Date 27/9/19.

Unit-3.

Types of Inst^n :- 0, 1, 2, 3 Add Inst^n

1. Check for zero. # Inst^n Code :-
2. Add the exponent & check for overflow.
3. Multiply the Mantissa.

Instruction Cycle :-

Inst^n cycle specifies a set of steps to process an instruction.

Major Sub-cycles of Inst^n cycle :

1. fetch cycle 2. Execute cycle.

Steps or Procedure

Steps in fetch cycle:
1. Fetch an Inst^n
2. Decode the opcode of Inst^n.
3. Evaluate effective address of operand.
4. Fetch operands from memory if any.

Steps in execute cycle:
5. Execute Inst^n (Processing by ALU).
6. Store the Result in Memory.

1 2 4 8
2 0 4 8
1 0 2 4

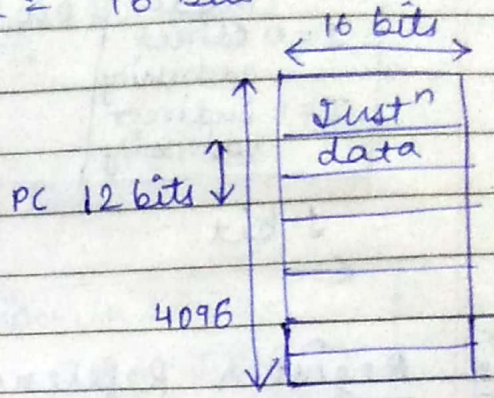
Date

Let us assume,

* Basic Computer Memory size = 4096×16
$= 2^{12} \times 16$ bits

add = 12 bits

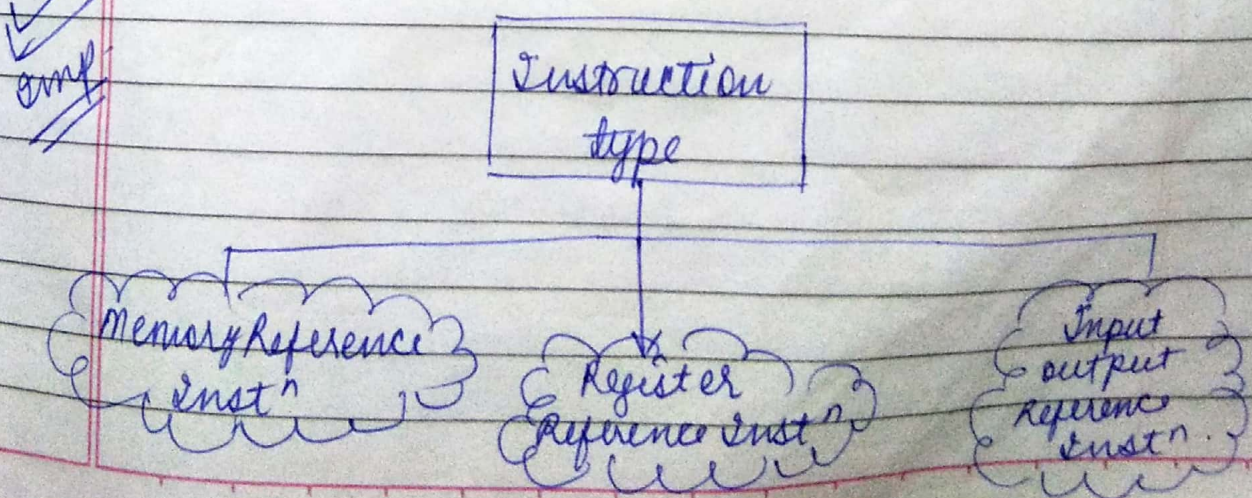
Memory word/data = 16 bits



Computer Register used :-

1. IR → Instruction Register
2. DR → Data Register → 16 bits.
3. AR → Address Register → 12 bits
4. PC → Program Counter → Address of next Inst^n.
5. AC → Accumulator Register
6. TR → Temporary Register.
7. INPR → Input Register → 8 bits
8. OUTR → Output Register → 8 bits.

