly accessed by name not by address.
  → Registers may be 8/16/32/64/128 bit depending upon its use.

Date

→ To perform execution of instruction CPU contains number of registers called as special purpose registers.

→ Special purpose registers is used to store data or address at a time but not both simultaneously.

* Program Counter (PC)
* Instruction Register (IR)
* Memory address register (MAR)
* Memory data register (MDR)
* Accumulator.

① Program Counter - It is used to store address of next instruction to be executed.

② Instruction Register - It is used to hold the instruction that is currently being executed.

→ Control unit access the content of IR to send the control signals to various processing elements to execute the given instruction.

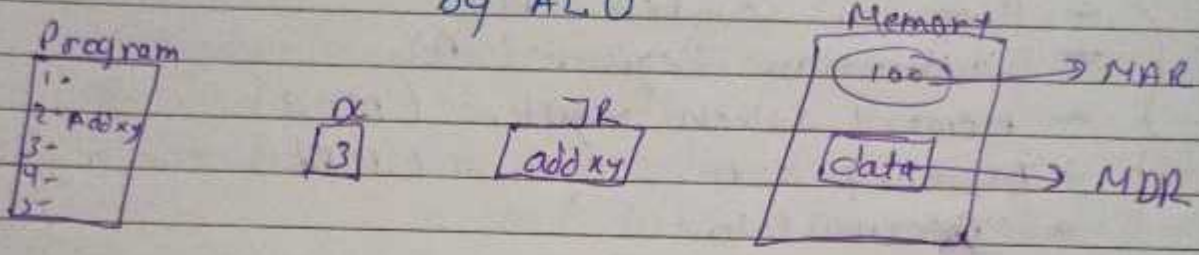
③ Memory address register [MAR] - The MAR holds the address of main memory to or from which data is to be transferred.

Date

④ Memory data Register (MDR) - Also known as memory buffer register.

* It contains the data to be written into or read from the addressed word of the MM.

⑤ Accumulator - It holds the result generated by ALU



b) General purpose Registers

→ Access time of registers is lowest hence they all used to store frequently used data.

→ GPR are used to hold operand for arithmetic & logic operations & / or used to store the result of the operation & memory location address.

[data / add]

→ It is multipurpose registers.

→ It may be used by programmer or user.

→ CPU has 8 general purpose registers each capable of storing 32-digit binary numbers (32-bits).

Date

→ In addition 32-bit data, they can also store 16 or 8-bit data.

→ Registers described in instruction as -

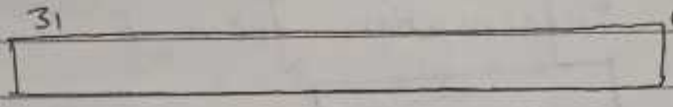
Store Instruction

32bit data → ER_0 ER_1 ER_2 ER_3 ER_4 ER_5 ER_5 ER_6 ER_7

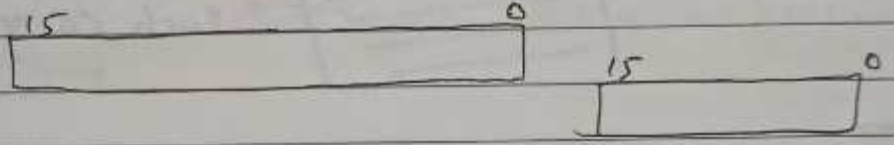
16 bit data → $E_0 - E_7$, $R_0 - R_7$.

8 bit data → R_0H, R_0L, R_1H, R_1L, ---, R_7H, R_7L.

→ ERn



→ En
Rn



→ RnH
RnL

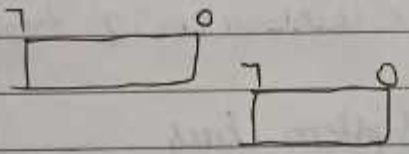
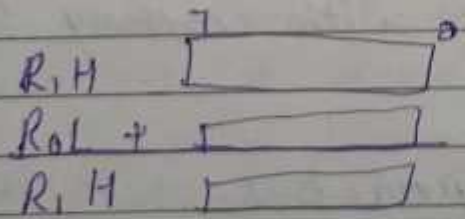


fig - General Purpose Registers.

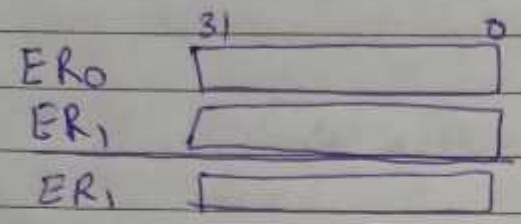
eg: ① ADD.B R_0L, R_1H (8 bit addition)
B → byte (8 bits)



② ADD.W R_0, E_1 (16 bit addition) W → word. (16 bit)

Date

③ ADD.L ER0, ER1 (32 bit addition)
(L → Long word)
32bit.



② # BUS

In computer system, bus is used to connect 2 or more devices.
It can send information from 1 device to another.


→ Each carry 1-bit data.

③ Types of Bus - There are 2 types of bus.

① Internal Bus / system bus
② External Bus / Peripheral bus / Expansion bus

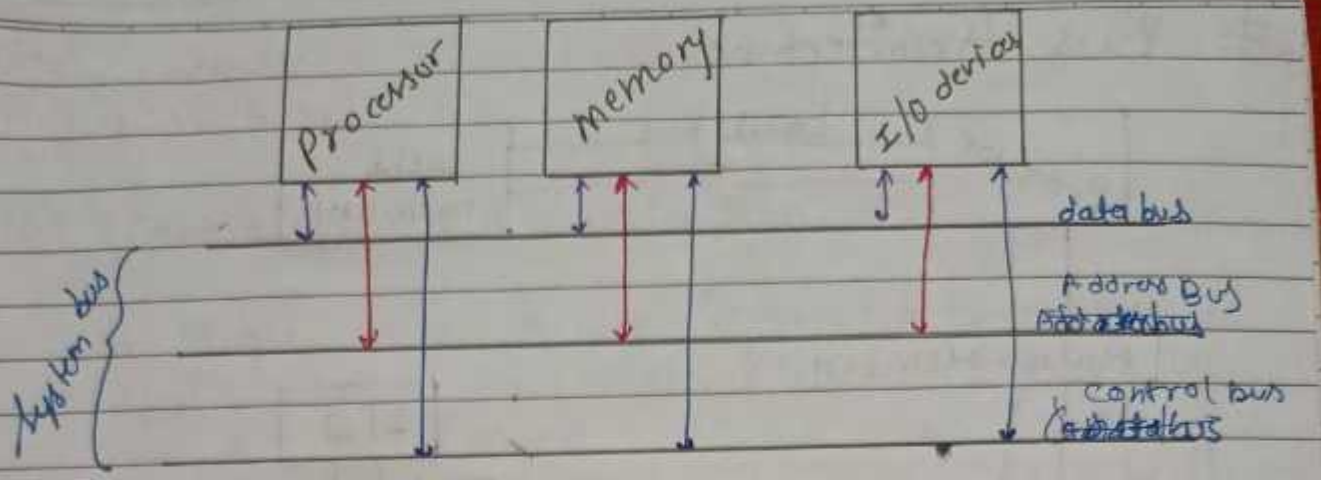
↓ System bus / internal bus

- These buses connect processor with memory input & output devices.

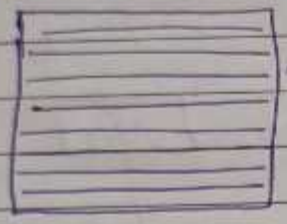
- There are 3 types of System bus.

(a) Data bus - bi - directional
(b) Address bus - Uni - "
(c) Control bus - bi - "

Date



→ Databus sends data. It is of different sizes, like 8 bit, 16 bit, 32 bit, 64 bits etc.



→ 8 bit data bus, contains 8 wire each carring 1-bit data.

→ Address bus takes address. It can be of different sizes.

→ Control bus takes control signals.

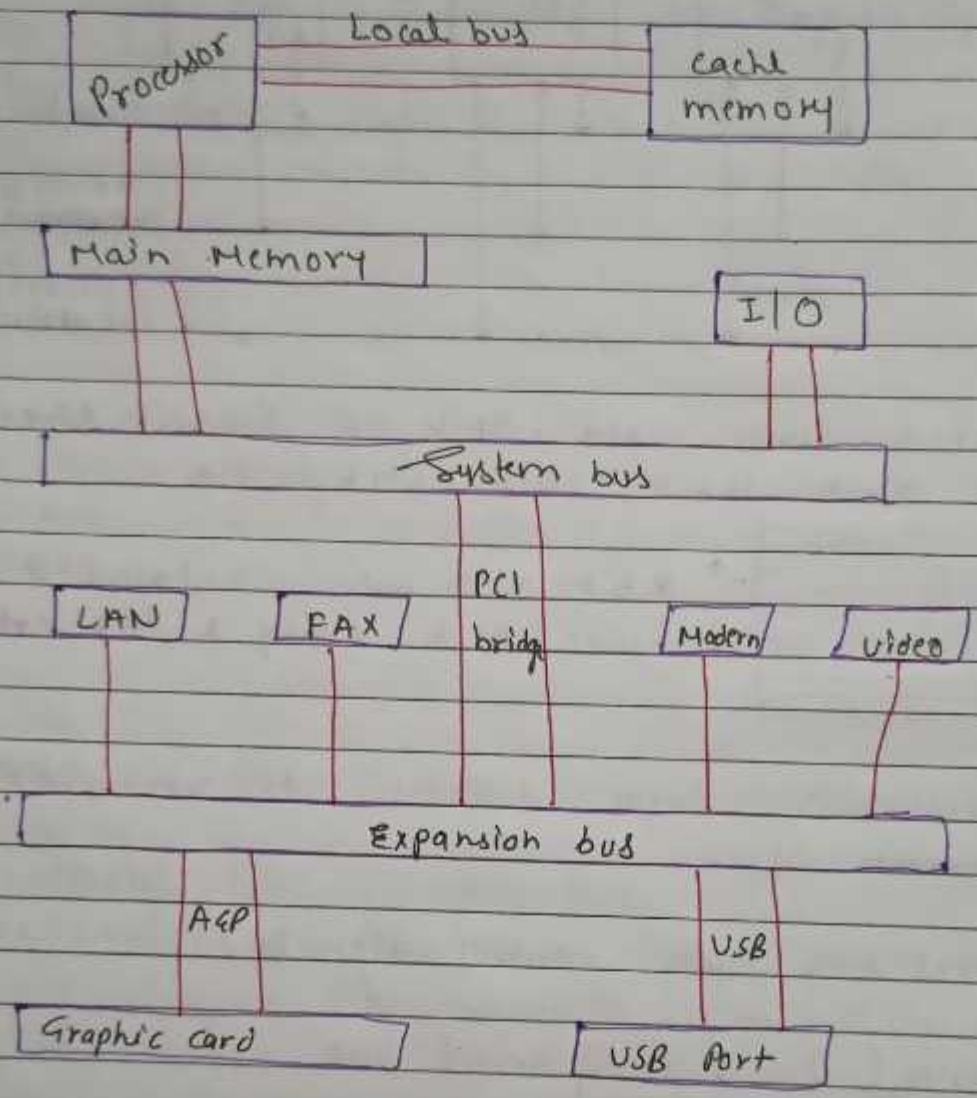
3) External bus / Peripheral bus

- These buses connect external device with processor & memory.

EISB → Extended Industry Standard Architecture.
MCI → Media Control Interface
SCSI → Small computer system Interface
PCI → Peripherial Controller interconnection
AGP → Accelerated Graphics Port [Grey in Color]
PCI-X → Peripheral controller interconnection express
USB → Universal Serial bus.

Date

Bus Architecture



Bus Arbitration -

* Bus arbitration is a process which devices requesting to use the system bus, which one to grant access of bus.

→ There are 2 types of bus arbitration.

Date

(a) Centralized bus arbitration
(b) Distributed " "

(a) Centralized bus arbitration

- A single hlw device (CPU DMA controller) is used to acts as arbiter (selector) to decide among a number of devices that which one will use the bus firstly.

It is of three types -
i) Daisy chaining arbitration
ii) Polling
iii) Independent request

(b) Distributed bus arbitration -

- Every device is involved is the selection of which one will use the bus firstly.

(i) * Daisy chaining method -

- In this method, the device which is closer to the bus controller / arbiter will be allowed to use the bus first.

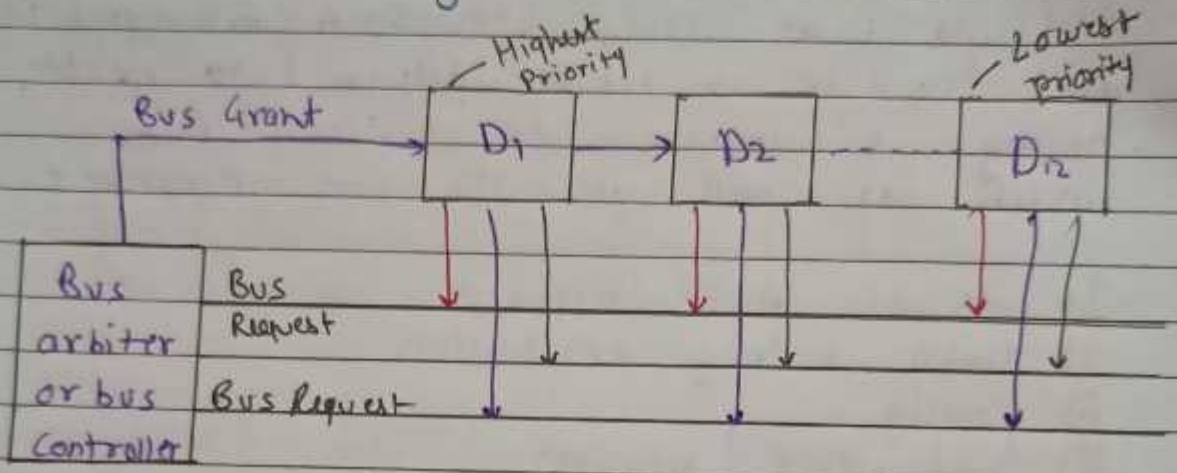
3 control signals are used -

(a) Bus Request :- It is used by devices to make a request for bus.

Date

(ii) Bus Busy :- It is used to show that bus already used by other devices.

(iii) Bus GRANT - It is used to allow a device to grant access of bus.



- when any device need the bus or want to become a bus master it sends the bus request to bus arbiter.
- Bus arbiter sends the bus grant & send bus busy line to all other devices.

* Bus Master - A device that initiate bus transfer on the bus at any given time is called bus master.

- Bus grant firstly goes to device 1. If device 1 sends all its data then it release it to the next device.

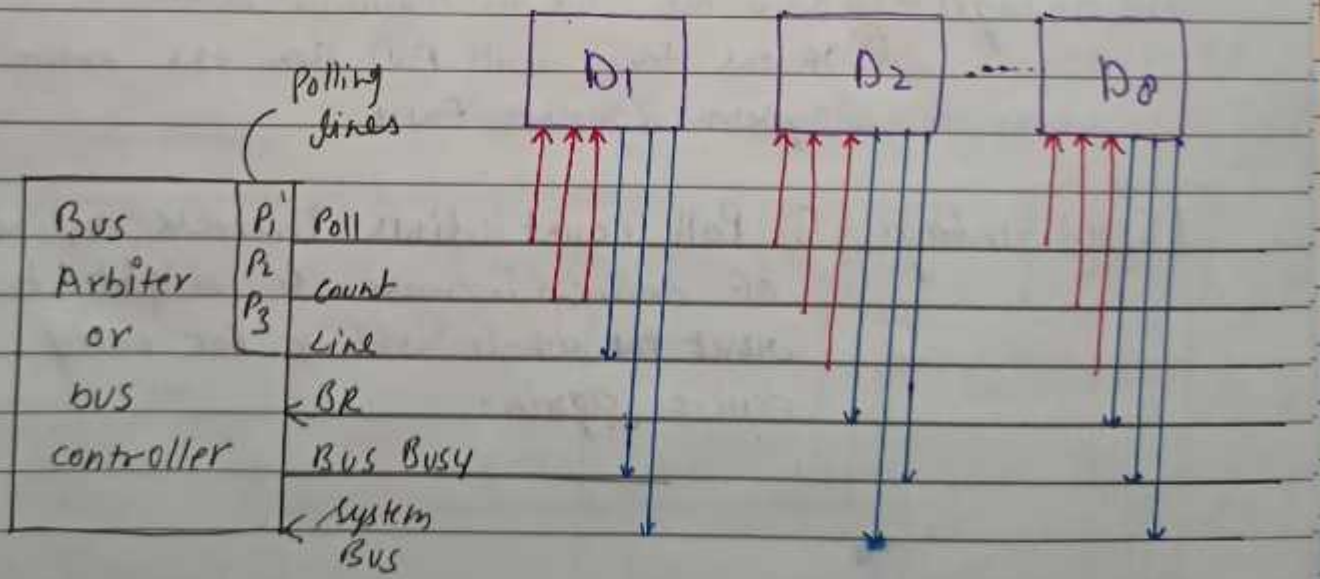
Date

Advantages - * It has simple circuit.
* Low cost due to simple circuit.
* we can attach or remove anything.

