Unit - 5 [18%]

Programming the basic Computers

1. Machine Language :-

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

→ Computer can execute programs only when they are in binary forms.

→ Programs written in any others language must be translated to binary format before execution by computer.

→ Programs written for a computers may be in one of the following categories:

1. Binary Code
   (Sequence of instructions & operands are in binary)
2. Octal or Hex Code :
   (Equivalent translation of binary code to octal & Hexadecimal code)
3. Symbolic Code
   (Users employs symbols - letters, Numeral, Special characters for operation part, address part & instruction code)

→ Each Symbolic instruction can be translated into binary coded data by using assemblers.

→ This type of Symbolic program is known as assembly language program.

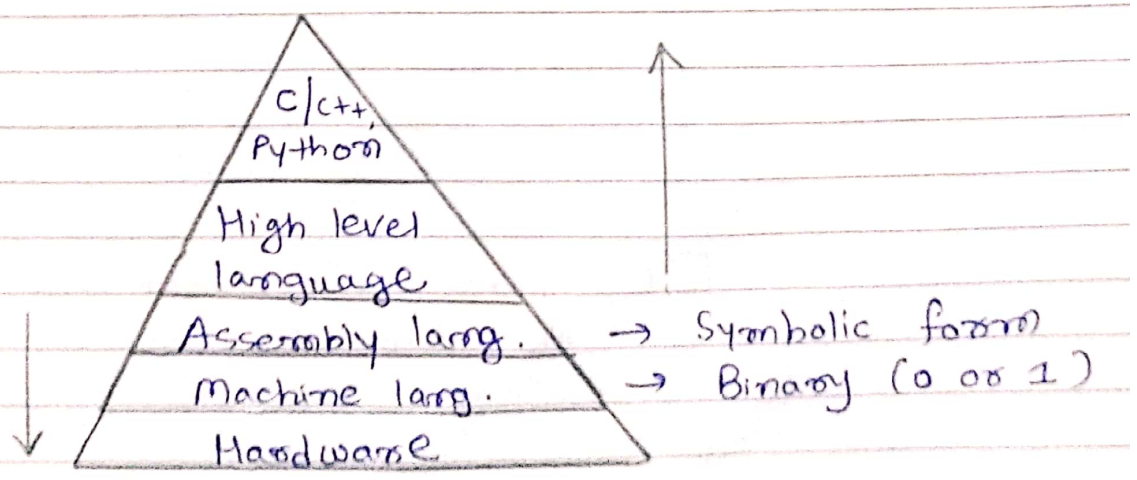
2 High level programming language :

→ In computer science, high level prog. language is a programming language which is closer to human language, and easier to read, write & maintain.

→ The well known examples of high level Programming languages are - Python, C/C++, Java, Javascript, PHP, Fortran etc

→ Programs written in high level language must be translated into machine language by Compilers.

→ A machine language program is a binary program



→ The 25 instructions of basic computers are repeated in the given table :

→ Each instruction is assigned 3 - letters Symbol.

→ In table, the symbol 'm' used to denote effective address and 'M' used to denote the memory word (operand).

*

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to $m + 1$
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC,
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

[Computer Instructions]
[Table - 0]

* Relation between binary & assembly languages :-

→ Consider the binary program written in table - 1.

→ 1st column is for location (in binary) and 2nd column lists the binary content of these memory locations).

→ However; a person looking at this program will have a difficult time understanding what is to be achieved when this prog. is executed.

→ But, the computer hardware recognized only this type of program.

Table-1 : Binary Program to Add Two Numbers

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

Table-2: Hexadecimal Program to add Two numbers

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

→ Writing 16 bits for each instruction is tedious because there are too many digits.

→ We can reduce the no. of digits per instruction if we write its octal or Hex equivalent as shown in Table-2.

→ The program in table-3 uses the symbolic names of instructions listed in Table-0, instead of their binary or Hex code.

Table-3: Program with symbolic Operation Codes

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

→ The address parts of MRI, as well as operands, remain in their Hex value.

→ Note that location 005 has negative operand because its sign bit = 1.