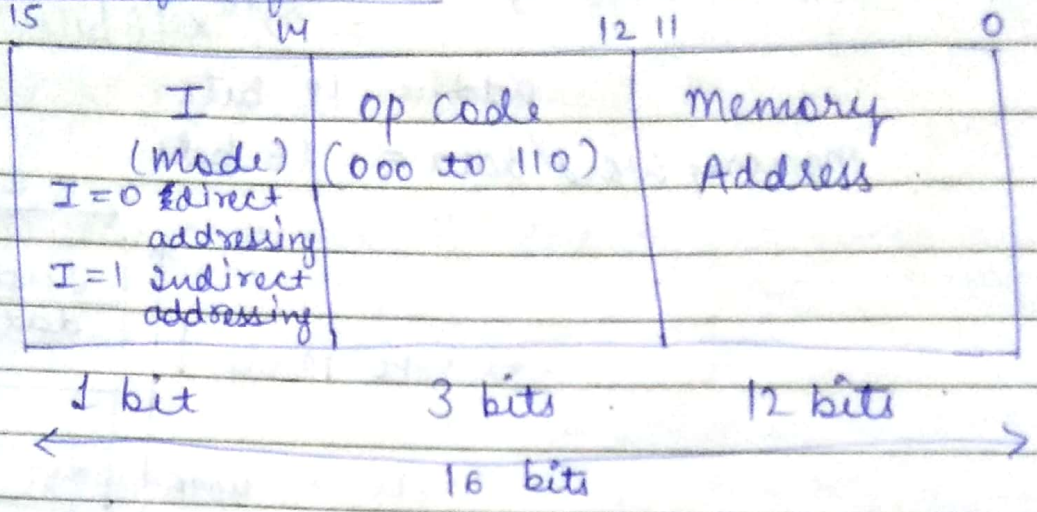
Instruction Cycle :-



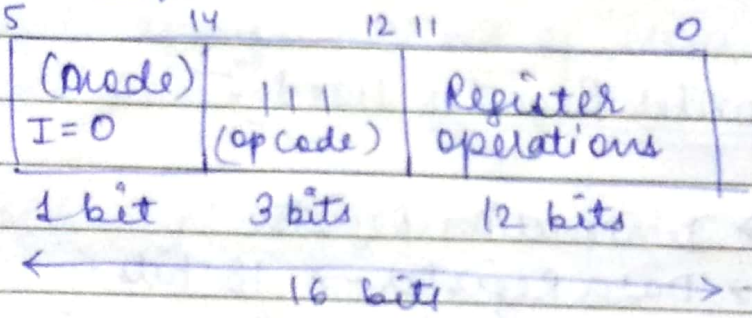
Date

Types of Inst^{n} cycle :-

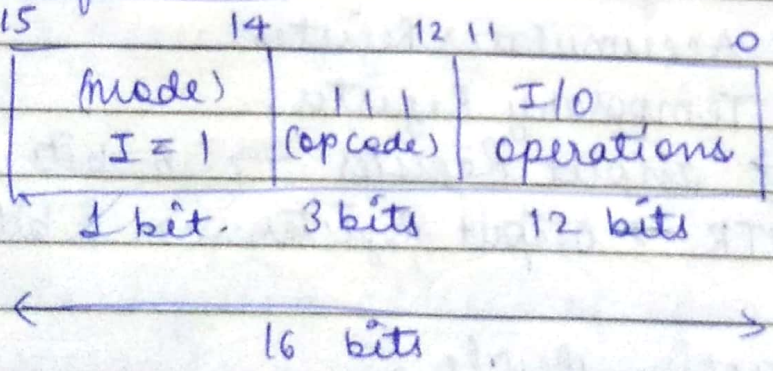
1. Memory Reference :-



2. Register Reference :-



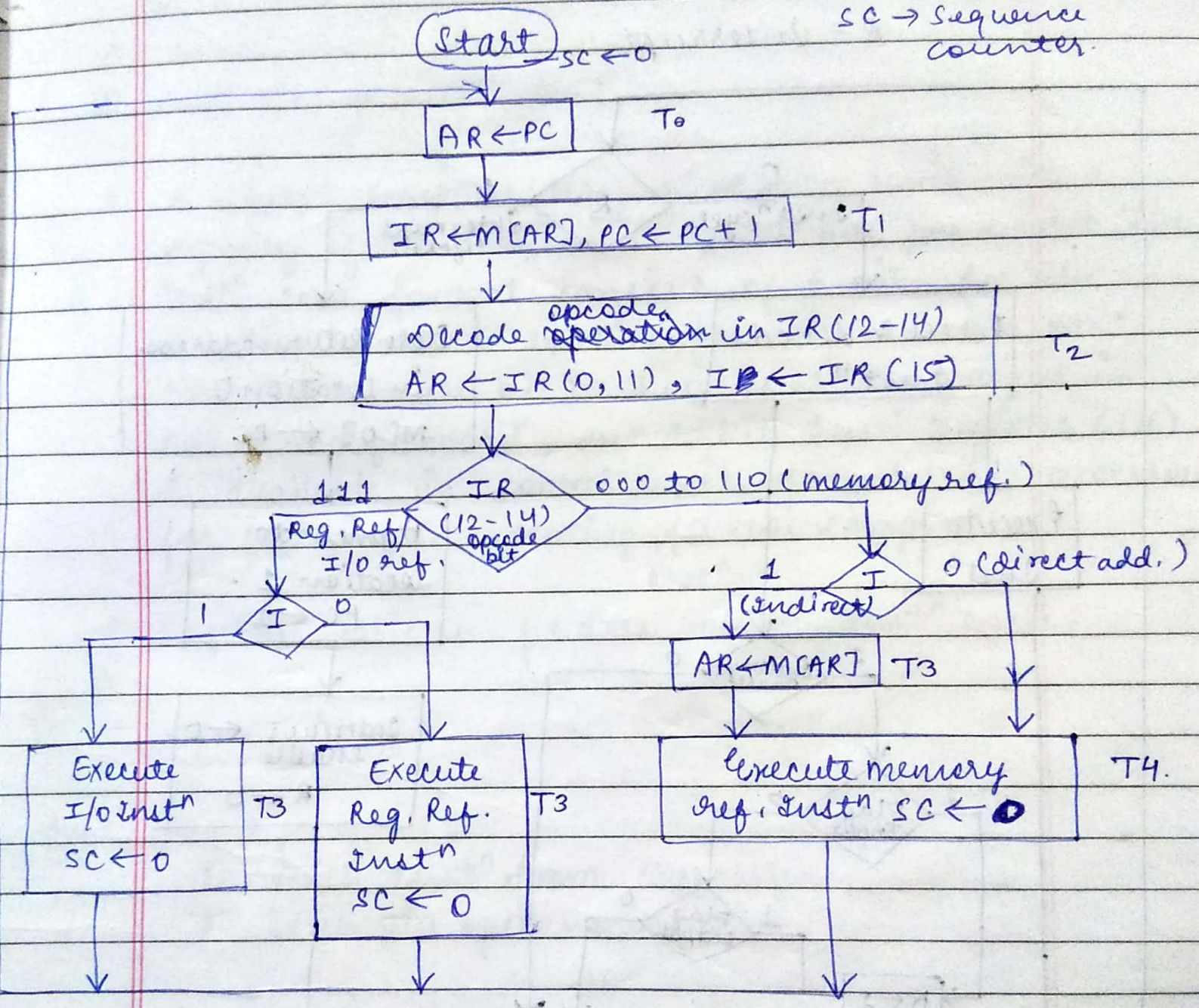
3. I/O Reference :-



Date 30/9/19.

imp

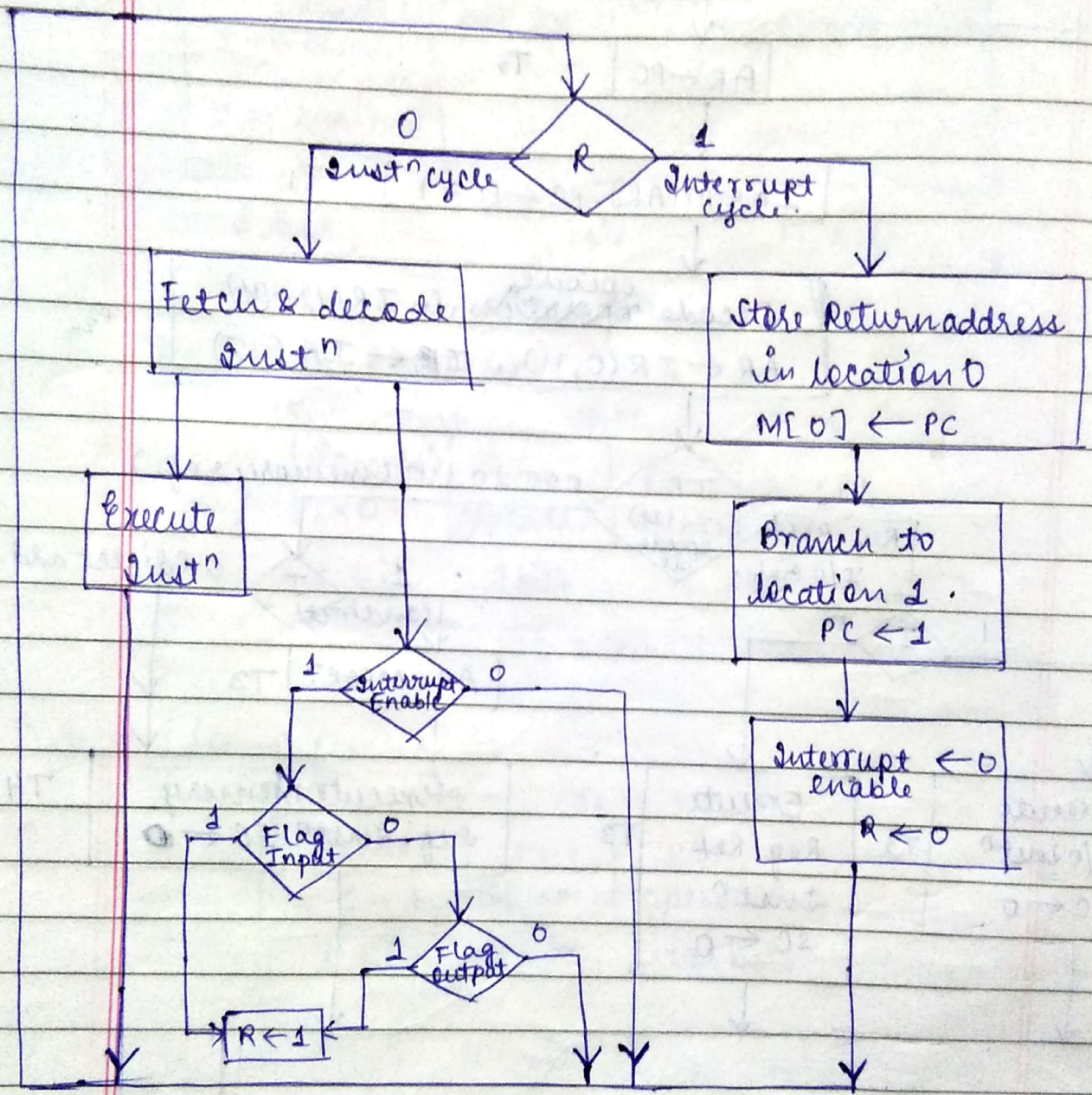
Flowchart of Basic Instruction Cycle :-

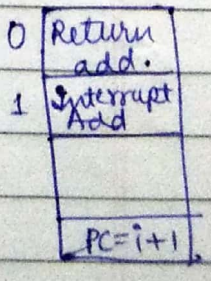


Date _____

Flowchart for Interrupt cycle :-

R = Interrupt.





Date 9.10.19

1. How many references to memory are needed to bring an operand into a processor register for
(i) Direct Address Inst^n.
(ii) Indirect Address Inst^n.

2. A digital computer has a memory unit with a capacity of 16,384 (or 16K) words, 40 bits per word. The Inst^n code format consist of 6 bits for the operation part & 14 bits for the address part (no direct mode bit). Two Inst^n are packed in one memory word, and 40 bit Inst^n register (IR) is available in control unit. formulated a procedure for fetching & executing Inst^n for this computer.