Disadvantages - * It has simple circuit Low speed
* It is not reliable because if one device will fail the entire system will fail.
* Low Priority device will have wait for long time if always high priority device will have data to send.

POLLING METHOD

- In this method, a set of Poll or Polling count lines are used to select a device from a number of devices requesting for a bus.



Here, No. of devices = 8
" " Poll lines = 2 = 3

Date

Poll lines = Poll address = 3

And,

Address lines = 3 i.e. 3bit address Then address will be 000, 001, 010, 011, 100, 101, 110, 111 for device 1 to Device 8 respectively.

→ The bus arbiter based on the priority of devices. allow the device having highest priority among the requested devices.

→ In this method a num of device make a request for bus using "Bus request" line.

→ Bus arbiter first check "Bus Busy line & if it is free then bus arbiter active "Poll count lines" based on priority & grant permission to acquire the system bus.

Advantages - ① Low cost due to common bus.
② If one device will fail then the entire system doesn't fail.

Disadvantages - ① Poll count lines increase as no. of devices increase & might be reset the whole address for every device again.

Date

Independent request Bus arbitration

- In this method each device make a bus request using seperate "Bus request control lines".
- Bus arbiter grant bus access to a device using seperate "Bus grant control line.
- All devices uses common "Bus Busy Control line".
- The bus arbiter allocates the bus to the device having "highest priority among a number of requesting devices.

Advantages- (1) Adding & removing devices is easy.
(2) Fast arbitration due to independent bus request & bus grant lines.

Disadvantages- (1) No. of control lines increases.
(2) High cost due to number of control lines increased.

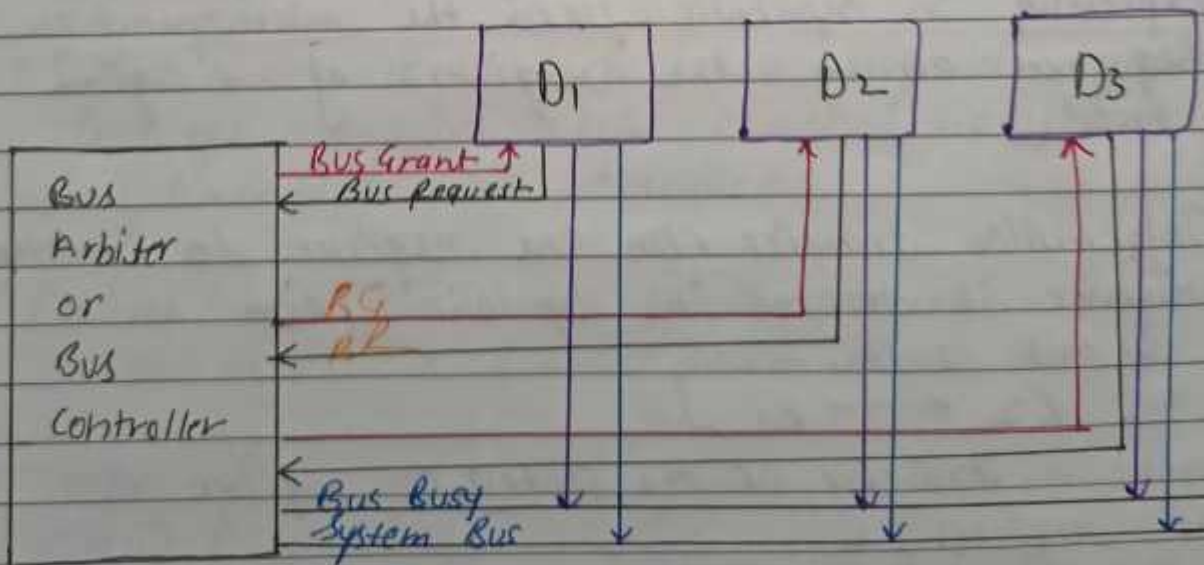


fig - Independent request for Bus arbitration.

→ Register
→ Arithmetic
→ Logic
→ Shift

Date

Micro-Operations

* A micro-operation is an elementary operation performed with the data stored in registers.

* In digital computer, thus are four types of micro operations are performed -

a) Register Transfer / Register Transfer micro-operation / Register transfer language.
b) Arithmetic microoperations
c) Logic micropperations
d) shift "

(a) Register Transfer Micro-operation

- The Symbolic notation used to describe the microoperation transfer among registers is called a Register transfer language (RTL)

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

- Information transfer from one register to another register is denoted in symbolic form as shown below -

$[R2 \leftarrow R1]$

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

Date _____

- In digital computer the transfer occur only under a pre-determined control condition. In RTL this can be shown by means of if-then statement.

if (P = 1) then ($R_2 \leftarrow R_1$)

where P is a control signal generated in control section.

→ It is sometimes converted to separate the control variable from the register transfer operation by specifying a control function.

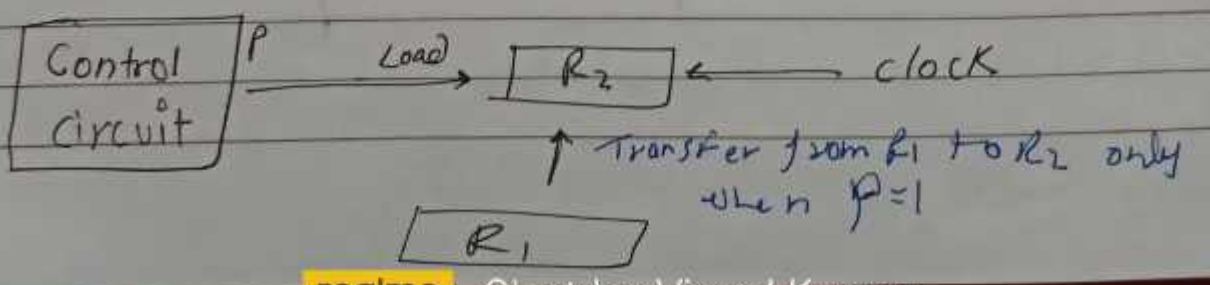
→ A control function is a boolean variable that is equal to 1 or 0. The control function can be included in the above statement as

$P : R_2 \leftarrow R_1$

Here, the control condition is terminated with a colon, the transfer operation executed by the hlw only if P=1.

→ Every statement written in a register transfer language implies a hlw construction for implementing the transfer.

→ Figure below shows the block diagram that shows the above transfer from R_1 to R_2.



Date

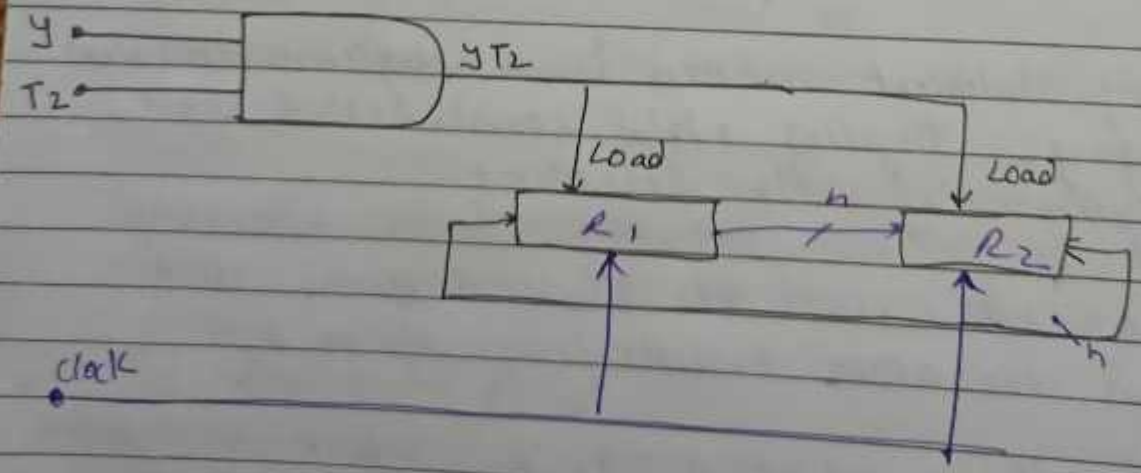
- The n O/p of register R_1 are connected to the n input of register R_2.
- Register R_2 has a load i/p that is activated by the control variable P.
- A Common is used to seperate 2 or more operation that are executed at the same time the Statement.

$$T: R_2 \leftarrow R_1, R_1 \leftarrow R_2$$

denotes on operation that exchange the content of 2 register during 1 common clock when $T = 1$.

Example – Show the block diagram of the hlw the implements the following register transfer statement.

$$yT_2: R_2 \leftarrow R_1, R_1 \leftarrow R_2$$



Date

(1)

Q- Represent the following conditional control statement by 2 register transfer statement with control function.

if (P = 1) then ($R_1 \leftarrow R_2$) else if (Q = 1) then ($R_1 \leftarrow R_3$)

The above 2 register transfer statements with control function can be shown as below -

$P : R_1 \leftarrow R_2$

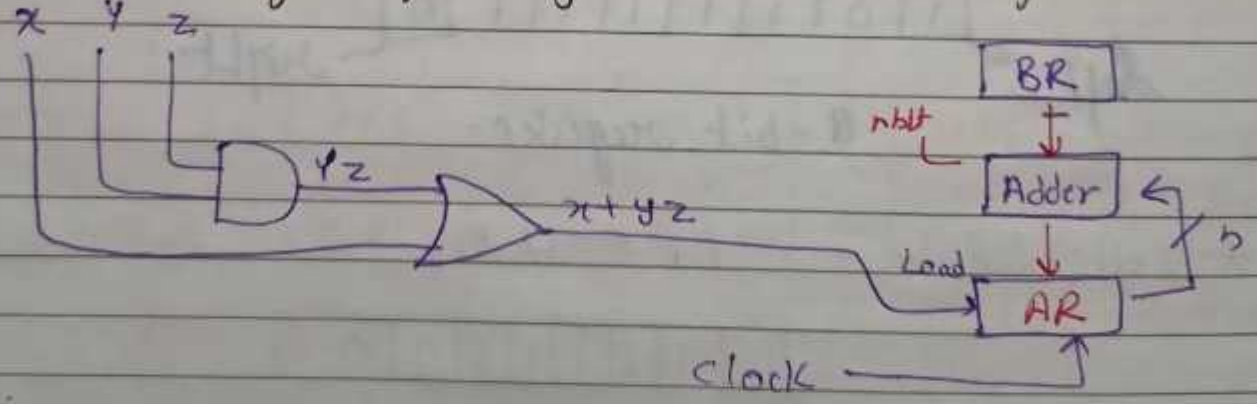
$P'Q : R_1 \leftarrow R_3$

(2)

Q- Draw the block diagram for the hlw that impliment the following statements -

$x + yz : AR \leftarrow AR + BR$

where AR & BR are two n bit register & x, y & z are control variables include the logic gates for the control function.



Date

2) Arithmetic microoperation

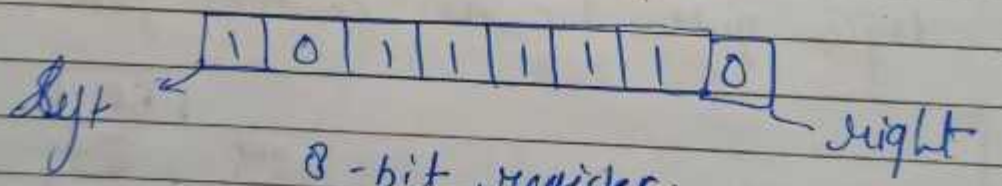
- Addition ($R_1 \leftarrow R_1 + R_3$)
- Subtraction ($R_1 \leftarrow R_1 - R_2$)
- Increment ($R_1 \leftarrow R_1 + 1$)
- Decrement ($R_1 \leftarrow R_1 - 1$)

3) Logic microoperation -

- AND ($R_1 \leftarrow R_1 \wedge R_2$)
- OR ($R_1 \leftarrow R_1 \vee R_3$)
- NOT ($R_1 \leftarrow \overline{R_1}$)
- NAND ($R_1 \leftarrow \overline{R_1 \wedge R_2}$)
- XOR ($R_1 \leftarrow R_1 \oplus R_2$)

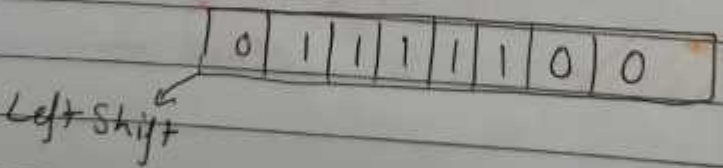
4) Shift micro-operation

In shift micro-operation we shift the register bits in serial order.

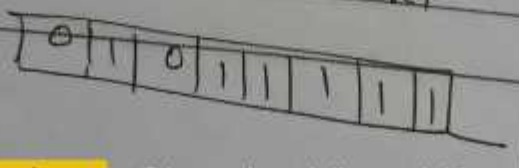


8-bit register.

* Left shift - $R_1 \leftarrow shl R_1$



* Right shift $R_1 \leftarrow shr R_1$



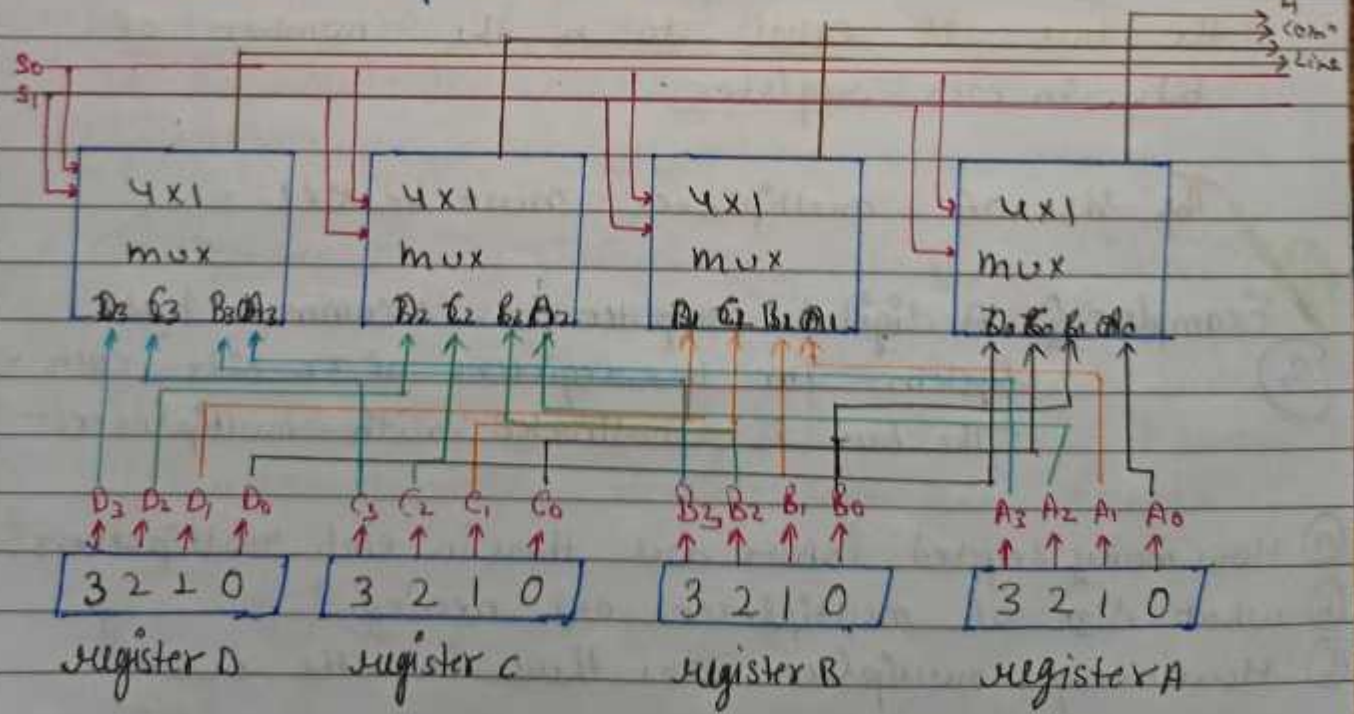
(11) Bus Transfer

Date _____

A bus transfer consist of a set of common lines 1 for each bit of a gr register through which binary information is transferred 1 at a time. control signals determine which register is selected by the bus during each particular register transfer.

One way of constructing a common bus system is with mutiplexers.
The multiplexers select one source register whose binary information is then placed on the bus. The connection of a bus system for four registers is shown below.

Each register has 4 bits, numbered 0 through 3. The Bus consist of 4x1 multiplexer each having 4 data i/p 0 through 3, & 2 Selection inputs S_1 & S_0.