→ Symbolic programs are easier to handle, so it is preferable to write programs to with symbols.

→ We can go one step further & replace each hexadecimal address by symbolic address & each hex operand by decimal operand, as shown in table - 4.

Table-4 : Assembly Language Program to Add two numbers

```
      ORG 0       /Origin of program is location 0
      LDA A       /Load operand from location A
      ADD B       /Add operand from location B
      STA C       /Store sum in location C
      HLT         /Halt computer
A,    DEC 83      /Decimal operand
B,    DEC -23     /Decimal operand
C,    DEC 0       /Sum stored in location C
      END         /End of symbolic program
```

→ The symbol ORG is not machine instruction. Its purpose is to specify an origin.

→ The last line has the symbol END, indicating end of the program.

→ Decimal Operands are Specified by the Symbol DEC.

→ The numbers may be +ve or -ve, if -ve, they must be converted to binary in Signed 2's complement representation.

→ The Symbols : ORG, DEC and END called pseudo - instructions.

→ Note that all comments are preceded by Slash.

⇒ Pseudo - instructions

→ It is not a machine instruction but it is used to give information about Some phase of translation to assembler

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

[Definition of Pseudoinstructions]

* Rules of assembly language :-

→ Each line of assembly language program is arranged in 3 columns called fields

label field: May be empty or it may specify a symbolic address

Instruction field: It specifies a machine instruction or pseudo instruction.

Comment field: May be empty or it may include a comment

→ Symbolic address consists of one, two or three, but not more than three alpha-numeric characters

→ First character must be a letter, next two may be letters or numerals.

→ Instruction field may consists one of the following items:

1. MRI
2. Non-MRI (RRI or I/O)
3. Pseudo instruction with or without an operand.

* Translation of Symbolic program to binary :-

→ Consider an example of ~~addition~~ Subtraction of two decimal numbers 83 and -23.

```
      ORG 100      /Origin of program is location 100
      LDA SUB      /Load subtrahend to AC
      CMA          /Complement AC
      INC          /Increment AC
      ADD MIN      /Add minuend to AC
      STA DIF      /Store difference
      HLT          /Halt computer
MIN,  DEC 83       /Minuend
SUB,  DEC -23      /Subtrahend
DIF,  HEX 0        /Difference stored here
      END          /End of symbolic program
```

[ALP to Subtract two numbers]

→ The translation of symbolic program into binary can be done by assemblers.

→ The translated program is as shown in table.

Hexadecimal code			
Location	Content		Symbolic program
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC -23
108	0000	DIF,	HEX 0
			END

[Translated Program]

→ The translation process can be simplified if we scan the entire program twice.

→ When the 1st scan is completed, we associate with each label its location numbers & form a table called address symbol table.

Address Symbol	HEX Address
MIN	106
SUB	107
DIF	108

→ During the second scan, we refer to the address symbol table to find the address value of MRI.

* The Assembler :-

→ It is a program that accepts symbolic language program & produce its binary machine language code.

→ The i/p symbolic Program is called the Source program & resulting binary program is called object program.

→ In basic computers, each characters is represented by 8-bit code (ASCII code).

→ Each character is assigned two hex digits which can be easily converted to their equivalent 8-bit code.

→ The code for CR is produced when return key is pressed & start typing in a new line.

Hexadecimal Character Code

Character	Code	Character	Code	Character	Code	
A	41	Q	51	6	36	
B	42	R	52	7	37	
C	43	S	53	8	38	
D	44	T	54	9	39	
E	45	U	55	space	20	
F	46	V	56	(	28	
G	47	W	57	)	29	
H	48	X	58	*	2A	
I	49	Y	59	+	2B	
J	4A	Z	5A	,	2C	
K	4B	0	30	-	2D	
L	4C	1	31	.	2E	
M	4D	2	32	/	2F	
N	4E	3	33	=	3D	
O	4F	4	34	CR	0D	(carriage return)
P	50	5	35			

→ A line of code is stored in memory locations with two characters in each location.