3. Explain various phases of an Inst^n cycle.

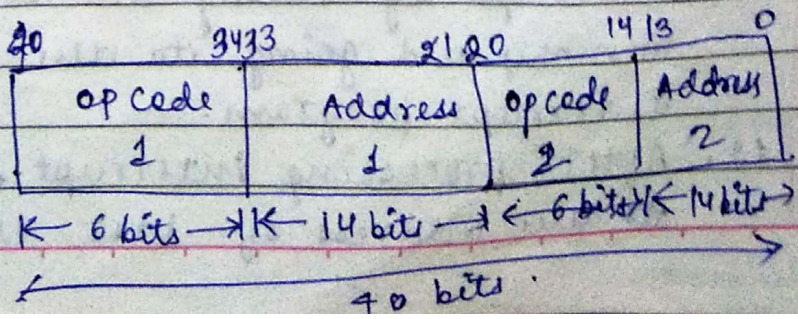
2018-19.

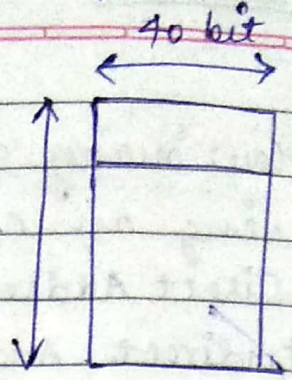
Answer:

Ans.1 (i) 2
1. Fetch Inst^n from memory.
2. Fetch the operand.

(ii) 3.
1. Fetch Inst^n from memory.
2. To determine Effective address of operand.
3. Fetch the operand.

Ans.2





$= 16K$
$= 2^4 \times 2^{10}$
$= 2^{14}$

address bits = 14.

Steps:

1. Fetch 40 bits Inst^n from main memory.
2. Save this Inst^n in IR.
3. Decode operation part 1 of Inst^n 1 using 6×2^6 decoder.
4. Fetch operand of Inst^n 1.
5. Execute Inst^n 1.
6. Decode op part 2 of Inst^n 2 using 6×2^6 decoder.
7. Fetch operand of Inst^n 2.
8. Execute Inst^n 2.

Ans.3

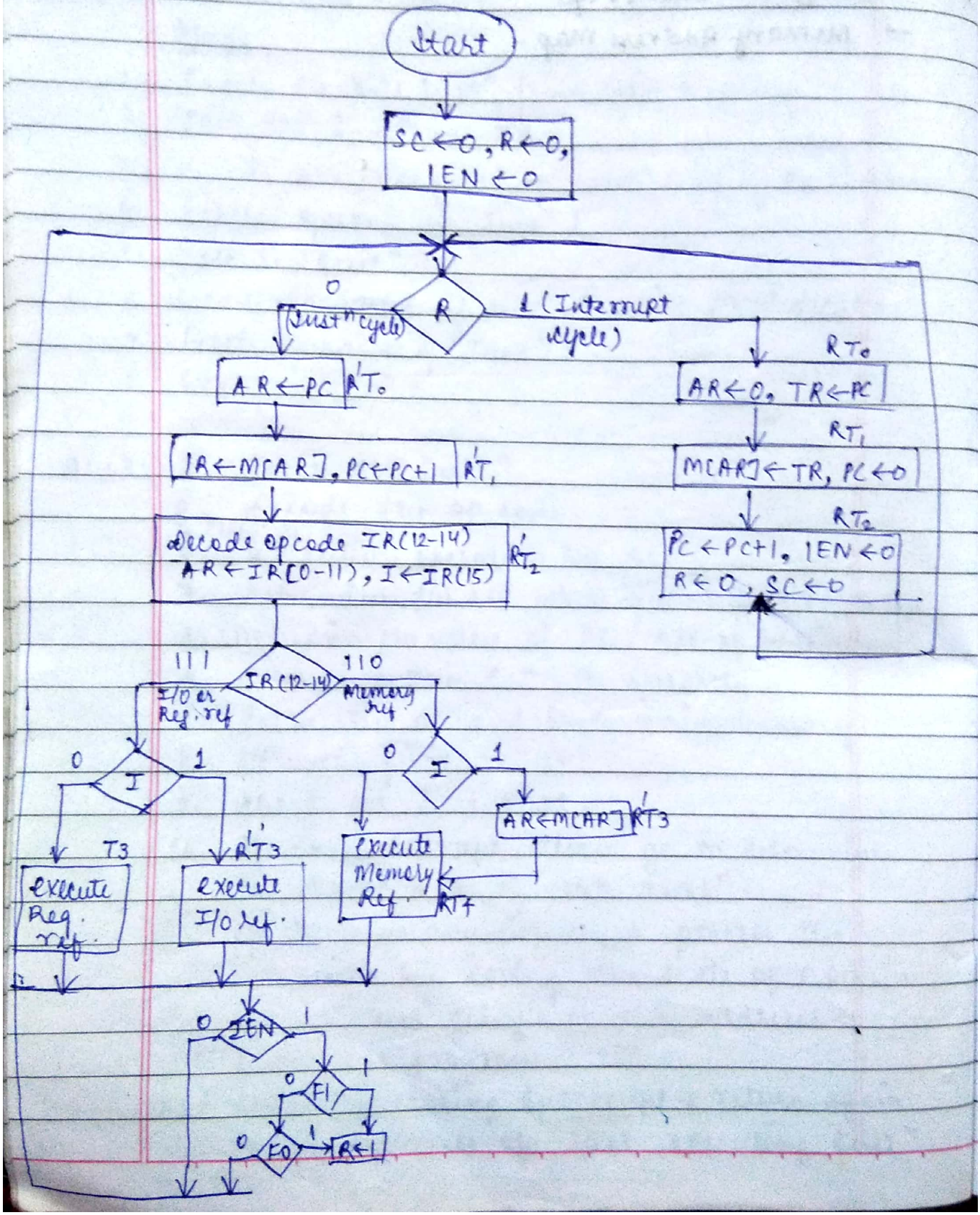
2. Fetch the Inst^n
3. Decode the op code
6. Perform execution operation by ALU.
7. Determine the Add. where resulting Operand is to be stored
1. Determine the value of PC / Add. of Inst^n.
4. Determine the E.A. of operands.
5. Fetch the operand from main memory.
8. Store the final result.
9. Check for interrupt.
10. If no interrupt then go to determine the Inst^n add. of next Inst^n.
11. If there is any interrupt process the interrupt by saving the state of CPU in memory and going to the address of interrupt program.
12. After processing interrupt, return again to the address of last executing Inst^n.

Date 20/10/19

Execution of Complete Inst^n cycle)

Complete Computer Description :-

Flowchart Of Complete Inst^n Cycle :



Date 10/10/19

Status flag / Status Bit Cond" / Cond" Codes :

After each ALU operation, the status of CPU is represented by status bits or cond" codes. It is stored in a register called Status Register.

Common Status Flags :

1. Sign Bit (S) — S ← 0, +ve; S ← 1, -ve
2. Zero Bit (Z) — Z ← 1, zero; Z ← 0, non-zero.
3. Carry Bit (C) — C ← 0, if no carry generates; C ← 1, if carry generates.
4. Overflow (V) — V ← 1; overflow; V ← 0; no overflow.

→ $MSB \oplus (MSB-1)$

Ques.

$$\begin{array}{r} 1011 \\ +1011 \\ \hline 10000 \end{array}$$

C ← 1

Z ← 1

Ques.

$$\begin{array}{r} 1011 \\ +1011 \\ \hline 10000 \end{array}$$

MSB, MSB-1

$V = 0 \oplus (0-1)$

$V = 0 \oplus 1$

$V = 1$

overflow.