Bus System for four Registers

Date

The selection lines choose the four bits of one register & transfer them into the four-line common bus.

S_0	S_1	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 multiplexer are selected & applie to the outputs that form the bus. This causes the bus line to receive to content of register A.

Note - In general, a bus system will multiplex K registers of n-bit each to produce an n-line common bus.

The number of multiplexers need to construct the bus is equal to n the number of bits in each register.

The size of multiplexer must be $K \times 1$.

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

(a) How many selected inputs are there in each multiplexers?
(b) what size of multiplexer are needed?
(c) How many multiplexer are there in the bus?

(a) Num. of multiplexer = No of bits each Register 32
(b) Size of " = No of Register. 16

Date

Solve- Digital computer has 16 register of 32 bits each i.e. num. of bits in each register = 32 bits.

(a) The num. of multiplexer needed to construct the bus = number of bits in each register.

∴ Number of mux = 32

(b) No. of register K = 16
Size of multiplexer = 16X1

(c) Since size of multiplexer = 16X1
$= 2^4 X 1$
So there are 4 selection inputs in each multiplexer.

12

Memory Transfer

- The transfer of information from a memory word to the outside environment is called read operation.
- The transfer of new information to be stored into the memory is called a write operation.
- It is necessary to specify the address of memory (M) when memory transfer operations occurs.

Date

(4)

Example - A memory unit that receives the address from a register called address register (AR). The data are transfered to another register called the data register (DR).

⇒ The read operation can be stated as Read :

DR ← M[AR]

⇒ The write operation can be stated as write: M[AR] ← R_1

because the transfer of R_1 register into the memory

(5)

Q The following transfer statement specify a memory explain the memory operation in each case :

(a) R_2 ← M[AR]

(b) M[AR] ← R_3

(c) R_5 ← M[R_5]

(a) R_2 ← M[AR]

⇒ Read memory word specified by the address in AR in to register R_2.

(b) M[AR] ← R_3

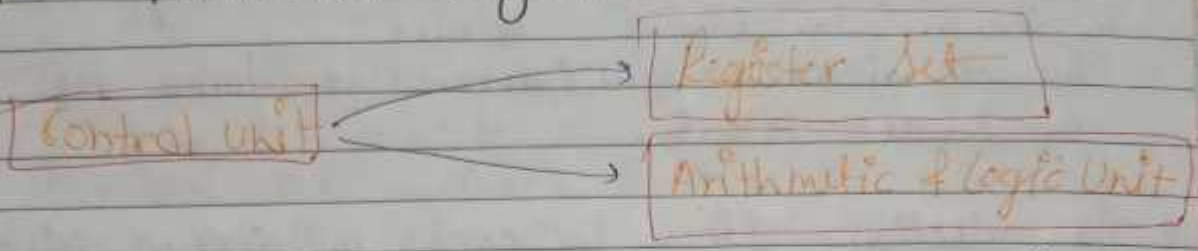
write content of register R_3 into the memory word specified by the address in AR.

(c) R_5 ← M[R_5]

Read memory word specified by the address in R_5 & transfer content to R_5.

Date

Processor Organization



a) The design of a CPU is a task that in large part involves choosing the HW for implementing the machine instruction.

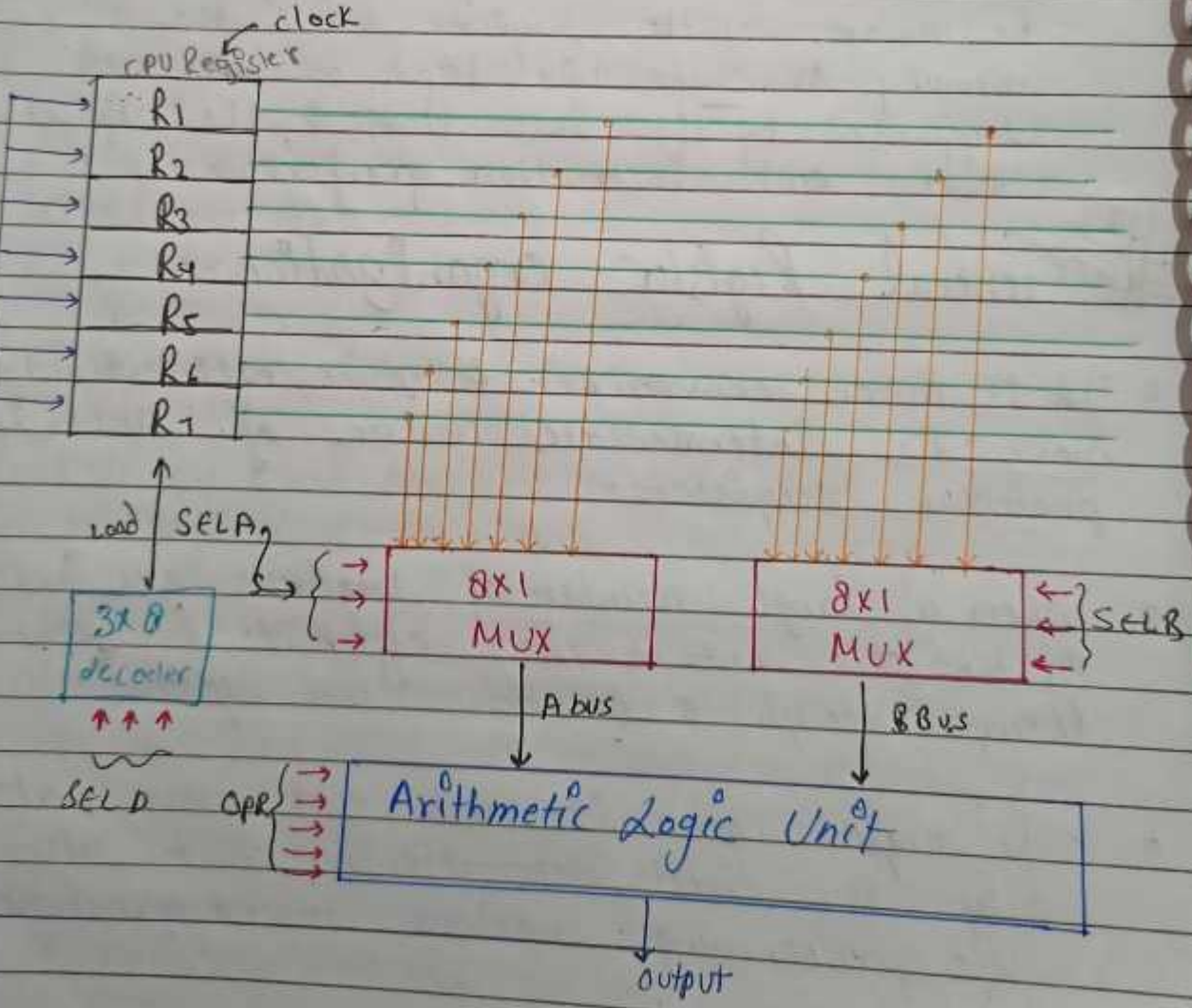
b) The user who programs the computer in machine or assembly language must be aware of the register set the memory structure the type of data supported by the instruction & the function that each instruction performs.

General Register organization.

- It is more convenient & more efficient to store the intermediate value of ALU in processor registers.
- when a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.
- The register communicate with each other not only for direct data transfer but also while performing various micro-operations.

Date

- A bus organization for Seven CPU registers is shown in figure below.
  The o/p of each multiplexer to form the 2 buses A & B.
- The selection lines in each multiplexer select one register or the input data for the particular bus.
- The A & B buses form the i/p to a common ALU.



Date

(15) # Register set with Common ALU

The control unit that operates the CPU bus system directs the information flow through the registers & ALU by selecting the various component in the system.

For Example - to Perform the operation:

$R_1 \leftarrow R_2 + R_3$

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

① MUX Selector [SEL A] :- to place the content of R2 into bus A.

② MUX Selector [SEL B] :- to place the content of R3 into bus B

③ ALU operation Selector :- to provide the arithmetic addition (A + B).

④ Decoder destination Selector [SEL D] :- The content of the output bus into R1.

16

Date

CONTROL WORD

⇒ There are 14 binary selection inputs in the unit & their combined value specifies a control word.

⇒ 3 fields contain 3 bits each & 1 field has 5 bits.

⇒ The 3bits of SELA select a source register for the A input of the ALU.

⇒ The 3 bits of SELD select a destination register using the decoder & its 7 load output.

⇒ The 5 bits of OPR select one of the operation in the ALU.

STACK ORGANISATION

Date

@ A stack is a storage device that stores information in such a manner that the item stored last in the first item retrieved.

@ The stack in digital computer is a memory unit. The register that holds the address for the stack is called Stack Pointer [SP]. because its value always point at the top item in the stack.

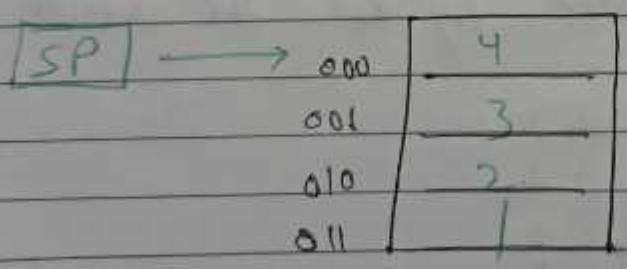
@ There is a data register which hold data.

@ The 2 operation of stack are the insertion & deletion of item.

@ The operation of stack of insertion is called PUSH

@ The operation of deletion is called POP.

Note — Nothing is pused or popped in a computer stack.
These operation are simulated by increament & decreement the STACK POINTER REGISTER.



Date

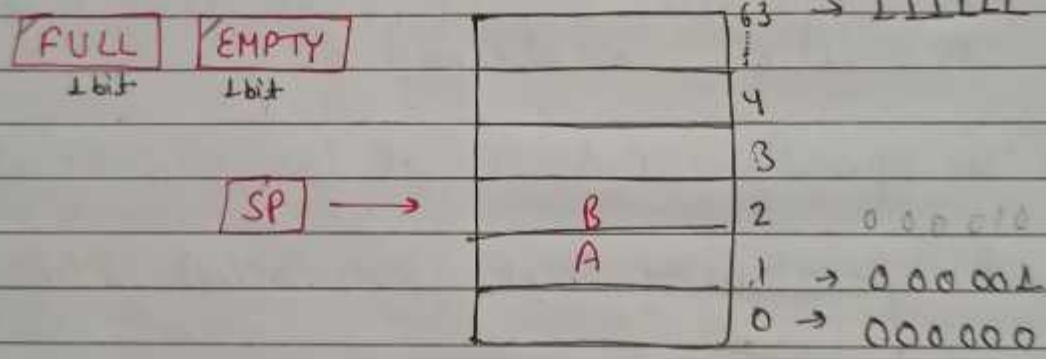
Types of Stack Organisation

(a) Register Stack
(b) Memory Stack

a) Register Stack

A stack which can be organized as a collection of a finite number of registers is called Register Stack.

Fig: below show the organization of a 64 word register stack



⇒ In a 64 word stack, the Stack Pointer contains 6 bit because $2^6 = 64$.
Since SP has only six bits, it can't exceed a number greater than 63 [111111 is binary] when 63 is incremented by 1 the result is 0 since.

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

Date

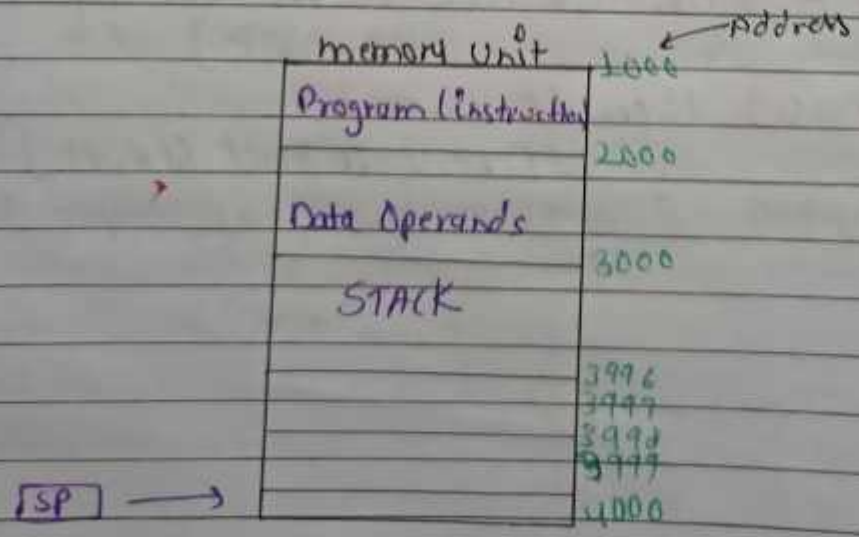
- **POP :** A new item can be deleted from stack if the stack is not empty. with the following sequence of micro operations.

DR ← M[SP] {Read item from the top of stack}
SP ← SP − 1 {Decrement the SP}
if (SP = 0), then (EMPTY ← 1)
{check stack is empty}
FULL ← 0 {Mark the stack Not full}

(19)

b) MEMORY STACK

A stack which can be organized as a collection of a finite number of memory ward is called Memory stack.



OR

Date

- **PUSH Operation** - A new item is inserted with the Push operation as

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

The stack pointer is decrement so that it points at the address of the word.

- **POP Operation** - A new item is deleted with the POP operation as.

$$DR \leftarrow M(SP)$$
$$SP \leftarrow SP + 1$$

The top item is read from the stack into DR. the SP is then incremented to point at the new item in the stack.

(6)

Q. Let SP = 000000 in the stack how many item are there in the stack if

(a) FULL = 1 & EMPTY = 0

(b) EMPTY = 1 & FULL = 0

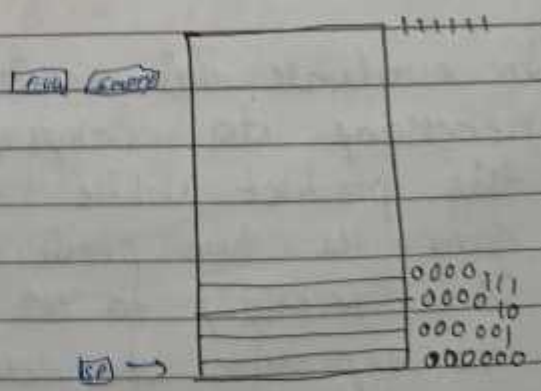
Solution

(a) $SP \leftarrow SP + 1$

$M[SP] \leftarrow DR$

Stack will be always full

total will be item = 64



(b) Stack will be empty total item 0.

Date

A stack is organized such that SP always points at the next empty location on the stack.
this means that SP can be initially to 4000 & the first item in the stack stored in location 4000. [memory stack]
Write the microoperation for PUSH & POP operation

PUSH ⇒ SP ← SP - 1
M[SP] ← DR

POP ⇒ DR ← M[SP]
SP ← SP + 1

Reverse Polish Notation [RPN]

A stack organization is very effective for evaluating arithmetic expression.

Arithmetic Expression
A * B + C * D is called infix notation

In evaluate this arithmetic expression it is necessary to compute the product A * B store this product while computing C * D & then sum the two products. So in infix notation it is necessary to the scan back & forth along the expression to determine the next operation to the performed.

Date

Arithmetic expression can be represented in Prefix notation (Polish Notation) places the operator before the operands.

Example - A+B infix Notation
+AB Prefix " or Polish Notation

The Postfix Notation referred to as Reverse Polish Notation. Places the operator after the operands.

example - A+B infix Notation
AB+ Postfix "

The RPN is in a form suitable for stack manipulation

(3) # CONVERSION TO RPN

we first Perform all arithmetic inside inner Paranthesis then inside outer paranthesis & do multiplication & division operations before addition & substraction operations.

Example -

(A+B) * [C*(D+E) +F]
AB+ * [C*DE+ +F]
AB+ *[CDE+*+F]
AB+ *[CDE+*F+]
AB+CDE+*F+*

Date

Evaluation of Arithmetic Expressions

- To produce consist of first converting the arithmetic expression into its equivalent RPN.
- The operands are pushed into the stack in the order in which they appear.
- The following microoperation are executed with the stack when an operation entered in a calculator or in a computer.

a) The two topmost operands in the stack are used for the operation.

b) The stack is popped & the result of the operation replaced the lower operand.

- By Pushing the operands into the stack continously & performing the operations as defined above, the expression is evaluated in the proper order & the final result remains on top of the stack.

Convert the following numerical expression into reverse Polish notation & show the stack operations for evaluating the numerical result.