→ For example, a line of code is:

PL3, LDA SUB I

PL3 → label symbol (Max 3 letters)

LDA SUB I → Instruction

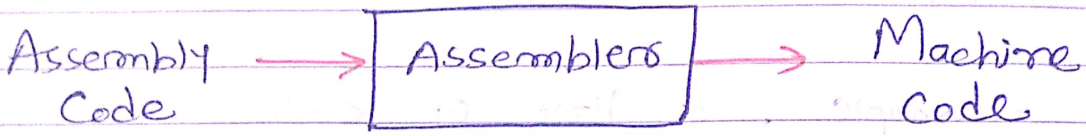
→ The above line of code is stored in 7 memory locations as shown in table:

Computers representation of line of Code: PL3, LDA SUB I

Memory word	Symbol	Hexadecimal code	Binary representation
1	P L	50 4C	0101 0000 0100 1100
2	3 ,	33 2C	0011 0011 0010 1100
3	L D	4C 44	0100 1100 0100 0100
4	A	41 20	0100 0001 0010 0000
5	S U	53 55	0101 0011 0101 0101
6	B	42 20	0100 0010 0010 0000
7	I CR	49 0D	0100 1001 0000 1101

→ As we can see, the lable PL3 occupies two words & is terminated by code for comma (2C).

* Two Pass Assembler :-



→ Assemblers divides the task into two passes :

Pass - 1 : ① Define symbols & remember them in symbol table.
② Keep track of location Counter (LC)
③ Process Pseudo - instruction.

Pass - 2 : ① Generate object code & data for literals & look for values of symbols.
② It does the actual assembly by translating operations into machine codes.

* Flowchart for 1st Pass of Assembler :

→ During the 1st pass, it generates a table that correlates all user defined address symbols with their binary value.

→ To keep track of the location of instructions, the assembler uses a memory word called **location Counter (LC)**

→ The assembler sets LC to 0 initially.

↓

→ A line of symbolic code is analyzed if it has label or not.