Ques.

$$\begin{array}{r} 1011 \\ +0100 \\ \hline 1111 \end{array}$$

Z → 0

S → 1

C → 0

V → 0

$V = MSB \oplus (MSB-1)$

$= 1 \oplus (1-0)$

$= 1 \oplus (1+0)$

$V = 0$

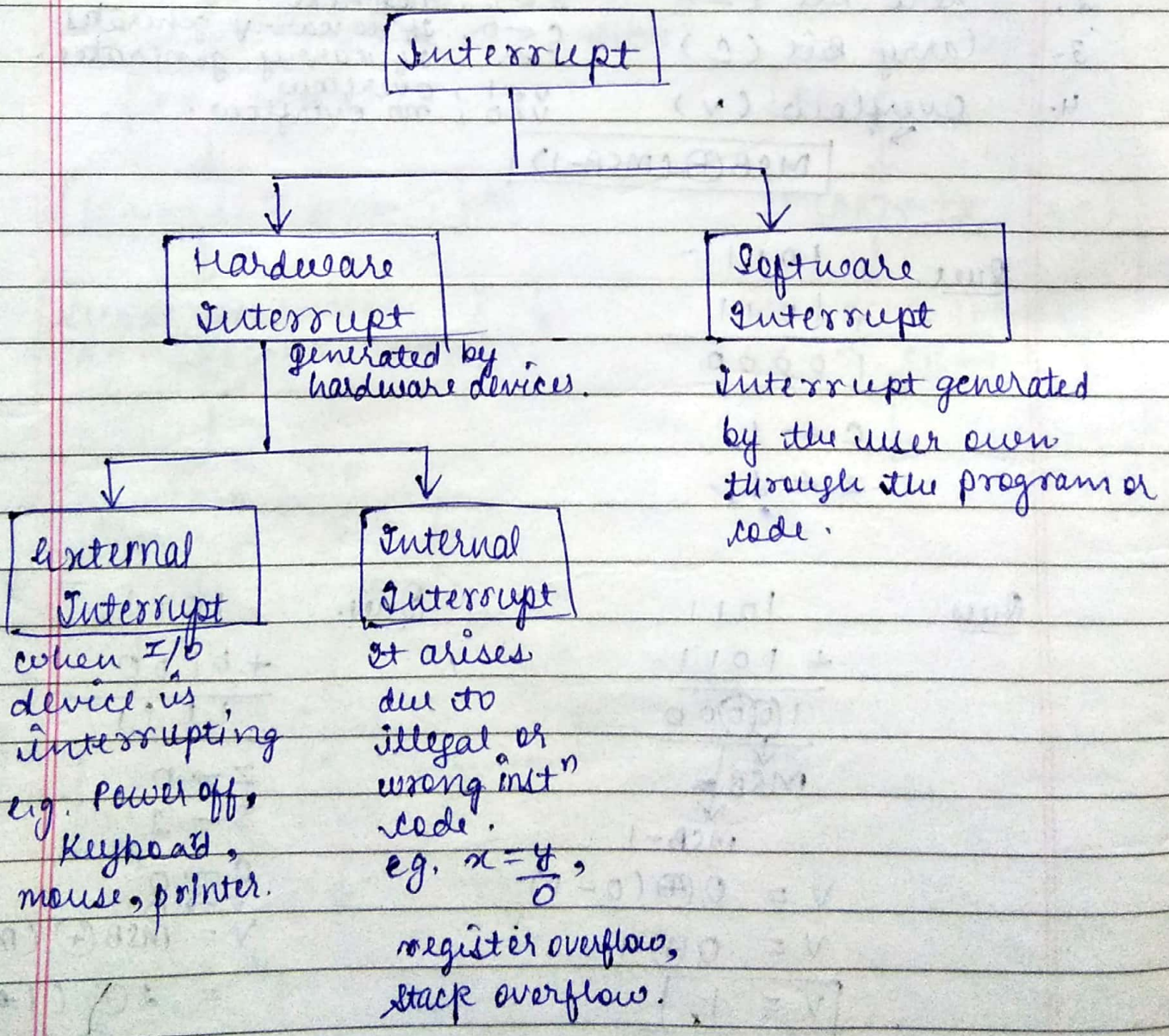
Date _____

Program Interrupt :-

A signal which temporary suspends the normal execution of an Inst^n cycle and executes its own functions / Sub-routine.

function is a small program that performs specific task.

Types of Program Interrupt :-



Program Control :

↳ status flag
↳ Interrupt

It will change the normal execution sequence of instruction. It affects the program counter badly. They can be -

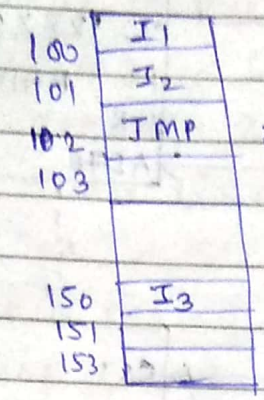
→ conditional.
→ unconditional.

Types of PCI :-

Program control Instⁿ	Symbol
1. Branch	BR
2. Jump	JMP
3. Skip	SKP
4. Call	CALL
5. Return	RETURN
6. Compare	CMP
7. Test	TST

Date

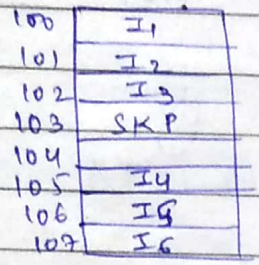
1. Branch & Jump Inst^n :-



Unconditional
JMP = 150
BR 150

Conditional
JC 150 C = 1
(Jump if Carry)
JP 150 S = 0
(Jump if positive no.)
JNC 150 C = 0
(Jump NO Carry)
JM 150 S = 1
(Jump Minus/Neg. No.)
JZ 150 Z = 1
(Jump zero).
JNZ Z = 0
(Jump no. zero).
JV V = 1
(Jump if overflow)
JNV V = 0.
(Jump no overflow).

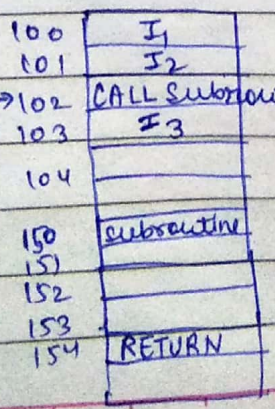
2. Skip Inst^n :-



* It causes to skip the next inst^n.

SKP Z

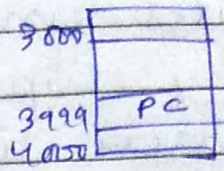
3. Call and Return Inst^n :-



CALL 150
↓
starting address of subroutine (function).

Date

Stack in memory.



CALL operations.

1. SP ← SP − 1
2. M[SP] ← PC.
3. PC ← Subroutine start address.

RETURN operation.

1. PC ← M[SP]
2. SP ← SP + 1.

4. Compare Instⁿ :-

CMP A, B.

It always compare two operands with the help of status bits. It is done by the subtraction of two no.

eg CMP 15, 30
→ CMP 30.

5. Test Instⁿ :-

TST A, B.

It AND the two no's and change its status flag.

TST 1100, 0000

→ Z = 1

Instruction Code :

A group of bits used in the instruction, used to specify what to do.
instruct computers to perform specific operation.

recalled as macrooperation as specifies set of micro-operations

1. Operation Code :- Specifies the operation to be performed by that instruction such as add, subtract, shift, complement, multiply.

2. Address Field(s) :- On what operation should be performed either stored in memory address or in register address. ~~it specifies the rule for interpreting or modifying address~~

3. Mode field → it specifies how to access the address of operands.

Instruction format :-

mode	op code	address field(s)

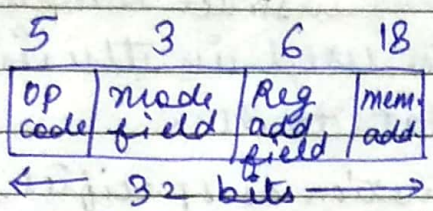
Ques. 1. The memory unit of a computer has 256 K words of 32 bit each. The computer has an instruction format with 4-fields, ① an op code field, ② a mode field to specify one of 7 addressing modes ③ a register address field to specify one of 60 processer register and ④ a memory address. Specify the instruction format & the no. of bits in each field if the inst^n is in 1 memory word.