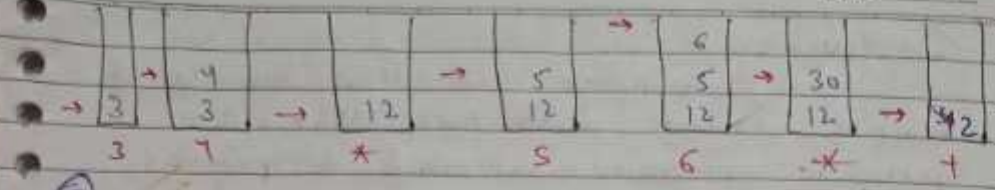
$(3 * 4) + (5 * 6)$

Convert in RPN

$34*+56*$

$34*56*+$

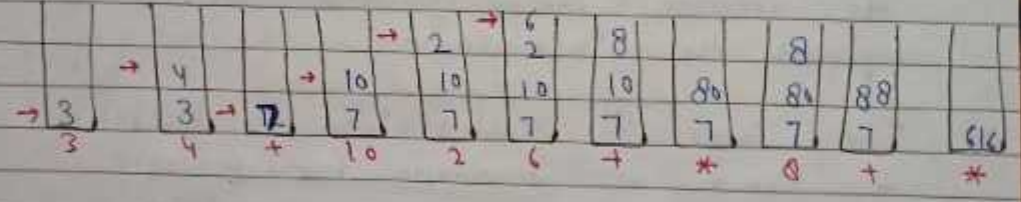
Date



(a) Convert the following numerical arithmetic expression in RPN & show the stack operation for evaluation the numerical result.

$(3*4)+[10(2+6)+8]$

$34+[10 26+ +8]$
$34+[1026+*+8]$
$34+[1026+*8+]$
$34+1026+*8+*$



$(3+4)[10(2+6)+8]$
$[34+)*[10*26+ +8]$
$34+*[1026+*+8]$
$34+*[1026+*8+]$
$34+1026+*8+*$

Date

23

* INSTRUCTION

Computer instructions are set of machine language instructions that a particular processor understands & executes.

→ A computer performs tasks on the basis of the instruction provided.

24

Instruction Formats

* CPU fetches instructions from memory & it is the function of control unit to interpret each instruction code & provide the necessary control functions needed to process the instruction.

* Every instruction divided into 3 field & known as instruction format.

1) Mode field - It specifies the way the operand (data) or effective address is determined.

2) Operation code field - (OPCODE) = It specifies what operation to be performed.

3) Address field - It shows the memory address or a processor register address.

Date

* The number of address fields in the instruction format of a computer depends on the internal organizational of its registers.

* Most computers fall into one of 3 types of CPU Organizations.

(25)#

1. Single accumulator Organization
2. General register "
3. Stack Organization

1) Single Accumulator Organization

An accumulator Organization all operations are performed with an implied accumulator register.

The instruction formation of in this type of computer uses one address field.

ADD X;

AC ← AC + M[X]

where X is the address of operand (data). AC is the Accumulator register M[X] is the address of Operand

2) General register Organization - The instruction format in this type of computer needs 3 register address fields.

ADD R_1, R_2, R_3; $R_1 \leftarrow R_2 + R_3$

Date

The number of address fields in the instruction can be reduced from 3 to 2 if the destination register is the same as one of the source register, so the above example can be reduced to 2 address as:-

ADD R_1 R_2; $R_1 \leftarrow R_1 + R_2$

3) Stack Organization - In stack org. we have PUSH & POP instructions which registers an address field.

PUSH x;
TOS $\leftarrow$ x;

will the PUSH the word at address x to the top of the stack.

The instruction in a stack organization consists of an operation code only with no address field this operation has the effect of popping the 2 numbers from the stack, adding the number & pushing the sum into the stack.

Note - For using zero, One, two or three address instruction we use the symbols ADD SUB MUL & DIV for 4 arithmetic operations; MOVE for the transfer type operation; LOAD & STORE for transfer to and from memory & AC register we assume that the operands are in memory addresses.

A, B, C, & D & the result must be stored in memory at address X.

(10) Ex- WAP to evaluate the arithmetic statement
$X = (A+B) * (C+D)$

(a) using a general register computer with 3 address instructions.
(b) using a general register computer with 2 address instruction.
(c) using a accumulator type computer with one address instruction.
(d) using a stack organized computer with zero address instructions.

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

```
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
```

(b) 2 address instruction

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

Date

(c) 1 address instruction

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

(d) 0 address instruction X = (A+B) * (C+D)

X = AB+ * CD+
X = AB+CD+ *

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

Q.11

Date

WAP to evaluate the arithmetic statement

$$X = \frac{(A-B)+[C*([D*E]-F)]}{G+H*K}$$

(a) using 3 Address

(b) " 2 "

(c) " 1 "

(d) " 0 "

(a) using 3 Address

```
MUL  R1  D,E ;   R1 ← m[D] * m[E]
SUB  R2  R1,F ;  R2 ← R1 - m[F]
MUL  R3  R2,C ;  R3 ← R2 * m[C]
SUB  R4  A,B ;   R4 ← m[A] - m[B]
ADD  R5  R3,R4 ; R5 ← R3 + R4
MUL  R6  H,K ;   R6 ← m[H] * m[K]
ADD  R7  R6,G ;  R7 ← R6 + m[G]
DIV  X   R5,R7   X ← R5 / R7
```

(b) using 2 Address

```
MOVE  R1,D ;    R1 ← m[D]
MUL   R1,E ;    R1 ← m[D] * m[E]
SUB   R1,F ;    R1 ← R1 - F
MUL   R1,C ;    R1 ← R1 * C
MOVE  R2,A ;    R2 ← m[A]
SUB   R2,B ;    R2 ← R2 - m[B]
ADD   R2,R1 ;   R2 ← R2 + R1
MOVE  R1,H ;    R1 ← m[H]
MUL   R1,K ;    R1 ← R1 * m[K]
ADD   R1,G ;    R1 ← R1 + G
MOVE  X,R1 ;    X ← R1
DIV   X,R2      X ← R2 / R1
```

Date

(c) $X = \frac{(A-B)+[(*[(D*E)-F])]}{G+(H*K)}$ by using 1 address

```
LOAD  D;   AC ← m[D]
MUL   E;   AC ← m[D] * m[E]
SUB   F;   AC ← AC - F
MUL   C;   AC ← AC * C
STORE T;   T ← AC
LOAD  A;   AC ← m(A)
SUB   B;   AC ← m[A] - m[B]
ADD   T;   AC ← AC + T
STORE P;   P ← AC
LOAD  H;   AC ← m[H]
MUL   K;   AC ← m[H] * m[K]
ADD   G;   AC ← AC + G
DIV   Q;   AC ← P/Q
STORE X;   m(X) ← AC
```

(d) by 0 address

$X = \frac{(A-B)+[(*[(D*E)-F])]}{G*(H*K)}$

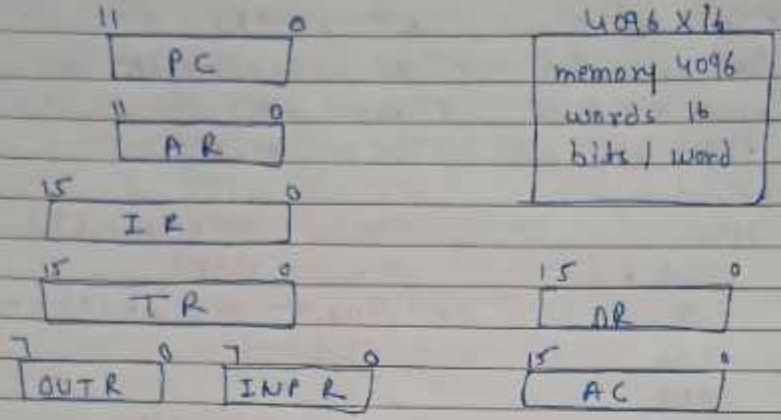
$X = AB- +[(*[DE*F-]])/HK*G+$

$X = AB-CDE*F-*+/HK*G+$

$AB-CDE*F-*+HK*G+/X=$

Date

Basic Computer Registers & memory



Let the memory unit has capacity of 4096 words & each word contain 16 bits.
Since, $4096 = 2^{12}$ so, 12 bits will be the address of memory.

- DR (Data Register) - Holds the operand (data) read from memory Here it's size will be 16 bits.

- AC (Accumulator Register) - It is a general Purpose register it also keep data so its size in this example will be 16 bits.

- PC (Program counter) - Its holds the address of the next instruction to be read from memory after the current

instruction is executed so, In this example its size will be of 12 bits.

- AR (Address Register) - It holds the memory address. So here its size will be of 12 bits.

- IR (Instruction Register) - The instruction read from memory is placed in the instruction register so, in this example it's size will be of 16 bits.

- TR [Temporary Register] - It is used to hold temporary data during the proceing here its size will be of 16 bits.

- OUTR (Output Register) - It holds the for an o/p character for an o/p device, suppose here its size in 8-bits.

- INPR (Input Register) - It holds the input character from an i/p device but it's size in 8-bits.

Date

Addressing Modes

The way the operands are choosen during program execution is dependent on the addressing mode of the instruction.

Mode	OPCODE	Operand	Address of operands

Instruction format

In the instruction format the mode field is used to locate the operand needed for the operation. Since operand may be in memory or register.

Computers use addressing modes techniques because of the following advantages.

(a) To give programming language versatility to the user by providing such facilities as pointer to memory counter for loop control indexing of data & program relocation.

(b) To reduce the number of bits in the addressing field of the instruction.

Date

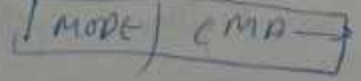
Types of addressing modes

1) Implied mode
2) Immediate mode
3) Register Mode
4) Register indirat Mode
5) Autoincrement or Auto decrement mode
6) Direct Address Mode
7) Indirect " "
8) Relative " "
9) Indexed " "
10) Base Register addressing mode

NOTE- Effective Address :- is the memory address or register address obtained from the computation dictated by a given addressing mode.

1) Implied Mode - In this mode operands are specified implicity in the definition of the instruction.

eg - The instruction "complement accumulater is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction. Zero address instructions in stack organized the computer are implied mode instruction since the operands are implied to be on top of the stack.



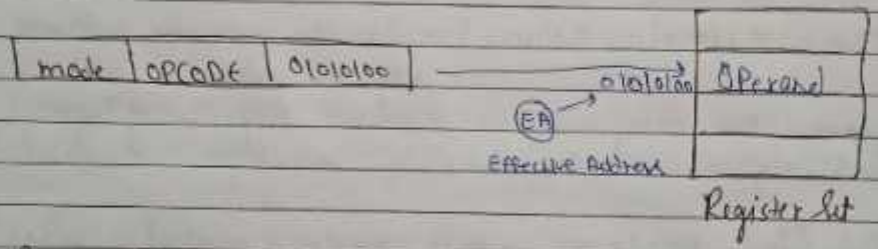
Date

(2) Immediate mode - In this mode the operand field contains the actual operand not the address of operand.
This mode is useful for initializing registers to a constant value.

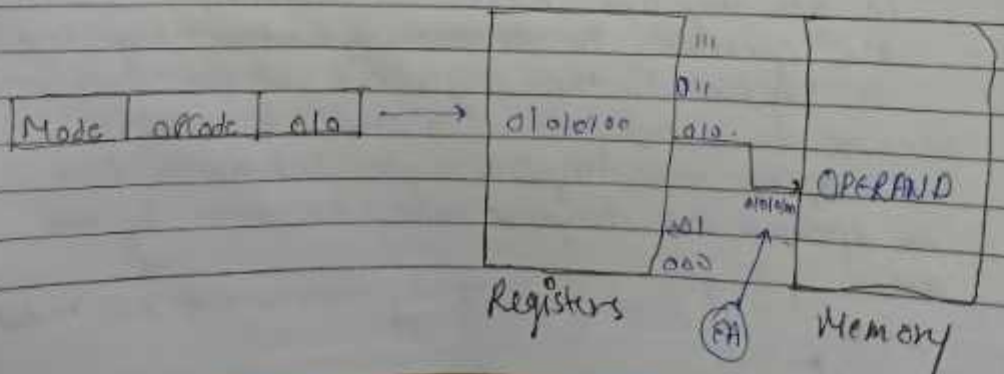
MOVE R #20 {initializes the register R to a value 20.}

Mode	MOV	#20

(3) Register mode or Register Direct Mode - In this addressing mode. The address field of the instruction refers to a CPU Register that Contains the Operand.

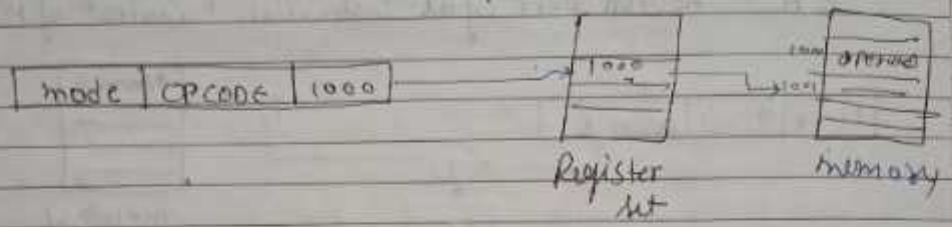


(4) Register indirect mode - In this addressing mode the address field of the instruction refers to a CPU register which contains the address of the operand in memory.

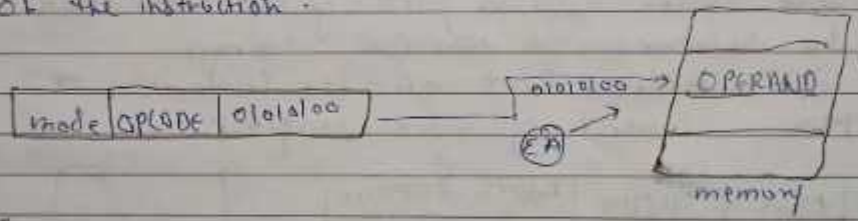


Date

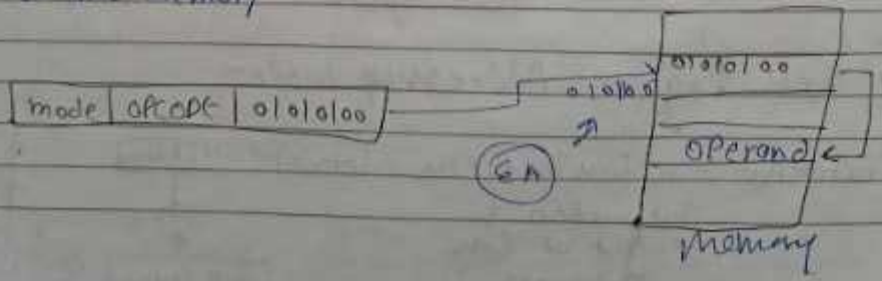
(5) Autoincrement or Autodecrement - This is similar to the register indirect mode except that the register is incremented or decremented after its value is used to access memory.



(6) Direct Address mode - In this addressing mode the operand resides in memory & its address is given directly by the address field of the instruction.



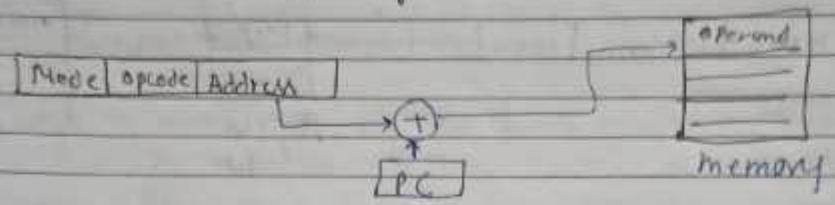
(7) Indirect Addressing mode - In this addressing mode the address field of the instruction specifies the address of memory location that contains the effective address of the memory.



Date

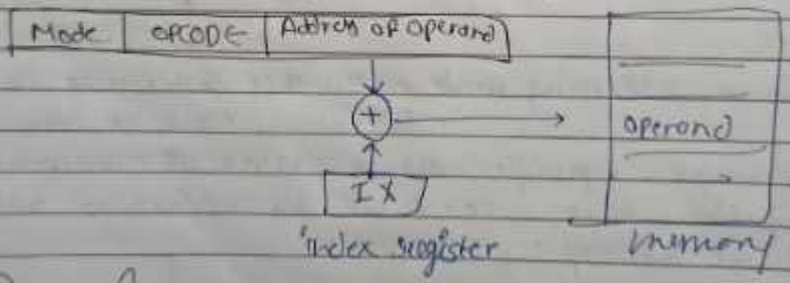
(8) Relative addressing mode - In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

EA = Address part of the instruction + Content of PC



(9) Indexed Addressing Mode - In this mode content of an index register is added to the address part of the instruction to obtained the effective address.

EA = Address Part of the Instruction + Content of index register.