↓

If Yes

Assembler generates address symbol table with value of LC

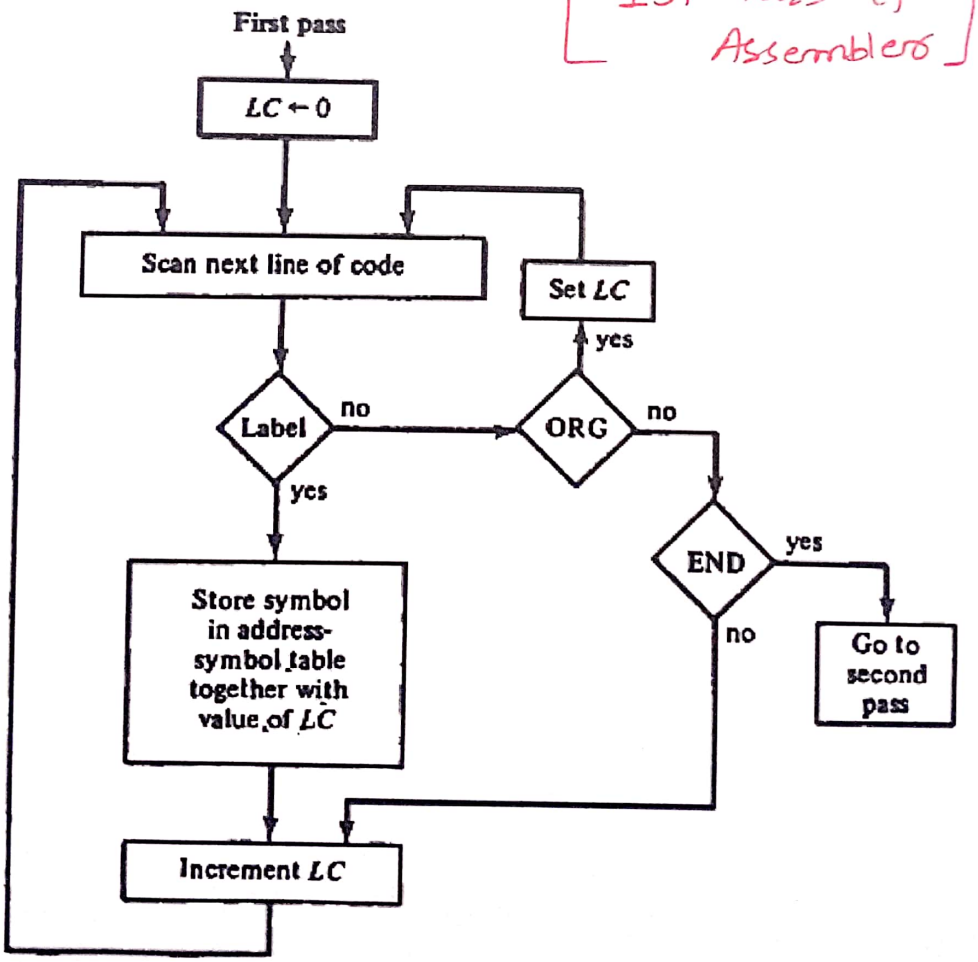
If No

Check for Pseudo instructions (ORG and END)

↓

At last, LC is incremented by 1 and scan next line of code.

[1'st Pass of Assembler]



→ Assembler generates address symbol table given in table - 1 in first pass.

Memory word	Symbol or (LC)*	Hexadecimal code	Binary representation
1	M I	4D 49	0100 1101 0100 1001
2	N ,	4E 2C	0100 1110 0010 1100
3	(LC)	01 06	0000 0001 0000 0110
4	S U	53 55	0101 0011 0101 0101
5	B ,	42 2C	0100 0010 0010 1100
6	(LC)	01 07	0000 0001 0000 0111
7	D I	44 49	0100 0100 0100 1001
8	F ,	46 2C	0100 0110 0010 1100
9	(LC)	01 08	0000 0001 0000 1000

* (LC) designates content of location counter.

[Table-1 : Address Symbol Table]

* Second pass of Assembler :-

→ Machine instructions are translated during the 2nd pass of assembler by means of look-up procedure.

→ Assembler uses four tables :

1. Pseudo-instruction table
2. MRI table
3. Non-MRI table
4. Address Symbol table

→ The entries of pseudo-instruction table are 4 symbols ORG, END, DEC and HEX.

→ The MRI table contains 7 symbols of MRI & 3-bit opcode.

→ The Non-MRI contains 18 instructions & their 16-bit binary code.
(12 RRI + 6 I/O = 18)