Sol. (ii) mode field - 7 addressing modes = 2^3 = 3 bits

(iii) Reg. add. field - 60 processor reg = 2^6 = 6 bits

(iv) Memory add. = 256K = $2^8 \times 2^{10}$

= 2^{18} = 18 bits



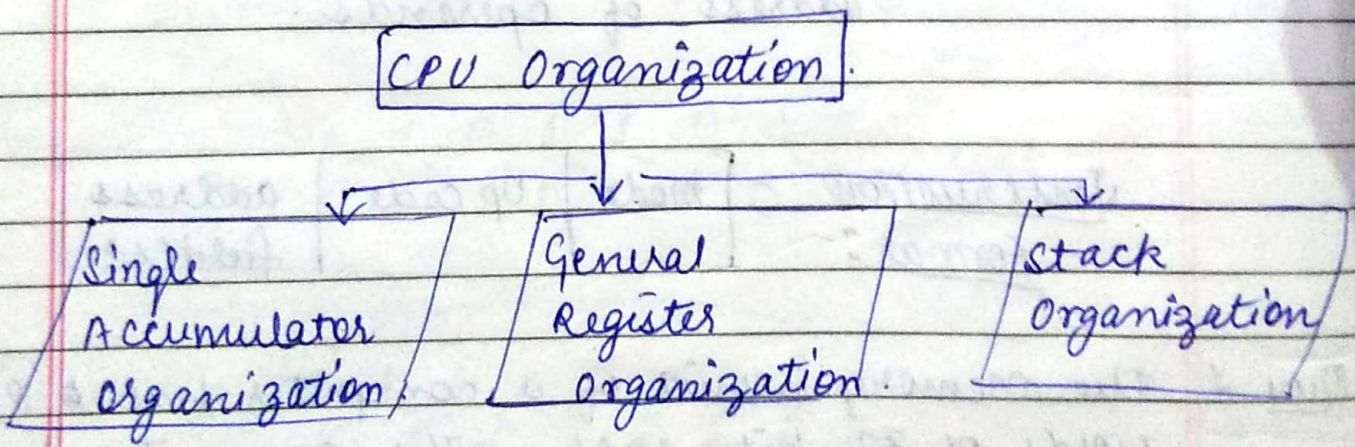
(i) OP code = 32 - (3+6+18)

= 32 - 27

= 5 bits.

Types of Instructions :-

It is Based on CPU organization.



1. Single Accumulator Organization :-

* In each operation there should be a use of single accumulator Organization. It is a specialised register itself.
* It has one address field.
* denoted by AC → Accumulator Register.

2. General Register Organization :

* No specialised register.
* 2-address field or 3-address field.

3. Stack Organization :

* There are two operations performed i.e. push and pop (deletion)
* Work on LIFO.
* It has 0 address field.
* Storage 1. Memory stack : portion of main memory.
  2. Register stack : set of reg. work on concept of stack.

Ques 1 Solve the following arithmetic operation.

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

1) 3-address inst"
2) 2-address inst"
3) 1-address inst"
4) 0-address inst".

memory add.

(D+C)*(B+A)
stack

Sol. 1.

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

2.

```
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]
```

Date

MOV X, R1 [M[X] ← R1].

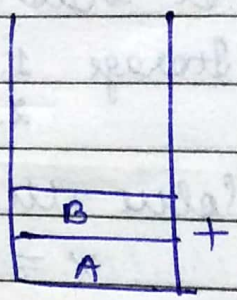
3.

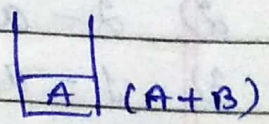
Accumulator Register ← AC.

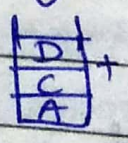
Load	A	AC ← M[A].
ADD	B	AC ← AC + M[B].
Store	T	M[T] ← AC.
Load	C	AC ← M[C].
ADD	D	AC ← AC + M[D].
MUL	T	AC ← AC * M[T].
Store	X	M[X] ← AC.

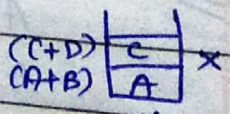
4.

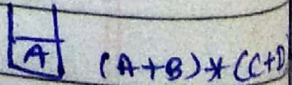
PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← A · (A+B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C+D)
MUL		TOS ← [(A+B) * (C+D)]
POP	X	M[X] ← TOS.











Date

8.12

Ques 2. WAP to evaluate the arithmetic statement

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

Sol. (1) 3 address Instⁿ

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

(2) 2 address Instⁿ

```
MOV  R1, A     [R1 ← M[A]]
SUB  R1, B     [R1 ← R1 - M[B]]
MOV  R2, D     [R2 ← M[D]]
MUL  R2, E     [R2 ← R2 * M[E]]
SUB  R2, F     [R2 ← R2 - M[F]]
MUL  R2, C     [R2 ← R2 * M[C]]
ADD  R1, R2    [R1 ← R1 + R2]
MOV  R3, H     [R3 ← M[H]]
ADD  R3, G     [R3 ← R3 + M[G]]
MUL  R3, K     [R3 ← R3 * M[K]]
DIV  R1, R3    [R1 ← R1 / R3]
MOV  X, R1     [M[X] ← R1]
```

Date

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

3. 1 adder Instruction :-

LOAD A	[AC ← M[A]]
Sub B	[AC ← AC − M[B]]
Store T	[M[T] ← AC]
Load D	[AC ← M[D]]
MUL E	[AC ← AC * M[E]]
Sub F	[AC ← AC − M[F]]
MUL C	[AC ← M[C] * AC]
Store Add T	[AC ← AC + M[T]]
Store T	[M[T] ← AC]
Load H	[AC ← M[H]]
MUL K	[AC ← AC * M[K]]
Add G	[AC ← AC + M[G]]
~~Store T1~~	~~[M[T1] ← AC]~~
~~Load T~~	~~[AC ← M[T]]~~
Div T1	[AC ← AC / M[T1]]
Store X	M[X] ← AC

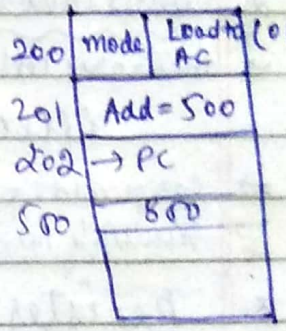
4. 0 adder Inst" :-

PUSH A	TOS ← A
PUSH B	TOS ← B
Sub	TOS ← (A − B)
Push C	TOS ← C
Push D	TOS ← D
Push E	TOS ← E
MUL	TOS ← (D * E)
Push F	TOS ← F
Sub	TOS ← (D * E) − F)
MUL	TOS ← C * ((D * E) − F)
ADD	TOS ← (A − B) + C * (D * E) − F))

Date _____

PUSH G	Tos ← G
Push H	Tos ← H
Push K	Tos ← K
MUL	Tos ← (H*K)
ADD	Tos ← G+(H*K)
DIV	Tos ← (A-B)+C*((D*E)-F)/ G+(H*K)
POP X	M[X] ← Tos.

Ques. PC = 202 (Program Counter)
R_1 = 400 (Processor Register)
AC (Accumulator)
XR = 100 (Index Register)



Sol. (i) Direct = EA = M[A] = 500

(ii) Indirect = EA = M[A] = M[500] = 800.

(iii) Immediate = EA = 201

(iv) Register direct = EA = R_1

(v) Register indirect = EA = 400.

(vi) Relative = PC + Add. = 202 + 500 = 702.

(vii) Index = Index + Add. = 100 + 500 = 600

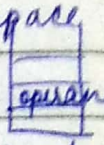
(viii) Autoincrement = 400 (Reg. Value)

(ix) Autodecrement = 399 (Reg. Value)

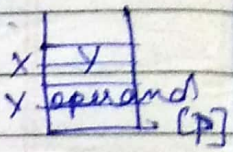
Addressing Mode :-

It specifies the different ways of determining the effective (actual) address of an operand.

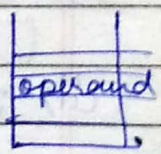
1. **Direct Addressing Mode :**
   → Add. field contain add. of operand → United add. space
   Effective address = address part = M[X]
   → Single memory reference to fetch data
   → no additional calculation to work out effective add.



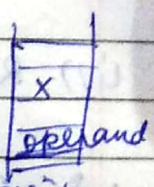
2. **Indirect Addressing Mode :**
   → memory cell pointed to add. field contains add. of operand.
   Effective address = ~~address part~~ = M[Y]
   → large add. space
   → Multiple memory accesses to find operand. → 2^n where n = word length



3. **Register Addressing Mode :**
   → operand is held in register named in add. field
   Effective address = R1
   → limited no. of registers → very small add. field.
   → fast execution → no memory access



4. **Register Indirect Addressing Mode :**
   → fewer memory access than indirect addressing.
   Effective address = X = M[R1]
   → large add. space (2^n)
   → operand is in memory cell pointed to contents of register.