(10) Base Register Addressing mode-

Effective Add. = Address Part of the instruction + Content of Base Address

30

Application Of Addressing Mode

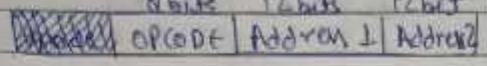
Date

Addressing Mode	Application
Immediate Addressing Mode	* To initialize register to a constant value.
Direct Addressing Mode & Register direct Addre. Mode	* To access static data * To impliment variables
Indirect Addressing Mode & Register indirect Address. Mode	* To impliment pointerers because pointer are memory location that store the address of another variable. * To pass array as a parameter because array name is the base address & pointer is needed to point the address.
Relative Addressing Mode	* For Program relocation at run time i.e. position independent code * To change the normal sequence of execution of instructions. * For branch type instructions since it directly updates the program counter.
Index Addressing Mode	* For array implimentation or array addressing.
Base Register addressing Mode	* For writing relocatable code i.e. for relocation of Program in memory or that run time. * For handling recursive Procedures.
Autoincrement & Autodecrement Mode	* For implementing loop. * For stepping through array in a loop. * For implementing a stack as push & POP.

Date

12

A computer has 32 bit instructions & 12 bit addresses. If there are 250 two-address instructions how many one-address instructions can be formulated.



Two address instructions

$2^8 = 256$ combinations

but there are 250 two address instructions, remaining $256 - 250 = 6$ combination can be used for one address.

1 address instruction $= 6 \times 2^{12}$

$= 6 \times 2^{10} \times 2^2$

$= 6 \times 1024 \times 4$

$= 24576$

In 1 address instruction there is no address field.

13

The memory unit of a computer has 256K words of 32 bits each. The computer has an instruction format with four fields: an operation code field, a mode field to specify one of seven addressing mode a register address field to specify 1 of 60 processor register & a memory address. Specify the instruction format & number of bits in each field if the instruction is in 1 memory word.

Date

Solved

$256\ K = 2^8 \times 2^{10} = 2^{18}$

5 bit	3bit	6 bit	18bit	= 32 bit
OPCODE	MODE	Register Address	Memory Address	

∴ Size of memory = 256 K
$= 2^8 \times 2^{10} = 2^{18}$
18 bits memory address

Total Processor register = 60
address of process register $= 2^6 = 64$
6 bit address of register
address of mode = seven address mode $= 7 = 2^3$
3 bits mode field.

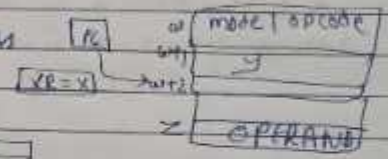
(14)

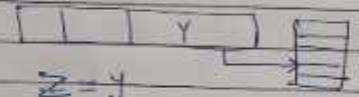
Date

A two word instruction is stored in memory at an address designated by the symbol w. The address field of the instruction (stored at w+1) is designated by the symbol y. The operand used during the execution of instruction is stored at an address symbolized by z an index register contains the value x. State how z is calculated from the other addresses if the addressing mode of the instruction is.

1) Direct 2) Indirect 3) Relative 4) Indexed

Solve- here Z = Effective Address



i) Direct - 

Z = y

ii) Indirect - Z = m[y]

iii) Relative - Z = Y + w + 2

iv) Indexed - Z = Y + X

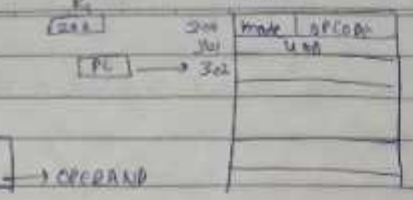
(15)

An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 800. Evaluate the EA if the addressing mode of the instruction is

1) Direct
2) Immediate
3) Index with R1 as the index register
4) Register indirect
5) Relative

Date

Soln → i) Direct - EA = 400



ii) Immediate -

mode	OPCODE	400	→ OPERAND

at address 301

Effective address - 301

iii) Relative - Effective address = Address of instruction + Content of PC

EA = 400 + 302
= 702

iv) Indexed - Effective Address = Address of instruction + content of index Register

EA = 400 + 200
= 600

v) Register Indirect -

EA = 200

[ALU] 30/01/2024

UNIT-2: Arithmetic Logic Unit

(31)

Design a binary adder or fast adder or ripple Adder.

Soln:- 4 bit binary Adder or Fast adder or Ripple Adder

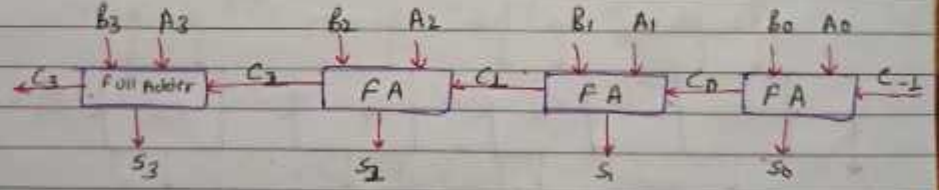
→ To implement the add micro-operation with hlw we need Binary Adder with registers.

→ A 4 bit binary adder is capable of adding 4-bit of Register.

→ we can design 4 bit binary Adder with 4 Full adder.

→ For designing n bit Binary Adder we need n-Full adder

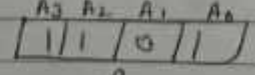
→ The binary adder is constructed with full adder circuit connected in cascode, with the output carry from on full adder.



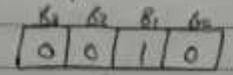
4-bit binary Adder

Connected to the input carry of the next full adder the first value of $C_0 = 0$

For Example - Let we have 2 register R_1 & R_2 of 4 bits each



R_1



R_2

we can add the value of these 2 registers by using the above binary Adder.

Full adder { 1/01 } half adder

100

Date

Carry Look Ahead Adder [CLA Adder] [Fast Adder]

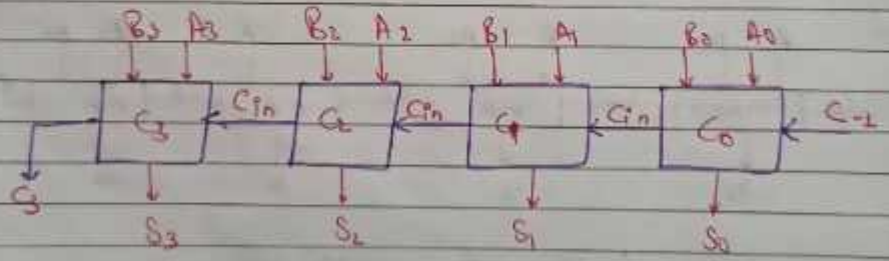
→ It predict the carry before actually it is produced.

→ It is superior than full adder in term of speed & speed is more important feature of any digital circuit.

→ In full adder when we add 2 number A & B & let suppose both A & B are 4 bit numbers. The sum will take some time because it depend on carry and this time is called Propogation delay.

→ The Propogation delay of sum & carry inside adder is different.

→ But we are more concerned about the propogation delay of carry.



Date

Input			Output	
A	B	Cin	Sum (S)	Carry (Cout)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$C_{out} = \underbrace{A \cdot B}_{G} + (A \oplus B) C_{in}$

Carry Generator — Carry propagator

$\boxed{C_{out} = G + P C_{in}}$

$\boxed{C_i = G_i + P_i C_{i-1}}$ — (i)

put $i = 0$ $\boxed{C_0 = G_0 + P_0 C_{-1}}$ — (ii)

put $i = 1$ in eqn (i)

$\boxed{C_1 = G_1 + P_1 C_0}$ — (iii)

put the value of C_0 in eqn (iii)

$C_1 = G_1 + P_1 (G_0 + P_0 C_{-1})$

$\boxed{C_1 = G_1 + P_1 G_0 + P_0 P_1 C_{-1}}$

put $i = 2$ in eqn (i)

Date

$C_2 = G_2 + P_2 C_1$ — (iv)

put the value of C_1 in eqn 4

$C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_{-1})$

$\boxed{C_2 = G_2 + P_2 G_1 + P_1 P_2 G_0 + P_2 P_1 P_0 C_{-1}}$

put $i = 3$ in eqn $-$ (i)

$C_3 = G_3 + P_3 C_2$ — (v)

put the value of C_2 in eqn (v)

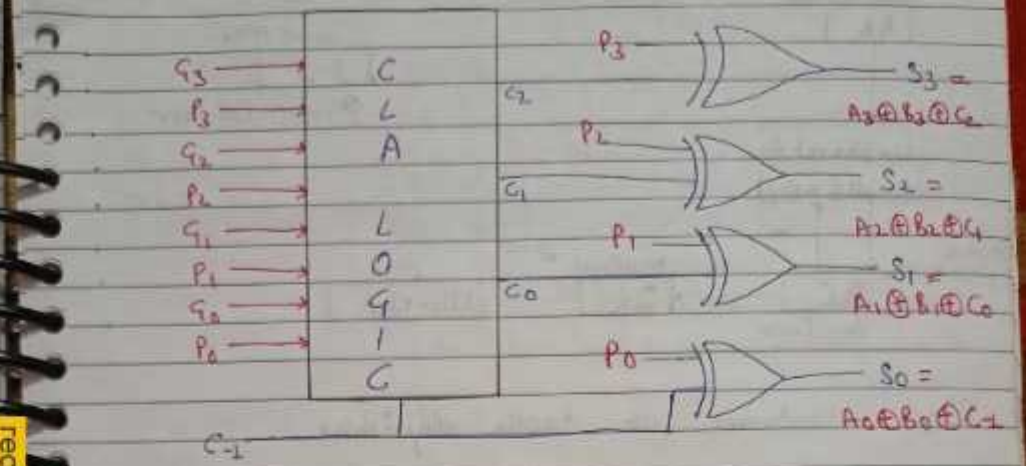
$C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_1 P_2 G_0 + P_2 P_1 P_0 C_{-1})$

$\boxed{C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_1 P_2 P_3 G_0 + P_3 P_2 P_1 P_0 C_{-1}}$

$G_0 = A_0 B_0$
$G_1 = A_1 B_1$
$G_2 = A_2 B_2$
$G_3 = A_3 B_3$

$P_0 = A_0 \oplus B_0$
$P_1 = A_1 \oplus B_1$
$P_2 = A_2 \oplus B_2$
$P_3 = A_3 \oplus B_3$

$S_0 = A_0 \oplus B_0 \oplus C_{-1}$
$S_1 = A_1 \oplus B_1 \oplus C_0$
$S_2 = A_2 \oplus B_2 \oplus C_1$
$S_3 = A_3 \oplus B_3 \oplus C_2$



35

BOOTH ALGORITHM

Multiplication both +ve & -ve (signed number) = Booth Algorithm

8 4 2 1

14 → 1 1 1 0 → multiplicand
×9 → 1 0 0 1 → multipliers
126

1 1 1 0 } Partial Product
0 0 0 0 } Reset / Store AC
0 0 0 0 }
1 1 1 0
1 1 1 1 1 1 1 0 → Product

* The hlw implementation of both algo. requires the register the configuration as shown below.
* we use register AC, BR & QR. Qn designates the least significant bit of the multiplier in register QR.
* An extra flip-flop Qn+1 is appended to QR for a double bit inspection of the multiplier.

Date 01/02/24

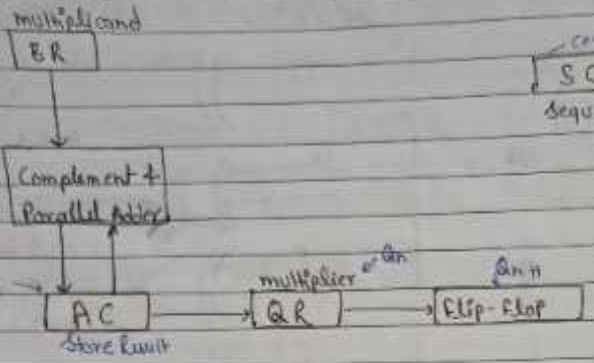


Fig- Hardware of Booth algorithm.

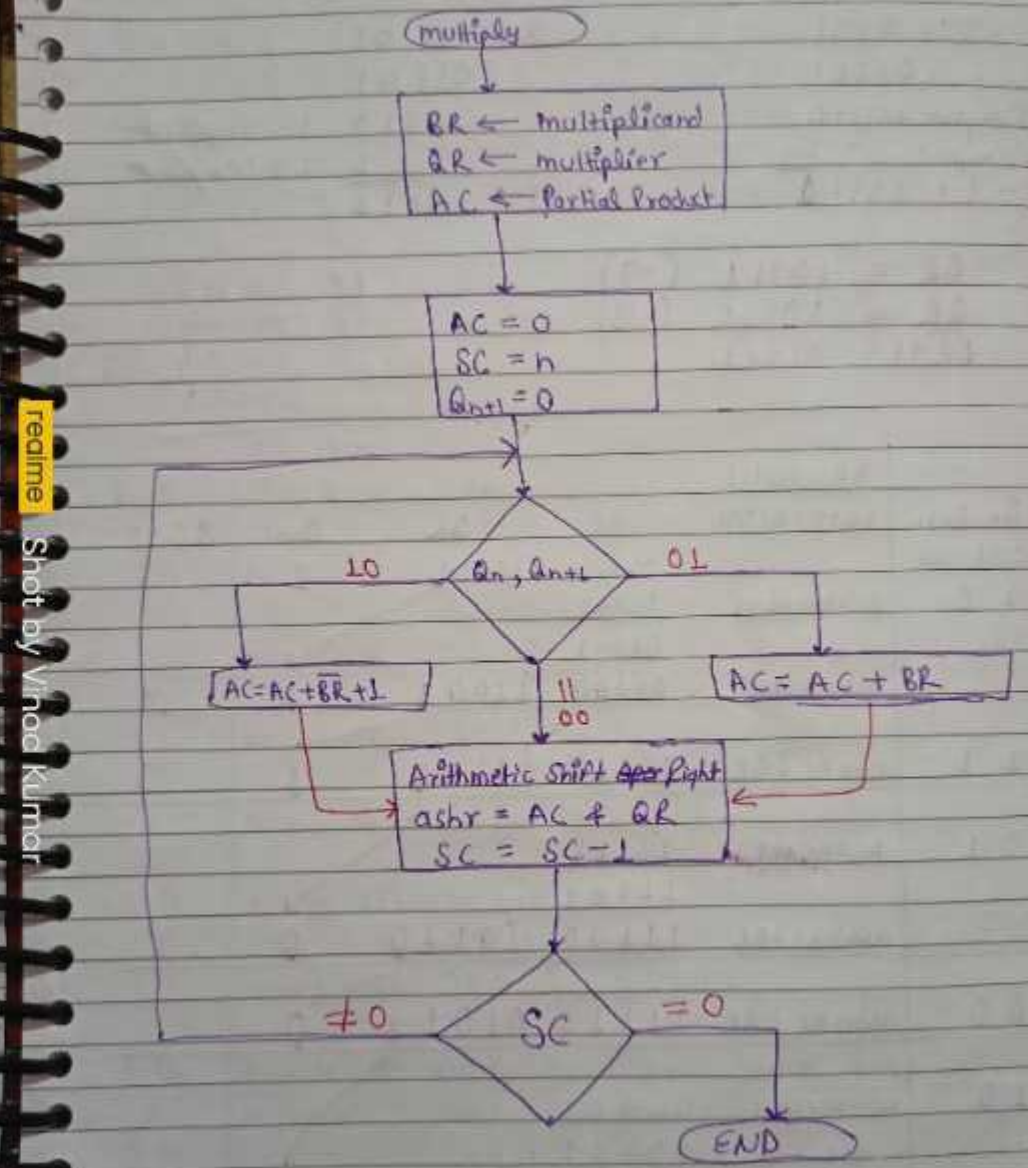
0 ← +ve
1 ← -ve

* If the two bits are equal to 10. it means that the 1st 1 in a string of 1's has been encountered. This requires a subtraction of multiplicand from the partial product in AC.
* If the two bits are equal to 01, it means that the 1st 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the Partial Product in AC.
* When the 2 bits are equal i.e. 00 or 11 the Partial Product does not change.
* The next step is to shift right the Partial Product & multiplier (including bit Qn+1). This is an arithmetic shift right operation which shift AC & QR to the right & leaves the sign bit in AC unchanged.
* The next step is the sequence counter (SC) is decremented & the computational loop is repeated n times.