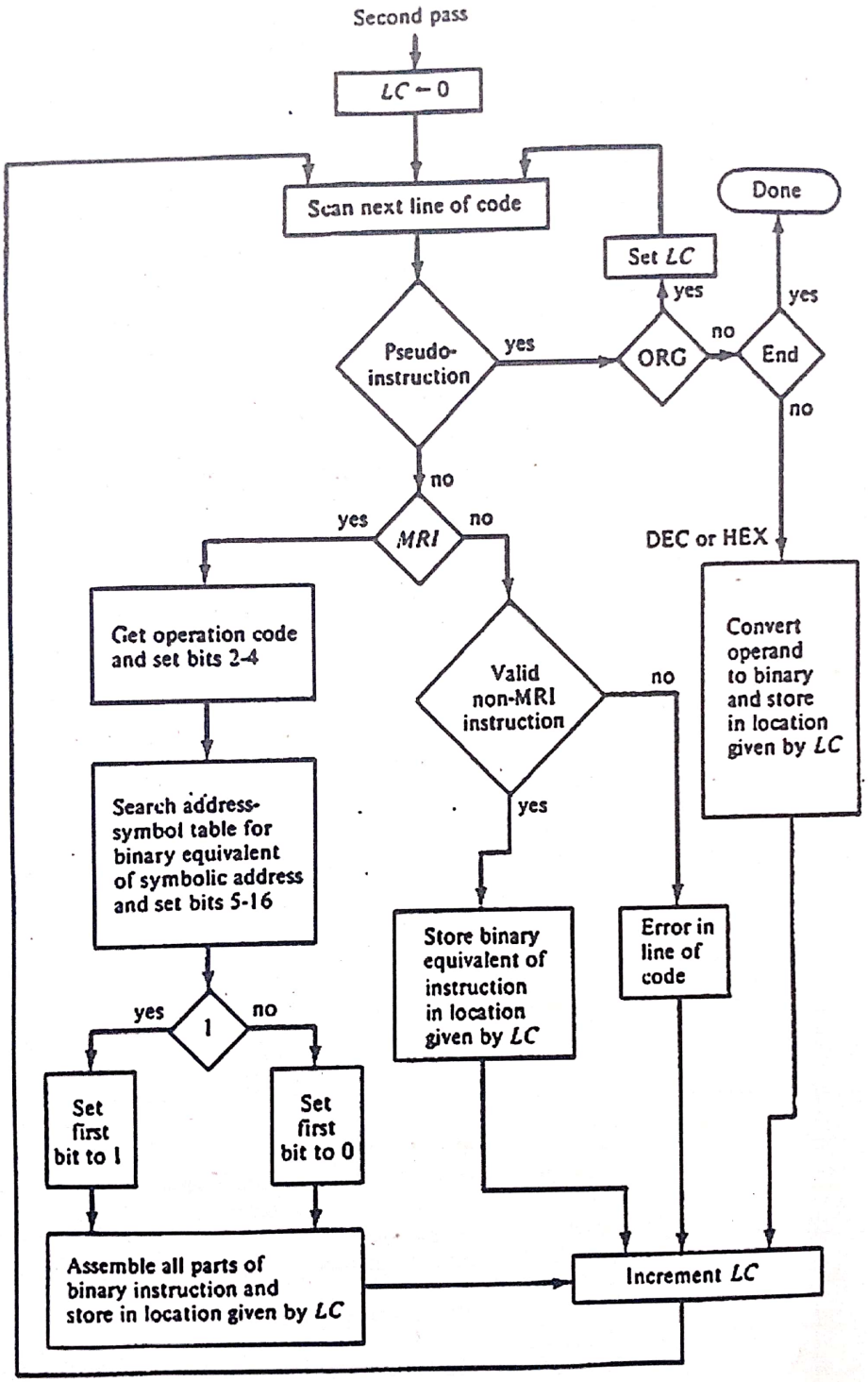
Error Diagnostics :

→ One important task of assembler is to check for possible errors in symbolic program.

→ If some error, may be an invalid machine code which is absent in MRI & Non-MRI table cannot be translated by assemblers.

→ Because assemblers does not know its binary equivalent value.

→ In such case, the assemblers prints an error message to inform the programmers that his symbolic code has an errors.



[Second Pass of Assemblers]

Programs:

1. Write an assembly language program (ALP) to add two double precision numbers.

$A_H\ A_L \rightarrow$ 32 bits
$+\ B_H\ B_L \rightarrow$ 32 bits
$C_H\ C_L$

B_L, A_L = 16 - low orders bits
B_H, A_H = 16 - high orders bits

ORG 100

```
100     LDA  AL
101     ADD  BL
102     STA  CL
103     CLA
104     CIL
105     ADD  AH
106     ADD  BH
107     STA  CH
108     HLT
109  AL, HEX  1800
110  AH, HEX  0032
111  BL, HEX  0020
112  BH, HEX  0001
113  CL, HEX  0
114  CH, HEX  0
115     END
```

A_H = 0032
A_L = 1800
B_H = 0001
B_L = 0020

```
  0032  1800
+ 0001  0020
---------------
  0033  1820
```

$C_H C_L$ = 0033 1820

[2] Write ALP for subtraction of two double precision number:

$A_H\ A_L$
$-\ B_H\ B_L$
$C_H\ C_L$

} A + (2's comp of B) = A − B

FF31 8FC3
− F010 8E82
0F21 0141

```
                ORG 100
100             LDA BL
101             CMA
102             INC
103             ADD AL
104             STA CL
105   Save      CLA
106   Carry     CIL
107             STA TMP
108             LDA BH
109             CMA
110             INC
111             ADD AH
112   Add Carry → ADD TMP
113             STA CH
114             HLT
115     TMP,    HEX 0
116     CH,     HEX 0
117     CL,     HEX 0
118     AL,     HEX 8FC3
119     AH,     HEX FF31
120     BH,     HEX F010
121     BL,     HEX 8E82
                END
```