5. **Immediate Addressing Mode :**
   → Operand is given in the instruction itself.
   → operand = value → fast → limited range → no memory reference to fetch data.

6. **Implied Addressing Mode :**
   Example - CMA (Complements the Content of Accumulator).

Date _____

operand = Accumulator.

Effective Address = Accumulator.

(7) Relative Addressing mode.

$$EA = PC + \text{Address part}$$

PC = Program Counter (next to inst")

(8) Index Addressing mode.

$$EA = \text{Index Register} + \text{Address part}$$

$$EA = X + M[R]$$

(9) Base Register Addressing mode.

$$EA = \text{Base Register} + \text{Address part}$$

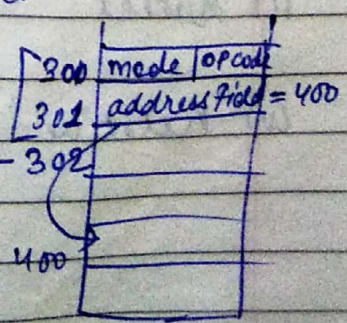
Ques.1 An Instruction is stored at Location 300 with its address field at Location 301. The address field has value 400. A processor register R_1 contains the value 200.

8.18

Determine effective address for.

1. Direct Addressing.
2. Immediate Add.
3. Relative Add.
4. Register Indirect Add.
5. Index addressing with R_1 as index register.
6. Indirect
7. Register direct.

Sol. R_1, #200



1. EA (Direct add.) = 400.
2. EA (Immediate) = 301.
6. EA (Indirect) = M[400]

3. Relative Add.

EA = PC + Add. part.

= 302 + 400

= 702

4. Register Indirect

EA = M[R,J] = 200.

5. Index Addressing.

EA = Index + Add. part.

= 200 + 400.

EA = 600

7. Register direct.

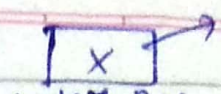
EA = R_1.

Que 2. An instruction is stored at memory address W. The address field is stored at address W+1 is designated by symbol Y. The operand used is stored at address Z. An index register contains the value X. Determine How Z is calculated using addressing modes.

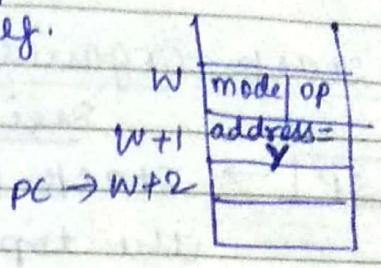
(i) Direct (ii) Indirect (iii) Register (iv) Register Indirect.

(v) Relative (vi) Index (vii) Immediate.

Date



Sol.



(i) EA = Address field
= Y

(ii) EA = M[Y].

(iii) EA = Index Register

(iv) EA = X

(v) Relative

EA = PC + Add.
= W+2 + Y
= W + Y + 2

(vi) Index.

EA = Index Reg + Add.
= X + Y

(vii) Immediate

EA = W+1.

Stack Organization :-

Register IN STACK PUSH Operation.

1. [SP] → Stack pointer contains the address of the top of stack. It reserves area in memory used in case of subroutine & program execution.
2. [DR] → Data Registers stores the data.
3. [FULL] → Represents, if stack is full.

   if [FULL = 1] → Totally fill.
   1 bit reg.

   if [FULL = 0] → Quite empty.
   1 bit reg

4. [EMPTY] → Represents, if stack is empty.

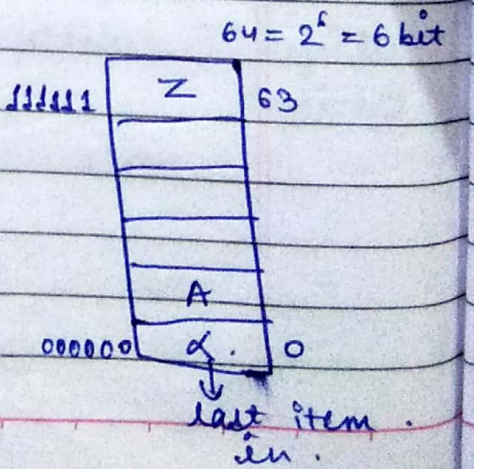
   if [EMPTY = 1] → Totally empty.
   1 bit reg.

   if [EMPTY = 0] → Quite fill.

→ PUSH Operations :-

Initial SP ← 0, empty ← 1, FULL ← 0.

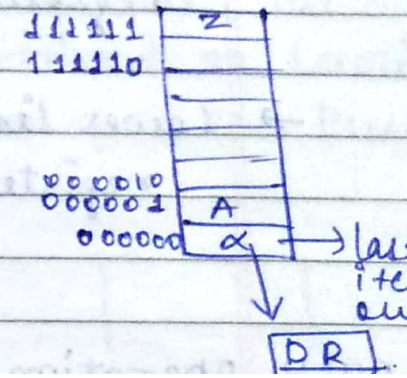
1. SP ← SP + 1
2. M[SP] ← DR
3. If (SP = 0) then FULL ← 1.
4. Empty ← 0.