(34)

Date 01/01/21

Flowchart of Booth Algorithm



16 8 4 2 1 ✓ ✓ 10011₂ / 10100 Date

Q. Multiply -9 x -13 using booth algorithm.

+9 → 1001
01001
1's complement 10110
2's complement + 1
-9 → 10111

+13 → 1101
01101
10010 1's complement
+ 1 2's complement
-13 → 10011

BR = 10111 (-9)
QR = 10011 (-13)
BR+1 = 01001

BR = 10111
BR = 01000
+1
01001

	BR = 10111				
Qn Qn+1	BR+1 = 01001	AC	QR	Qn+1	SC = 5
Initial Value		00000	10011	0	5
1 0	AC = AC + BR + 1	01001			
		01001			
	ashr AC & QR	00100	11001	1	4
1 1	ashr AC & QR	00010	01100	1	3
0 1	AC = AC + BR	10111			
		11001			
	ashr AC & QR	11100	10110	0	2
0 0	ashr AC & QR	11110	01011	0	1
1 0	AC = AC + BR + 1	01001			
		00111			
	ashr AC & QR	00011	10101	1	0

64 32 16 8 4 2 1

Result 96 117 21

8 4 2 1 Date

(17)

Q multiply -9 X 13 using booth Algo

9 → 1001
01001
1's → 10110
2's +1
10111

13 → 1101
01101
1's → 10010
2's +1
10011

{ if +ve value give as no comp - binary required

BR = 10111
QR = 01101
$\overline{BR}+1$ = 01001

BR = 10111
BR = 01001
Block

	BR = 10111				
Q_n Q_{n+1}	$\overline{BR}+1$ = 01001	AC	QR	Q_{n+1}	SC
		00000	01101	0	
10	AC = AC + $\overline{BR}+1$	01001			
		01001			
	ashr AC & QR	00100	10110	1	
01	AC = AC + BR	10111			
		11011			
	ashr AC & QR	11101	11011	0	
10	AC = AC + $\overline{BR}+1$	01001			
		00110			
	ashr AC & QR	00011	01101	1	
11	ashr AC & QR	00001	10110	1	
	AC = AC + BR	10111			
01		11000			
	ashr AC & QR	11100	01011	0	

64 32 16 8 4 2 1

Date

Result 1 1 1 0 0 0 1 0 1 1

1's → 0 0 0 1 1 1 0 1 0 0

Complement 2's +1

0 0 0 1 1 1 0 1 0 1 Ans

64 32 16 8 4 2 1

Q.16 multiply 9 x -13 using Booth Algorithm.

6 + 2

9 → 1 0 0 1

+9 → 0 1 0 0 1

13 → 1 1 0 1

+13 → 0 1 1 0 1

1's com → 1 0 0 1 0

2's com → +1

-13 → 1 0 0 1 1

BR = (9) 01001

BR = (-13) 10011

BR+1 = 10111

	BR = 01001				
$Q_n Q_{n+1}$	$\overline{BR}+1 = 10111$	AC	QR	Q_{n+1}	SC
		00000	10011	0	
1 0	AC = AC + $\overline{BR}$ + 1	10111			
		10111			
	ashr AC + QR	11011	11001	1	
1 1	ashr AC + QR	11101	11100	1	
	AC = AC + BR	01001			
0 1		00110			
	ashr AC + QR	00011	01110	0	
	AC				
0 0	ashr AC + QR	00001	10111	0	
	AC = AC + $\overline{BR}$ + 1	10111			
1 0		11000			
	ashr AC + QR	11100	01011	1	

Result = 11100 01011

complement 1's = 00011 10100

2's = 1

00011 10101

(19)

i) +15 × +13 16 8 4 2 1

+15 → 01111 +13 → 01101

BR = 01111 $\overline{BR}+1$ = 10001 QR = 01101

$Q_n Q_{n+1}$	BR = 01111 $\overline{BR}+1$ = 10001	AC	QR	Q_{n+1}	SC
		00000	01101	0	
1 0	AC = AC + $\overline{BR}+1$	10001			
		10001			
	ashr AC & QR	11000	10110	1	
	AC = AC + BR	01111			
0 1		00111			
	ashr AC & QR	00011	11011	0	
	AC = AC + $\overline{BR}+1$	10001			
1 0		10100			
	ashr	11010	01101	1	
1 1	ashr	11101	00110	1	
0 1	AC = AC + BR	01111			
		01100			
	ashr	00110	00011	0	

BR 4 3 2 1 16 8 4 2 1

Date

(30)

+15 * -13

15 = 1111
= 01111

+13 = 1101
= 01101

1's Com' = 10010
= 1
-13 → 10011

QR = 10011
BR = 01111
$\overline{BR}+1$ = 10001

	BR = 01111				
A_n Q_{n+1}	$\overline{BR}+1$ = 10001	AC	QR	A_n	SC
		00000	10011	0	
1 0	AC = AC + $\overline{BR}+1$	10001			
		10001			
	ashr AC+QR	11000	11001	1	
1 1	ashr AC+QR	11100	01100	1	
0 1	AC = AC+BR	01111			
		01011			
	ashr AC+QR	00101	10110	0	
0 0	ashr AC+QR	00010	11011	0	
1 0	AC = AC+$\overline{BR}+1$	10001			
		10011			
	ashr AC+QR	11001	11101	1	

Result 1100111101

1's Comp → 0011000010

2's Complement +1

0011000011

128 64 32 16 8 4 2 1

Array Multiplier

Date _____

→ The multiplication of two binary numbers can be done with 1 micro-operation by means of a combinational circuit that forms the product bits all at once. This fast way of multiplying two numbers is known as Array multiplier.

* Consider the multiplication of two 2 bit numbers as shown in fig below. The multiplicand bits are b_1 & b_0, the multiplier bits are a_1 & a_0 & the product is c_3, c_2, c_1, c_0.

* The First Partial Product is formed by multiplying a_0 by $b_1 b_0$.

* This is identical to an AND operation & can be implemented with an AND Gate.

* As shown in figure, the first Partial Product is formed by means of two AND gates. The second Partial Product is formed by multiplying a_1 by $b_1 b_0$ & is shifted one position to the left.

* The Two Partial Product is added with 2 half adder

Note → For 'j' multiplier bits & 'k' multiplicand bits we need j*k AND gates & (j-1)*k bit adders to Produce a product of j+k bits.

Date

b_1 b_0 – multiplicand

a_1 a_0 – multiplier

a_0b_1 a_0b_0 } Partial Product

a_1b_1 a_1b_0

C_3 C_2 C_1 C_0 ← Product

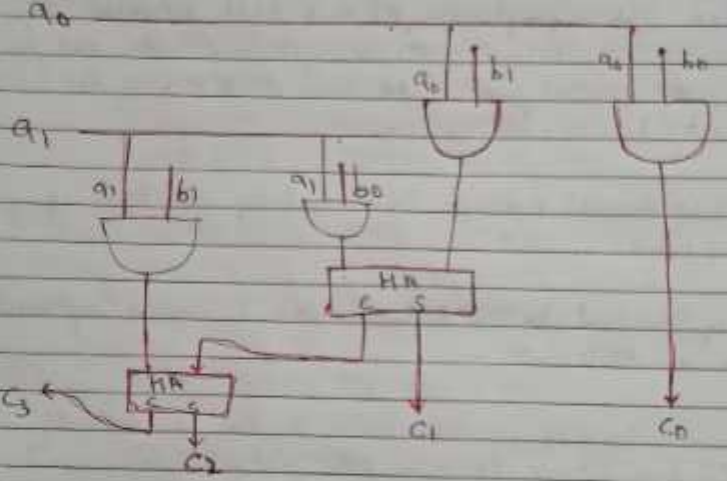


Fig- 2bit by 2 bit array multiplier.

4bit by 3 bit array multiplier

b_3 b_2 b_1 b_0

a_2 a_1 a_0

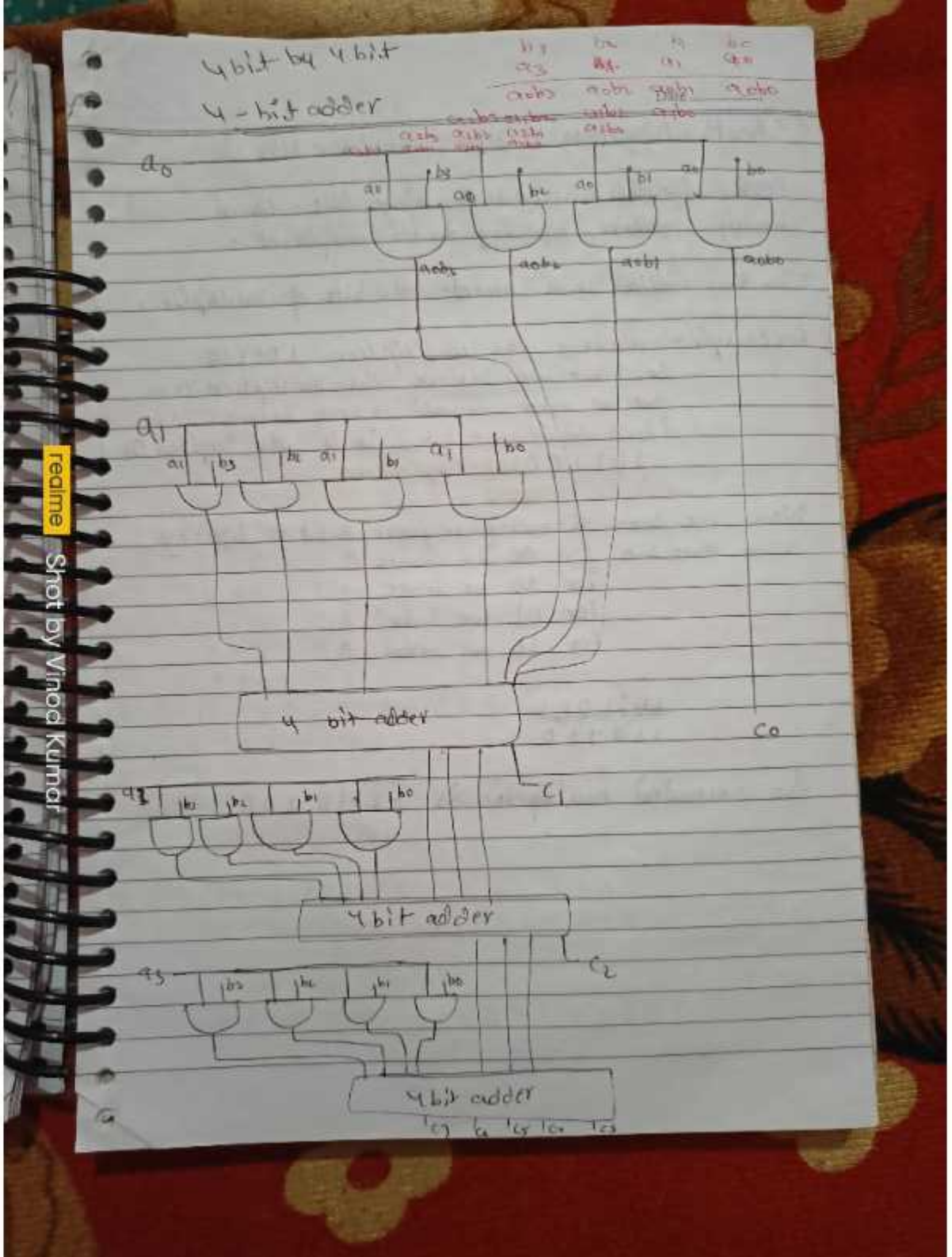
a_2b_3 a_2b_2 a_2b_1 a_2b_0

a_1b_3 a_1b_2 a_1b_1 a_1b_0

a_0b_3 a_0b_2 a_0b_1 a_0b_0

C_6 C_5 C_4 C_3 C_2 C_1 C_0

↑ Carry



Date

Booth Algorithm [Recoded multiplier bits]

Booths algorithm is used to multiply two signed binary numbers by using two's compliment.

- In this method we recode the bits of multiplier.

For Example - we have the multiplier 101100
So, first we recode the multiplier for that at first we add a zero before L.S.B. [least significant bit] it is called implied 0.
101100[0] → implied zero

Now, we scan the multiplier from right to left by using two bits for 00 we write 0
for 10 we write -1
For 01 we write 1
For 11 we write 0

1 0 1 1 0 0 0
-1 1 0 -1 0 0

So recorded multiplier is -1 +1 0 -1 0 0

Date 03/02/24

(22) Q multiply 010011 by 001100 using Booths Algorithm

Solve - Here multiplier : 001100 (+12) ⇒ QR
multiplicand : 010011 (+19) ⇒ BR

Two's Compliment of multiplicand [$\overline{BR}$+1] ⇒ 101101

multiplier - 0 0 1 1 0 0 ← implied zero
modified multiplier - 0 0 1 1 0 0 0
← ← ← ← ←
0 +1 0 -1 0 0

write multiplicand bit

0 1 0 0 1 1 → Take 2's compliment of multiplicand bits.
0 +1 0 -1 0 0

0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 ·
Sign bits → 1 1 1 1 1 0 1 1 0 1 · ·
0 0 0 0 0 0 0 0 0 · · ·
0 0 0 1 0 0 1 1 · · · ·
0 0 0 0 0 0 0 · · · · ·
1 | 0 0 0 0 1 1 0 0 1 0 0 | Result
Discard ↓ represent Sign

(23) Q multiply 110101 by 011010 using modified booth algorithm.

BR → multiplicand → 1 1 0 1 0 1
QR → multiplier → 0 1 1 0 1 0

$\overline{BR}$ +1 → 0 0 1 0 1 1
modified multiplier - 0 0 1 1 0 1 0 0
← ← ← ← ← ←
+1 0 -1 +1 -1 0

Date

110101

```
        0 1 0 0 1 0
       +1 0 -1 +1 -1 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 1
0 0 0 0 0 1 1 0 1 0
0 0 0 0 0 1 0 1 1
0 0 0 0 0 0 0 0
0 0 1 1 0 1 0
[0 0 1 1 1 0 0 1 0 1 1 0] Result
```

Q multiply -13 * -17 using modified Booth Algorithm.

16 8 4 2 1

13 → 01101	+17 → 010001
+13 → 001101	-17 → 101111
-13 → 110010	Multiplier ⇒ 101111
-13 → 110011	

BR ⇒ 110011

$\overline{BR}+1$ ⇒ 001101

101111 0

+1 0 0 0 -1

modified Multiplier

```
              1 1 0 0 1 1
            -1 1 0 0 0 -1
0 0 0 0 0 0 0 0 1 1 0 1
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 1 1
0 0 0 1 1 0 1
[1] 0 0 0 0 1 1 0 1 1 1 0 1
Discard   128 64 32 16 8 4 2 1
```

17
13
51
17
209

128
64
192
24 211

(37)

Modified Booth's Algorithm [Booth's Recoding] or Bit-pair recoding.

→ In modified Booth's algorithm recoded multiplier bit of Booths algorithm are paired due to which no. of multiplier bits are reduced.

+1	-1	→	+1
-1	0	→	-2
0	-1	→	-1
0	+1	→	+1
-1	+1	→	-1
+1	0	→	+2

(24)

Ex- multiplicand : 1101

2's compliment of multiplicand : 0010
+1
0011

Let we have 2 multiplier recoded but +1, -1 then

```
        1 1 0 1
          +1 -1
      ---------
  0 0 0 0 1 1
  1 1 1 0 1
-------------
(1 1 1 1 0 1)
 sign bit
```

```
        1 1 0 1
          +1 0
      ---------
  0 0 0 0 0 0
  1 1 1 0 1
-------------
  1 1 1 0 1 0
```

(25)

Date

Q) 110101 by 011011

BR = 110101
QR = 011011
$\overline{BR}+1$ = 001011

multiplier QR → 0 1 1 0 1 1
0 1 1 0 1 1 0
+1 0 -1 +1 0 -1
+2 -1 -1

1 1 0 1 0 1
+2 -1 -1
0 0 0 0 0 0 0 0 1 0 1 1
0 0 0 0 0 0 1 0 1 1
1 1 1 1 0 1 1 0
1 1 1 1 0 0 1 0 1 1 1 — Result

38

DIVISION ALGORITHM

Date

Quotient

14
12) 169 — Dividend
12
49
48
1 — Remainder

→ Division Algo. is used to divide the two binary numbers.

```
        00001110  → Quotient
1100 ) 10101001
       1100
       10010
        1100
       01100
         1100
       000001
       000000
       000001  — Remainder
```