3. Write ALP for logical XOR operation between two logic operands 8 and 5 in decimal.

$$Z = X \oplus Y = \overline{\bar{X}Y + X\bar{Y}}$$

$$= \overline{\bar{X}Y + X\bar{Y}}$$

$$= \overline{\bar{X}Y} \cdot \overline{X\bar{Y}}$$

$8 \rightarrow 1000$
$5 \rightarrow 0101$ $\oplus$
1101

```
        ORG 100
100     LDA Y
101     CMA
102     AND X
103     CMA
104     STA TMP
105     LDA X
106     CMA
107     AND Y
108     CMA
109     AND TMP
110     CMA
111     STA Z
112     HLT
113   X, DEC 8
114   Y, DEC 5
115   Z, HEX 0
116 TMP, HEX 0
117     END
```

Note: Similarly, we can write ALP for all other logic operations (OR, XNOR, NAND, NOR)

[4] Write ALP to multiply two 8-bit numbers

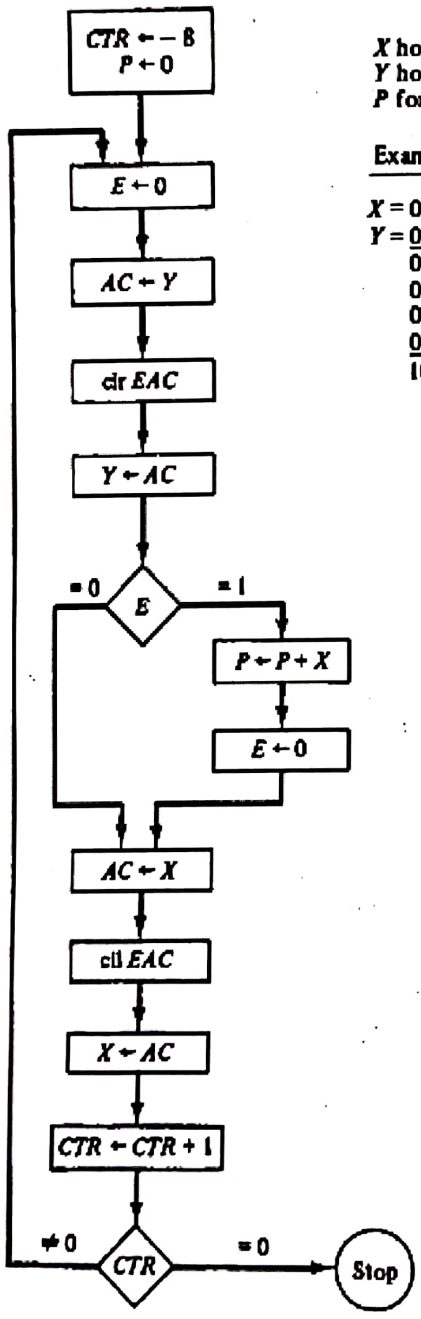
Let's take X = Multiplicand
Y = Multipliers
P = Product of X & Y

		P
X =	0000 1111	
Y =	0000 1011	0000 0000
	0000 1111	0000 1111
	0001 1110	0010 1101
	0000 0000	0010 1101
	0111 1000	1010 0101
	1010 0101	

Program:

```
                ORG 100
100  LOP,  CLE          / Clears E
101        LDA Y         / Load Multipliers (Y)
102        CIR           / Transfers multipliers bit to E
103        STA Y         / Store shifted multipliers
104        SZE           / Check if E bit = 0 & skip
105        BUN ONE       / If bit = 1, go to ONE
106        BUN ZRO       / If bit = 0, go to ZRO
107  ONE,  LDA X         / Load Multiplicand
108        ADD P         / Add to partial product
109        STA P         / Store partial product
110        CLE           / Clears E
111  ZRO,  LDA X         / Load Multiplicand
112        CIL           / Shift left
113        STA X         / Store shifted Multiplicand
```

```
114        ISZ   CTR     / Increment Counter
115        BUN   LOP     / Counter not zero, repeat
116        HLT           / Counter = 0 then halt
117  CTR,  DEC   -8      / Counter value
118  X,    HEX   000F
119  Y,    HEX   000B
120  P,    HEX   0
121        END
```