Date

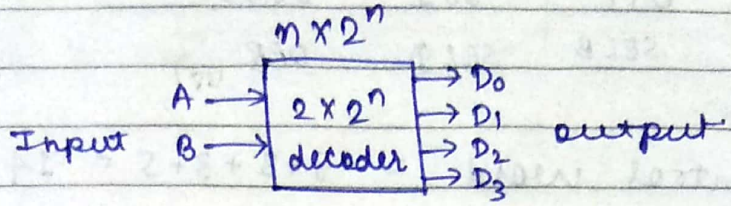
→ POP operation :-

1. DR ← M[SP]
2. SP ← SP - 1.
3. If (SP = 0) then empty ← 1.
4. Full ← 0.



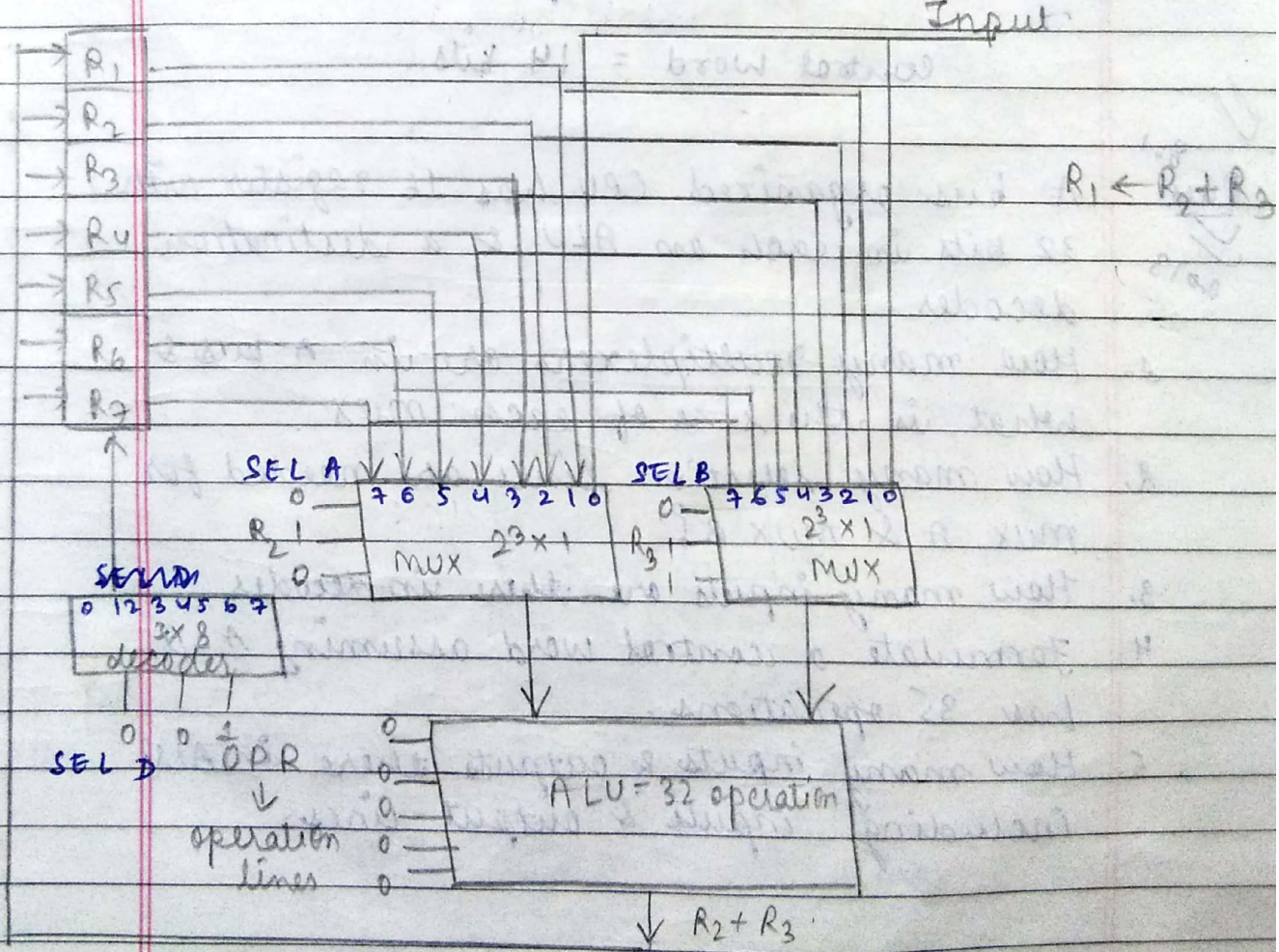
Date _____

Decoder :-



A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

General Register Organization :-



Registers are very useful component of CPU. An inst" work faster when its operand are stored in CPU register.

Date

→ For $R_1 \leftarrow R_2 + R_3$

3 bit	3 bit	3 bit	5 bit
010	011	001	00000
SEL A	SEL B	SEL D	OPR (+)

Control word = 3+3+3+5 = 14 bits

→ For $R_4 \leftarrow R_1 - R_2$

3 bit	3 bit	3 bit	5 bit
001	010	100	00001
SEL A	SEL B	SEL D	OPR

Control word = 14 bits

8.1

Ques. A bus organized CPU has 16 register with 32 bits in each an ALU & a destination decoder.

2013

1. How many multiplexers are in A bus & what is the size of each MUX.
2. How many selection lines are needed for MUX A & MUX B?
3. How many inputs are there in decoder.
4. Formulate a control word assuming ALU has 35 operations.
5. How many inputs & outputs there in ALU including inputs & output lines.

Date

Sol. (i) No. of MUX in A bus = No. of bits in each reg.
= 32 MUX

Size of each MUX = No. of reg. x 1
= 16 x 1 MUX.
= 2^4 x 1

(ii) No. of selection lines in MUX A = 4

No. of selection lines in MUX B = 4.

(iii) Size of decoder = $n \times 2^n$
= 4×2^4
= 4 x 16 decoder.

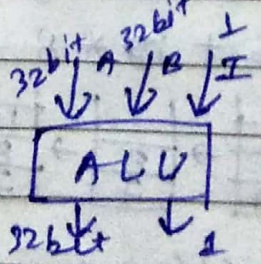
No. of Input = 4 inputs.

(iv) 35 operation = 2^6

Sel A	Sel B	Sel D	OPR
4 bit	4 bit	4 bit	6 bit

Control words = 18 bits.

(v)



Inputs = 32 + 32 + 1 = 65 lines.

Outputs = 32 + 1 = 33 lines

1) Information Transfer from 1 - Reg. to another is called Register transfer; $R_2 \leftarrow R_1$.

2) It shows the transfer of content of Reg. R_1 to Reg. R_2.

3) Content of R_1 does not change.

Register Transfer Language :-

Symbolic notation used to represent the transfer of information between register.

Representation of Registers :-

1. Data register → DR.
2. Instruction register → IR.
3. Program Counter → PC. [contain address of next instruction to be executed]
   PC++.
4. Memory Address Register → MAR.
5. General Purpose Register → $R_1, R_2, R_3, \ldots$ [carry data, Add, Instⁿ]

(i) Letter → $[R_1]$ ✓ ①
Numeral

(ii) Only Symbol → MAR, PC, IR,

(iii) Individual Bit Representation → 7 6 5 4 3 2 1 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ✓ ②

(iv) Bit system → 7 … 0 [R_1] ✓ ③

(v) 16 - Bit System → 15 … 8 7 … 0 [R_2(H) | R_1(L)] ✓ ④
Higher byte, Lower byte

(vi) $R_1(0-5)$ → Part of register.

② → It can be shown by if-else statement.

(vii) Register Transfer → P: $R_2 \rightarrow R_1$
If P = 1 then $R_1 \leftarrow R_2$

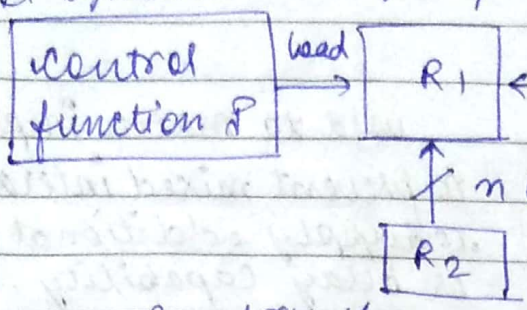
① → [Conditional Transfer of information] occur only under specific condition

4) Content of R_2 is replaced by content of R_1.
5) This transfer happen in 1 cycle.

Date

③ → It symbolise transfer operation takes place by hardware only if [P=1]

④ →



Block diagram

⑤ Parallel Reg. transfer (microoperation)

Set of Register transfer (more than one transfer, comma is used)

$R_3 \leftarrow R_2$, $R_1 \rightarrow R_2$

★ Register is a combination of flipflop.
1 Flipflop = 1 bit memory elements

Memory Transfer :-

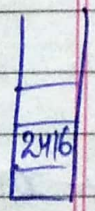
1. Memory Read → To transfer any information from memory to external environment (CPU)

if AR = 300

DR ← M[AR]

(Memory content at address stored in AR register) is transferred into data Register

~~from memory words~~ words



2. Memory Write → To transfer any information from external environment (CPU) to memory.

M[AR] ← DR.

Content of data Register transfer to memory words at address stored in AR register.

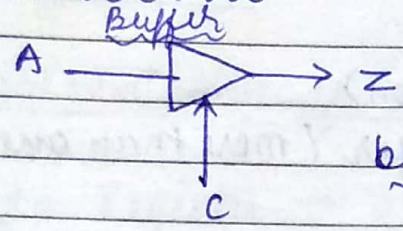
→ A 3-state gate can be any logic gate But in Bus system we use only 3-state Buffer gate.

Date

Bus Transfer :-

★ Representation of Tristate Gate -

used to match impedance, to prevent mixed interaction to supply additional drive or delay capability.

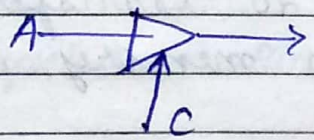


Tristate buffer gate used for (time delay)

If C = 1, Z = A i.e., A Gate behaves as like its actual behaviour or Gate is active.

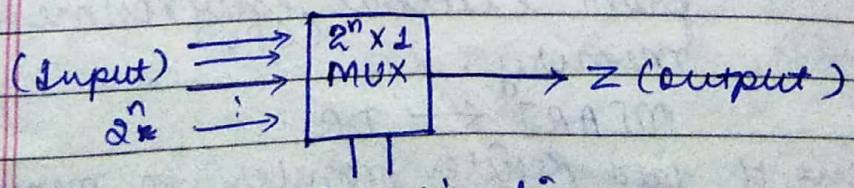
If C = 0, High impedance state, circuit act as open which is disable state.

★ Tri state Buffer Gate → A buffer gate having one input & one output as in conventional buffer gate along with one control input.