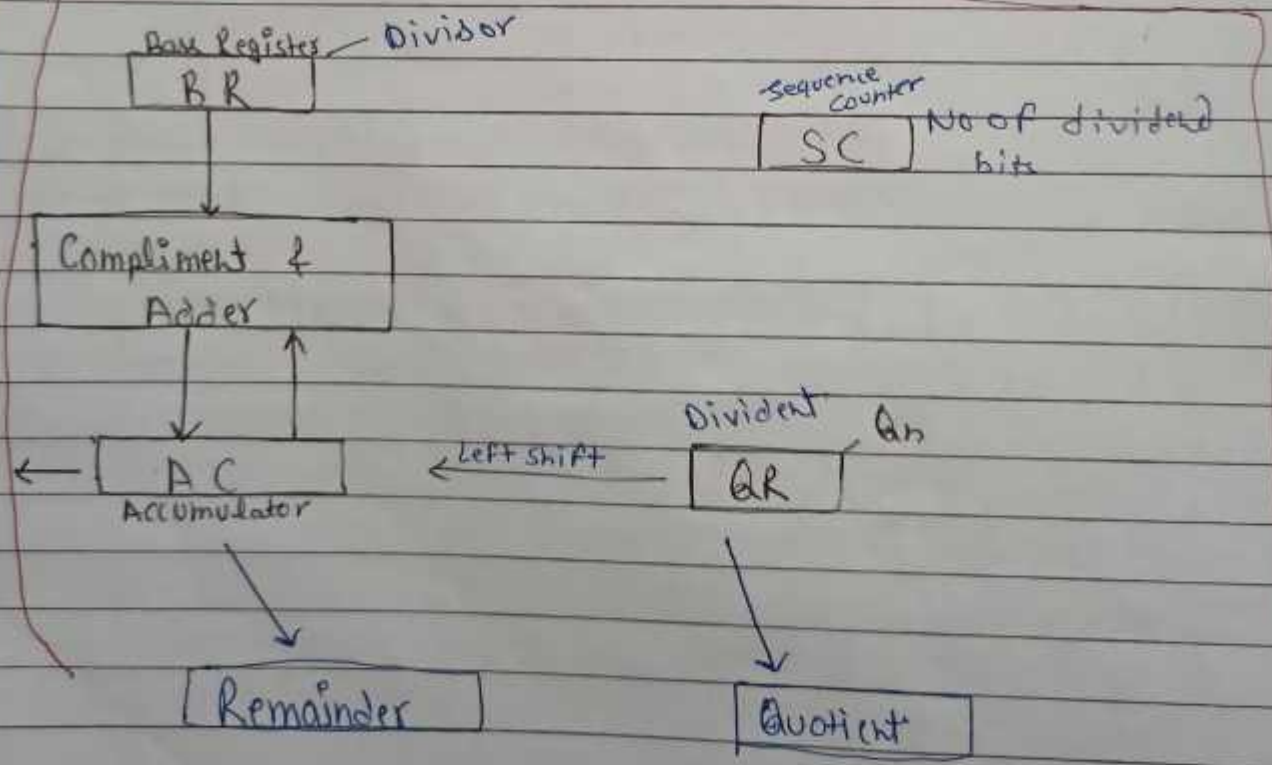
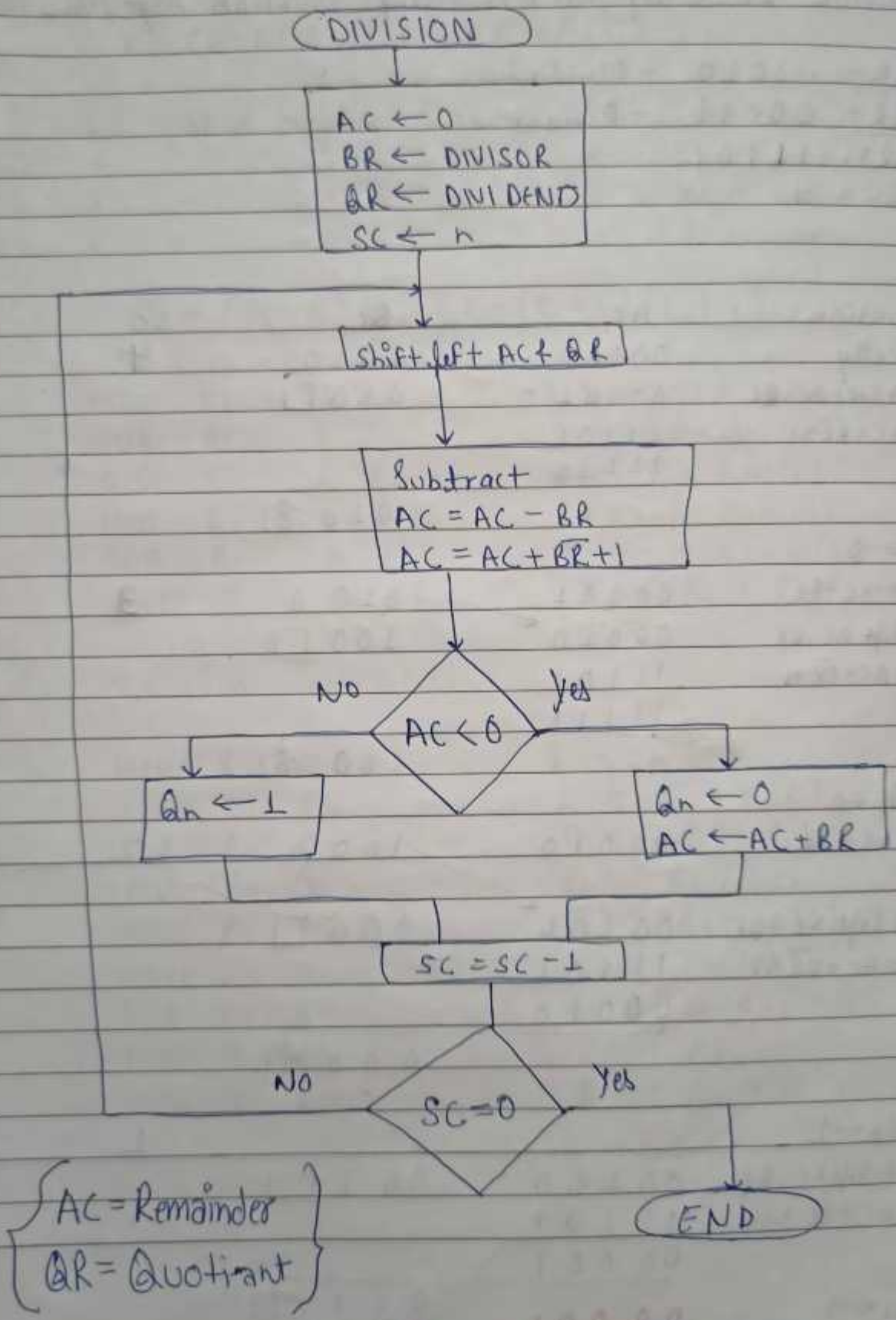


Fig :- H/w of Division Algorithm

(39)

FLOWCHART OF DIVISION ALGORITHM

Date 04/02/21



Date

26

Divide 1010 by 0011 using Division Algorithm.

QR - 1010 - Dividend
BR - 00011 - Divisor
$\overline{BR}+1$ - 11101
SC = 4

Operation	AC	QR	SC
Initially	00000	1010	4
shift left AC, QR	00001	010 [ ]	
AC = AC + $\overline{BR}+1$	11101		
	11110		
	00001	010 [0]	
$Q_0 \leftarrow 0$			
AC ← AC + BR	00001	0100	3
shift left AC, QR	00010	100 [ ]	
AC = AC + $\overline{BR}+1$	11101		
	11111		
	00011	100 [0]	
Set $Q_0 \leftarrow 0$			
AC ← AC + BR	00010	1000	2
shift left AC + QR	00101	000 [ ]	
AC = AC + $\overline{BR}+1$	11101		
	00010		
		000 [1]	
Set $Q_A \leftarrow 1$			1
Shift left AC, QR	00100	001 [ ]	
AC = AC + $\overline{BR}+1$	11101		
	00001		
		001 [1]	
Set $Q_1 \leftarrow 1$	00001	0011	0

(27)

WAP to evaluate arithmetic expression
$X = (A-B) * (((C-D*E)/F)G)$

i) using general register computer with 3 address instructions.
ii) " " " " " 2 " "
iii) " " " " " 1 " "
iv) " " " " " 0 " "

i) $X = (A-B) * (((C-(D*E)/F)G)$

MUL	R_1	D, E	$R_1 \leftarrow m[D] * m[E]$
SUB	R_2	R_1, C	$R_2 \leftarrow m[C] - R_1$
DIV	R_1	R_2, F	$R_1 \leftarrow R_2 / m[F]$
MUL	R_2	R_1 G	$R_2 \leftarrow R_1 * m[G]$
SUB	R_1	A, B	$R_1 \leftarrow m[A] - m[B]$
MUL	X	R_1, R_2	$X \leftarrow R_1 * R_2$

ii) 2 address instruction

MOVE	R_1	D	$R_1 \leftarrow m[D]$
MUL	R_1	E	$R_1 \leftarrow R_1 * m[E]$
SUB	R_1	C	$R_1 \leftarrow m[C] - R_1$
DIV	R_1	F	$R_1 \leftarrow R_1 / [F]m$
MUL	R_1	G	$R_1 \leftarrow R_1 * m[G]$
MOVE	R_2	A	$R_2 \leftarrow m[A]$
SUB	R_2	B	$R_2 \leftarrow m[A] - m[B]$
MOVE	X	R_2	$X \leftarrow R_2$
MUL	X	R_1	$X \leftarrow R_2 * R_1$

Date _____

iii) $X = (A-B) * (((C-(D*E))/F)G)$

LOAD D ;	AC ← m(D)
MUL E ;	AC ← m(D) * m[E]
SUB C ;	AC ← m[C] - AC
DIV F ;	AC ← AC / F
MUL G ;	AC ← AC * m[G]
STORE T ;	T ← AC
LOAD A ;	AC ← m[A]
SUB B ;	AC ← [m(A) - m(B)] AC - (B)
MUL T ;	AC ← AC * T
~~LOAD~~	
STORE X ;	X ← AC

iv) $X = AB- *(((CDE*-)/F)G)$

$X = AB-*(CDE*-F/G*)$

$AB-CDE*-F/G**X =$

PUSH A ;	TOS ← m[A]
PUSH B ;	TOS ← m[B]
SUB	TOS ← m[A] - m[B]
PUSH C ;	TOS ← m[C]
PUSH D ;	TOS ← m[D]
PUSH E ;	TOS ← m[E]
MUL ;	TOS ← m[D] * m[E]
SUB	TOS ← m[C] - (D*E)
PUSH F ;	TOS ← m[F]
DIV	TOS ← (C-(D*E))/F
PUSH G ;	TOS ← m[G]
MUL	TOS ← [(C-D*E)/F]*G
MUL ;	TOS ← (A-B)*[(C-D*E)/F)*G]
PUSH X ;	X ← TOS

28

Date

Q. Divide 1 1 1 by 0 1 1.

BR = 0 0 1 1

QR = 1 1 1

$\overline{BR}+1$ = 1 1 0 1

SC = 3

(1) shift left AC & QR

(2) AC = AC + $\overline{BR}+1$

check condition

Set $Q_0 \leftarrow 0$ [0]

$AC \leftarrow AC + BR$

Set $Q_0 \leftarrow 1$ [1]

check

Exit

Operation	AC	QR	SC
Initially	0 0 0 0	1 1 1	3
shift left AC & QR	0 0 0 1	1 1 □	
AC = AC + $\overline{BR}+1$	1 1 0 1		
	1 1 1 0		
		1 1 [0]	
Set $Q_0 \leftarrow 0$			
AC = AC + BR	0 0 1 1		
	Discard 1 0 0 0 1	1 1 0	2
shift left	0 0 1 1	1 0 □	
AC = AC + $\overline{BR}+1$	1 1 0 1		
	Discard 1 0 0 0 0		
		1 0 [0]	
Set $Q_0 = 1$	0 0 0 0	1 0 **1**	1
shift left	0 0 0 1	0 1 □	
$AC \leftarrow AC + \overline{BR}+1$	1 1 0 1		
	1 1 1 0		
		0 1 [0]	
Set $Q_0 \leftarrow 0$			
	0 0 1 1		
	0 0 0 1	0 1 0	0

Remainder ↑ Quotient ↑

29

By- Single Precision

Date

Q $(-307.1875)_{10} \Rightarrow -(100110011.00110)_2$

Normalized form convert → $-1.00110011001110 \times 2^8$

here, sign = -ve ⇒ 1

Actual Exponent (e) ⇒ 8

⇒ 8 + 127 ⇒ $(135)_{10}$

e ⇒ 10000111

2	135
2	67 1
2	33 1
2	16 1
2	8 0
2	4 0
2	2 0
2	1 0
	1 ↗

2	307
2	153 1
2	76 1
2	38 0
2	19 0
2	9 1
2	4 1
2	2 0
	1 0
	1 ↗

$307 \Rightarrow (100110011)_2$

•1875 X2 → 0.3750 → 0
•3750 X2 → 0.7500 → 0
•7500 X2 → 1.5000 → 1
•5000 X2 → 1.0000 → 1
•0000 X2 → .0000 → 0

$.1875 \rightarrow (0011)_2$

Mantissa → 00110011001100000000000

1 bit	8 bit	23 bit
1	10000111 (e)	00110011001100000000000 (mantissa)

(30)

Date 05/01/24

Double Precision. $-(-307.1875)_{10}$

Normalized form Convert → $-1.0011001100110 \times 2^8$

here sign = -ve ⟹ 1

Actual Exponent = 8

= 8 + 1024 ∴ 1032

$(1032)_{10} \therefore (10000001000)_2$

2	1032
2	516 0
2	258 0
2	129 0
2	64 1
2	32 0
2	16 0
2	8 0
2	4 0
2	2 0
	1 0
	1

mantissa = 0011001100110000000000000000000000000000000000

Sign	Exponent	mantissa
1 bit	11 bit	52 bit

1 bit 11 bit 52 bit

1 | 10000001000 | 0011001100110...

Sign Exponent mantissa

(31)

Date

Q Represent $(128.25)_{10}$ in double Precision

$128 \rightarrow 10000000$

$.25 \rightarrow 010$

$.25 \times 2 \rightarrow 0.50 \rightarrow 0$

$.50 \times 2 \rightarrow 1.00 \rightarrow 1$

$.00 \times 2 \rightarrow 0.00 \rightarrow 0$

2	128
2	64 0
2	32 0
2	16 0
2	8 0
2	4 0
2	2 0
	1 0
	1 ↗

Single Precision

$(128.25)_{10} \rightarrow (10000000.010)_2$

$\rightarrow (1.0000000010) \times 2^7$

Sign → +ve → 0

Actual Exponent = 7

$= 7 + 127 \Rightarrow 134$

$(134)_{10} \rightarrow (10000110)_2$

0	10000110	00000000100000000000000

1 bit | 8 bit | 23 bit

2	134
2	67 0
2	33 1
2	16 1
	8 0
	4 0
	2 0
	1 0
	1 ↗

double Precision

actual exponent $= 7 + 1023$

$= 1030$

$1030 \rightarrow (10000011111)_2$

Sign → +ve → 0

Exponent → 10000011111

Mantissa → 00000000100000000000000000000000000000000000000000000

2	1023
2	511 1
2	255 1
2	127 1
2	63 1
2	32 1
2	16 0
2	8 0
2	4 0
2	2 0
	1 0
	1 ↗

Sign	exponent	mantissa
0	10000011111	00000000100000000000000000000000000000000000000000000

1 bit | 11 | 52 bit

(32)

Date

Q Represent the following number IEEE 754 floating point representation of 32 bit & 64 bit format.

(a) -1.75

$(-1.75)_{10} \Rightarrow -1.110 \times 2^0$

$.75 \times 2 \rightarrow 1.50 \rightarrow 1$
$.50 \times 2 \rightarrow 1.00 \rightarrow 1$
$.00 \times 2 \rightarrow 0.00 \rightarrow 0$

exponent → 0
→ 0 + 127 ⇒ 127
exponent → 10000011
sign bit → -ve → 1

mantissa → 110

2	127
2	63
2	31
2	15
2	7
2	3
2	1
	1

Single Precision (32 bit format)

1bit	8bit	23 bit
1	10000011	11000000000000000000000

Double Precision (64 bit format)

2	1023
2	511
2	255
2	127
2	63
2	31
2	15
2	7
2	3
2	1
	1

Signbit → -ve → 1
exponent → 0 + 1023
→ 10000000001

mantissa → 11000000000000000000000000000000000000000000000000

1bit	11bit	52 bit
1	10000000001	11000000000000000000000000000000000000000000000000

Date

(3) +21 → 10101

→ 1.01012 X2^4

Sign bit → +ve → 0

exponent → 4+127 ⇒ 131

2	131
2	65 1
2	32 1
2	16 0
2	8 0
2	4 0
2	2 0
	1 0
	1

2	21
2	10 1
2	5 0
2	2 1
2	1 0
	1

exponent → 10000011

mantissa → 01010000000000000000000

1bit	8bit	23bit
0	10000011	01010000000000000000000
sign	exponent	mantissa

Double Precision (64 bit)

Sign → +ve → 0

exponent ⇒ 4 + 1023

⇒ 1027

exponent ⇒ (11..11100 01)

2	1027
2	513 1
2	256 1
2	128 0
2	64 0
2	32 0
2	16 1
2	8 1
2	4 1
2	2 1
	1

mantissa → 0101000000000 .....