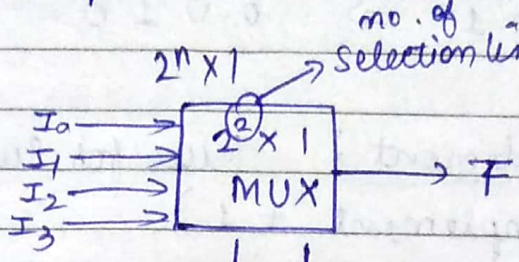
Multiplexer (MUX) :-

It is a digital circuit used to pass one output, 2^n (or less) input. It is called data selector from



High Impedance state behaves like open circuit i.e. the output is disconnected & does not have a logic significance.

for example :-



S_1	S_0	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

* No. of Mux Required = No. of bits in each registers
* Size of Mux = No. of Registers.

* Three State Bus Buffer :-

A Bus system that can be constructed with three state buffer gate.

* Three state gate :-

A 3-state gate is a digital circuit that shows 3-states. (i) State - 1 (ii) state - 0
(iii) High Impedance state.

Date

1's complement:

1 1 0 1 ⇒ 0 0 1 0

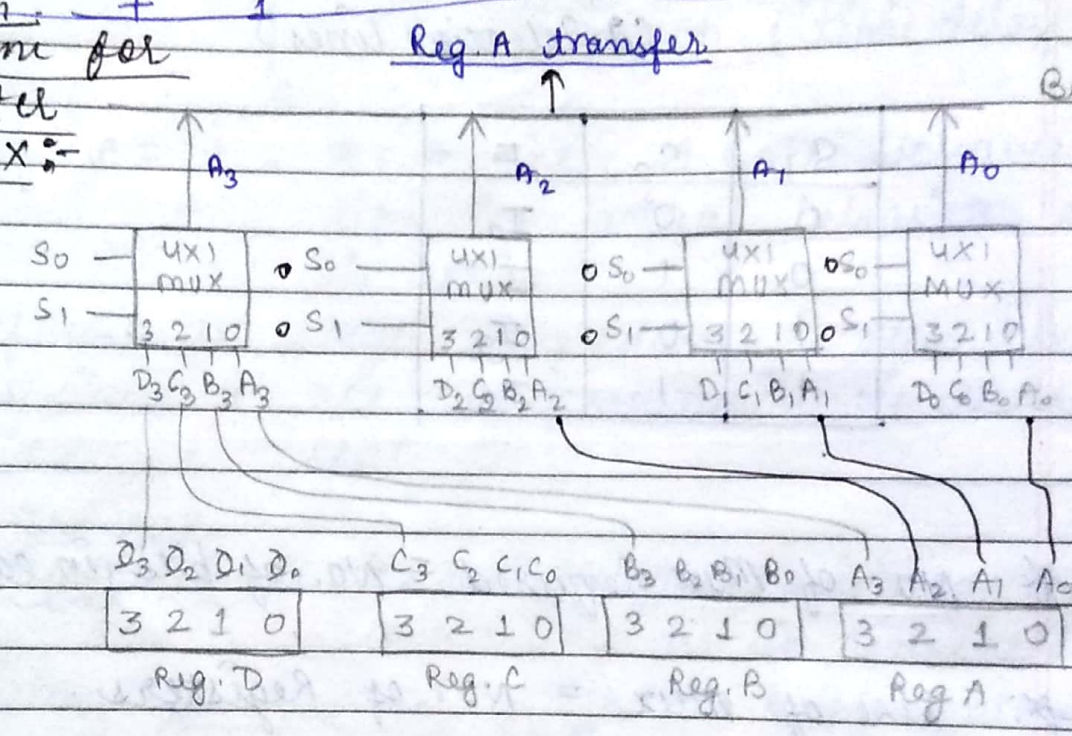
2's complement: (use for subtraction)

(1's complement + 1)

0 0 0 0 ⇒ 0 0 1 1

+ 1

Common Bus System for four-Register using MUX:-



	S_0	S_1	Reg. Select
I_0	0	0	A
I_1	0	1	B
I_2	1	0	C
I_3	1	1	D

A − B = A + 2's complement of B. = A + B' + 1

for eg. A + 0 = 0 0 0 0

− 0 = ?

1's complement of +0 = 1 1 1 1

2's complement of +0 + 1 = 1 1 1 1

\+ 1

neglected ← (1) 0 0 0 0

in 4-bit reg.

Ques 1 A digital computer has a common work system for 16 registers of 32 bits each. The bus is constructed with multiplexers.

(i) How many multiplexers are there in the bus?

(ii) What size of multiplexer are needed.

(iii) How many selection inputs are there in each multiplexers.

Sol.

(i) No. of multiplers = No. of bits in each register
= 32

(ii) Size of MUX = No. of registes used.
= 16 x 1 MUX.
= 2^4 x 1 MUX

(iii) No. of selection input = 4
= 4 lines

Date

Ques.2 The following transfer statement specify a memory. Explain the memory operation in each case.

(i) $R_2 \leftarrow M[AR]$.

(ii) $M[AR] \leftarrow R_3$

(iii) $R_5 \leftarrow M[R_5]$

(i) Memory Read operation, the memory content at an address specified by Register AR is transferred to Register R_2.

(ii) Memory Write Operation, the Register R_3 value is stored as memory content at an address specified by Register AR.

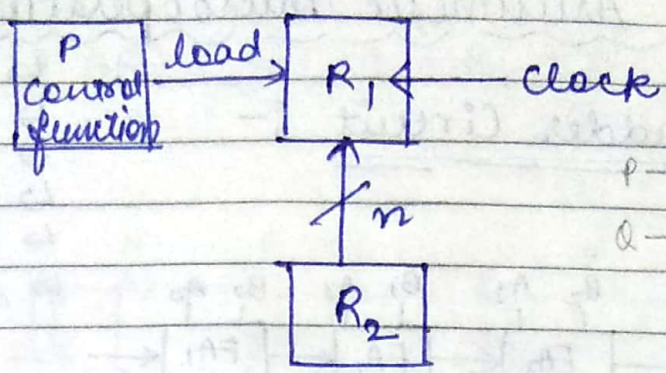
(iii) Memory Read operation, the memory content at an address specified by Register R_5 is transferred to Register R_5.

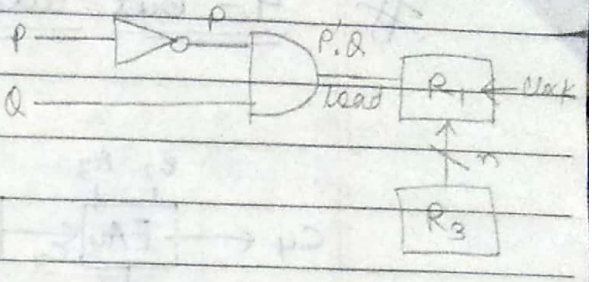
Ques.3 Represent the following conditional control statement by two register transfer statement with control function.

Sol. (i) If $P = 1$ then $(R_1 \leftarrow R_2)$, else if $Q = 1$ then $(R_1 \leftarrow R_3)$

Sol. $P : R_1 \leftarrow R_2$

$P' \cdot Q : R_1 \leftarrow R_3$.

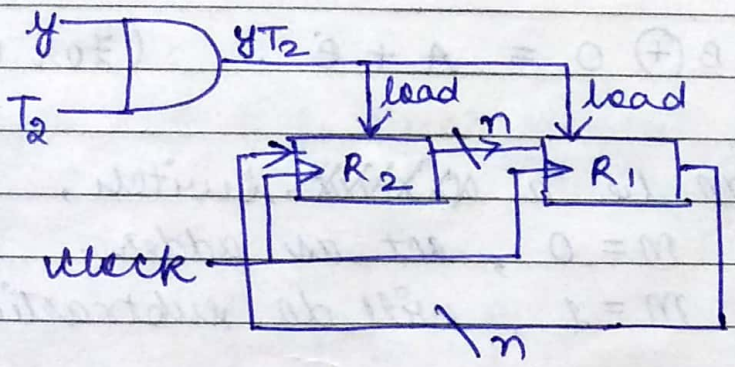




Ques.4. Show the block diagram of a hardware that implements the following register transfer statement.

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

Sol.



Common bus system using decoder & Buffergate.

S_1	S_0	output
0	0	A
0	1	B
1	0	C
1	1	D