1bit	11bit	52bit
0	11111100 01	0101000000000000000000000000000000000000000000000000

23

Date

$11\,0101\,01 \cdot 0101$

$\Rightarrow 1.1010101010 \times 2^7$

exponent $\Rightarrow 7$

$\Rightarrow 7+127 \Rightarrow 134$

exponent $\Rightarrow 10000110$

Signbit $\Rightarrow$ +ve $\rightarrow 0$

mantissa $\rightarrow 10101010100000000000$

2	134	
2	67	0
2	33	1
2	16	1
2	8	0
2	4	0
2	2	0
	1	0
	1	↗

0	10000110	10101010100000000000000
1bit	8bit	23bit

Double Precision

$\Rightarrow 1.1010101010101 \times 2^7$

- exponent $\rightarrow 7$

$\Rightarrow 7+1023$

$\Rightarrow 1030$

exponent $\rightarrow 10000000110$

mantissa $\rightarrow 1010101010100000000 \cdots$

2	1030	
2	515	0
2	257	1
2	128	1
2	64	0
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
	1	0
	1	↗

sign bit / exponent / mantissa

1bit	11 bit	52 bit
0	10000000110	1010101010000000000000 ⋯

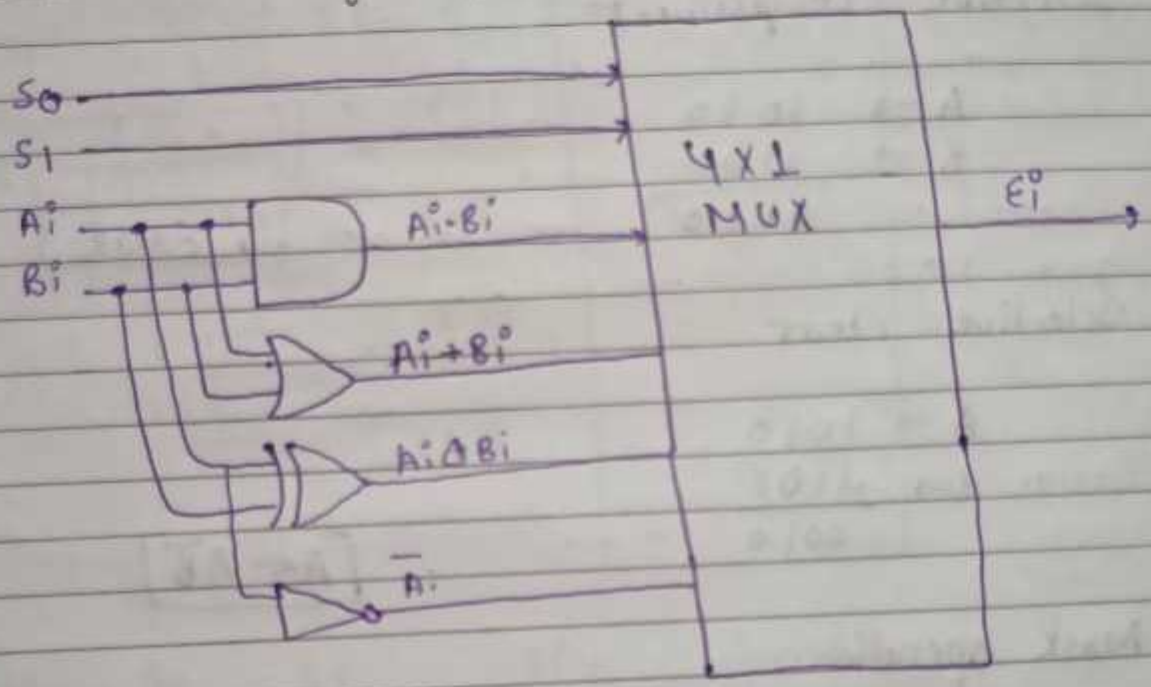
(46)

Date

Logic Operation

Q) Design logic circuit that perform four basic logic operations AND, OR, NOT, XOR.

Then 16 logic operation perform in CPU.



S_0	S_1	E_i	Operation
0	0	$A \vee B$	OR
0	1	$A \wedge B$	AND
1	0	$A \oplus B$	XOR
1	1	$\bar{A}$	Compliment

(4) Application of Logic operation

Date _____

i) what is Selective Set

Set → 1
Reset → 0

A → 1010
1100
1110

OR

ii) Selective Complement

A → 1010
B → 1100
0110

EXOR

iii) Selective clear

A → 1010
B → 1100
0010

$A \leftarrow A\bar{B}$

iv) Mask operation

A → 1010
1100
1000

AND

v) clear operation

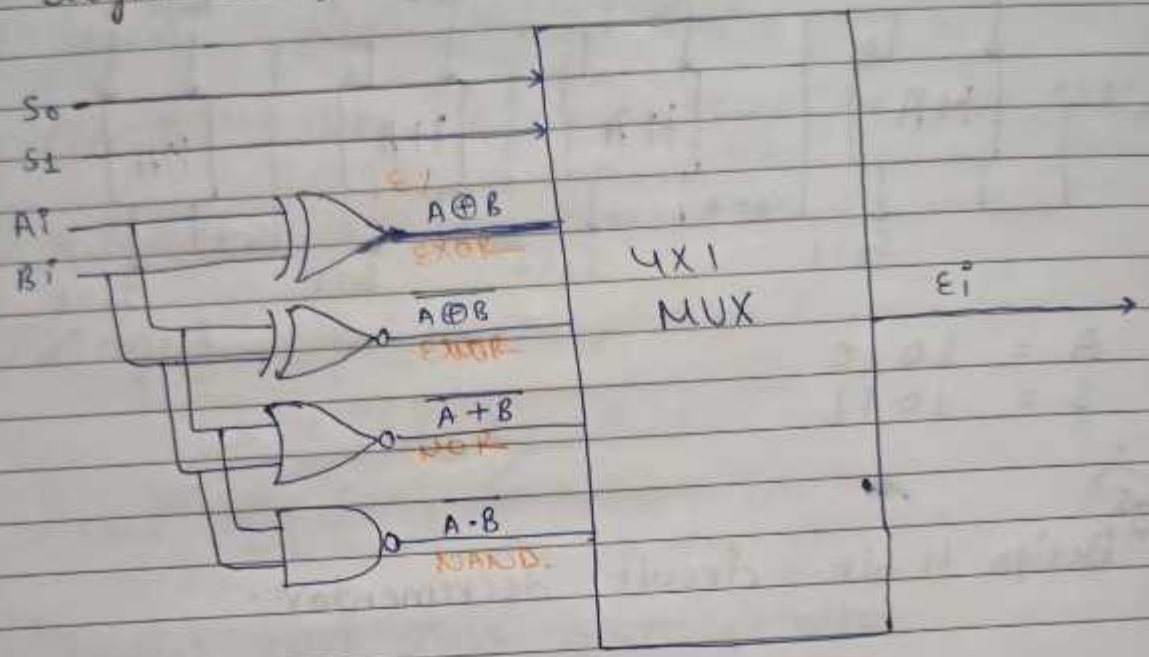
A → 1010 → Compare both then =0
B → 1100
1010

$A \leftarrow A \cdot B$

Q4

Date

Q) Design Digital circuit that Perform the logic operation Exclusive OR, Exclusive NOR, NOR & NAND used to Selected variable So logic diagram of 1 typical Stage.



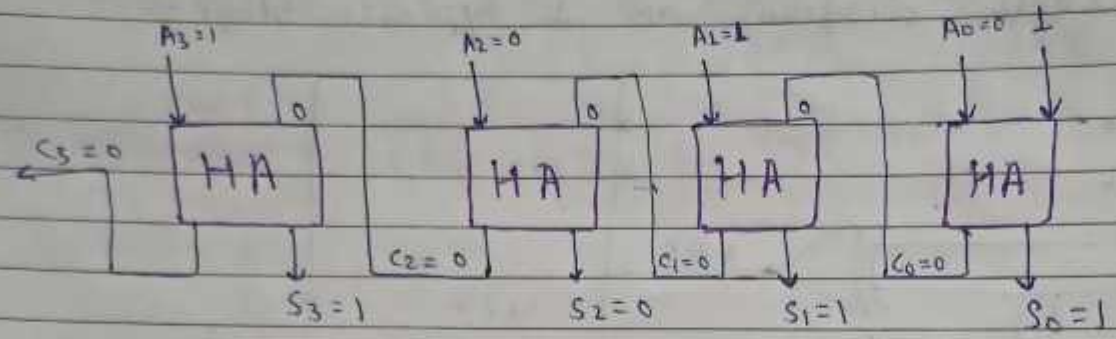
S0	S1	Ei	Operation
0	0	$A \oplus B$	EOR
0	1	$\overline{A \oplus B}$	ENOR
1	0	$\overline{A+B}$	NOR
1	1	$\overline{A \cdot B}$	NAND

42

Date

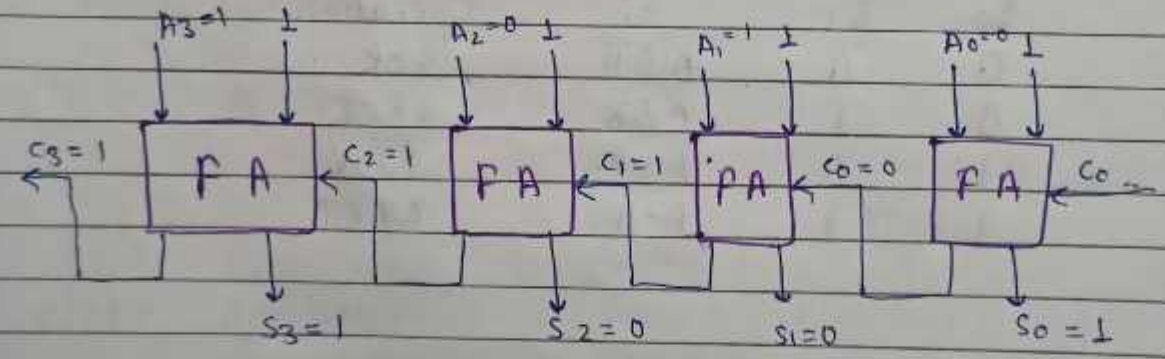
Binary Increment

1) Design 4 bit binary incrementor.



A = 1010
S = 1011

2) Design 4 bit circuit decrementor.



A = 1010
S = 1001
C = 1110

43

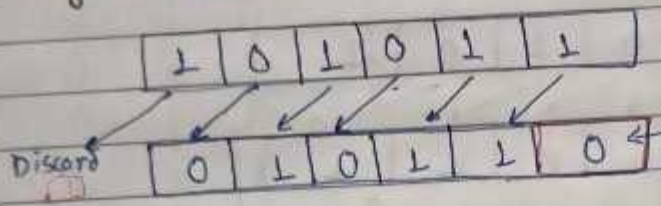
Date

Design of Arithmetic logic & Shift unit [ALU]

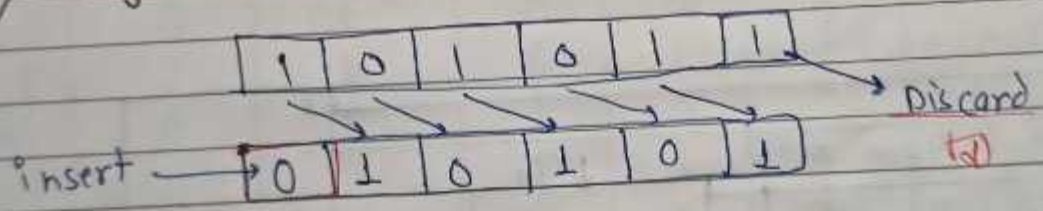
a) Shift Microoperation

1) logical shift micro-operation
2) Circular shift microoperation
3) Arithmetic shift microoperation

1) Logical shift left microoperation

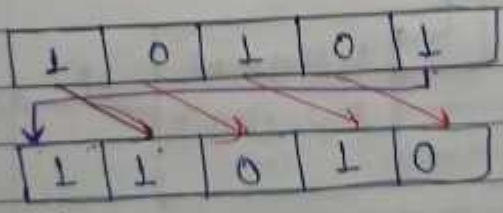


2) Logical shift right micro-operation.



b) Circular shift micro-operation

1) circular shift left microoperation.

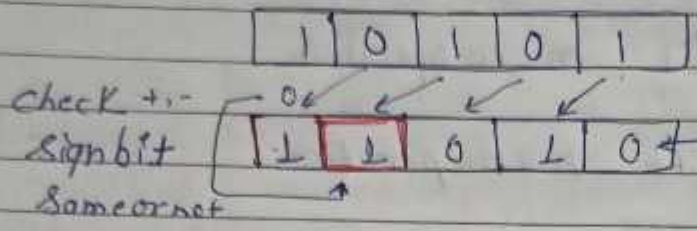


2) Circular shift right micro-operation.

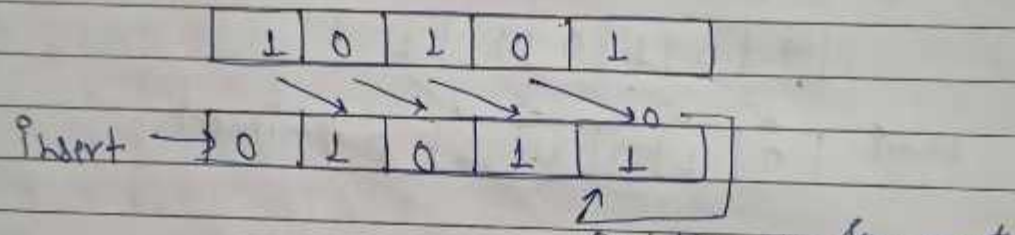
Date

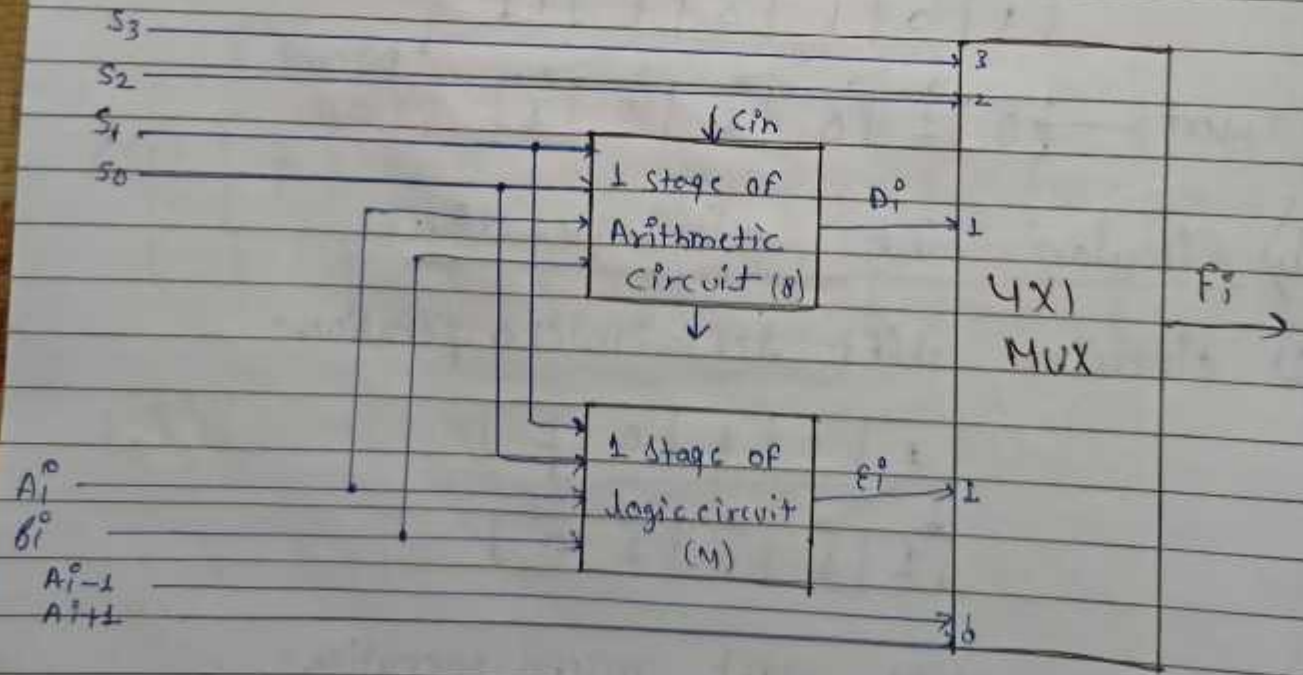
6) Arithmetic shift micro-operation

1) Arithmetic shift left micro-operation



2) Arithmetic shift Right micro-operation





Date _____

S_0	S_1	F_i
0	0	D_i
0	1	E_i
1	0	shift left
1	1	shift right

	S_3	S_2	S_1	S_0	C_{in}	Operation F_i
						Function
Arithmetic	0	0	0	0	0	F=A Transfer
	0	0	0	0	1	F=A+1 Increment
	0	0	0	1	0	F=A+B Add
	0	0	0	1	1	F=A+B+1 Add with carry / Sub with borrow
	0	0	1	0	0	F=A+$\bar{B}$ / subtract
	0	0	1	0	1	F=A+$\bar{B}$+1 / Decrement
	0	0	1	1	0	F=A-1 / Transfer
	0	0	1	1	1	F=A / AND
Logic	0	1	0	0	x	F=A∧B / OR
	0	1	0	1	x	F=A∨B / EXOR
	0	1	1	0	x	F=A⊕B / NOT/compla
	0	1	1	1	x	F=$\bar{A}$
Shl	1	0	X	x	x	F=shl shift left
Shr	1	1	X	x	x	F=shr shift right