X holds the multiplicand
Y holds the multiplier
P forms the product

Example with four significant digits

		P
X =	0000 1111	
Y =	0000 1011	0000 0000
	0000 1111	0000 1111
	0001 1110	0010 1101
	0000 0000	0010 1101
	0111 1000	10100101
	10100101	

[Flowchart for Multiplication Program]

5. Write a program to multiply two positive numbers by repeated addition method. For example, to multiply 5 x 4, the program evaluates the product by addition of 5 four times, 5 + 5 + 5 + 5.

```
        ORG 100
100     LDA A      / Load multiplier
101     SZA        / Is it zero ?
102     BUN NZR
103     HLT        / A = 0, product = 0 in AC
104 NZR, CMA
105     INC
106     STA CTR    / Store -A in counter
107     CLA        / Start with AC = 0
108 LOP, ADD B
109     ISZ CTR
110     BUN LOP    / Repeat loop A times
111     HLT
112   A, DEC 4
113   B, DEC 5
114 CTR, HEX 0
115     END
```

* Subroutines :

→ Sometimes, the same piece of program must be written over again in many parts of program.

→ Instead of repeating the code every time it is needed, there is an advantage if common codes are written only once.

→ A set of common instructions that can be used in program many times is called a Subroutine.

* Example of Subroutine :-

```
                ORG 100
100             LDA X       / Load X
101             BSA SH4     / Branch to Subroutine
102             STA X       / Store Shifted no.
103             LDA Y       / Load Y
104             BSA SH4     / Branch to Subroutine
105             STA Y       / Store Shifted no.
106             HLT
107     X,      HEX 1234
108     Y,      HEX 4321
109     SH4,    HEX 0       / Store return add. here
10A             CIL
10B             CIL
10C             CIL
10D             CIL         / Circulate left 4th time
10E             BUN SH4 I   / Return to Prog Main
10F             END
```

Convert:
1234 ---> 2340 and
4321 ---> 3210
To perform the above operation we do not need 4 times CIL operations for 1234 and another 4 times CIL operation for 4321. Instead of this we make a subroutine program and call it whenever it requires in main program.

The above subroutine program is used to mask the last 4 LSB bits of any given number. For example: User wants to convert the number from 1234 to 2340, for that we need to circulate the digits four times left that is CIL instruction : Circulate left E and AC

7 Write an ALP to add 100 numbers from location 150 to 249.

```
        ORG  100
100     CLA          / Clear AC
101 LOP, ADD PTR I
102     ISZ  PTR
103     ISZ  CTR
104     BUN  LOP
105     STA  SUM
106     HLT
107 PTR , HEX  150
108 CTR , DEC  -100
109 SUM , HEX  0
        END
```

* I/O Programming:

→ The symbols are strings of characters & each character is an 8-bit binary code so that it can be stored in computer memory.

INP: A binary coded characters enters the computers when INP is executed.

OUT: A binary coded characters is transferred to o/p device when OUT is executed.

Program to Input & Output One Characters

```
(a) Input a character:
CIF,    SKI         /Check input flag
        BUN CIF     /Flag=0, branch to check again
        INP         /Flag=1, input character
        OUT         /Print character
        STA CHR     /Store character
        HLT
CHR,    —           /Store character here
(b) Output one character:
        LDA CHR     /Load character into AC
COF,    SKO         /Check output flag
        BUN COF     /Flag=0, branch to check again
        OUT         /Flag=1, output character
        HLT
CHR,    HEX 0057    /Character is "W"
```

Ex The following program is Stored in memory unit of computer. Show the contents of AC, PC and IR (in Hex), at the end, after each instruction is executed. (All the no. are in Hex).

Location	Instruction
010	CLA
011	ADD 016
012	BUN 014
013	HLT
014	AND 017
015	BUN 013
016	C1A5
017	93C6

Content of AC, PC & IR:

Location	Instruction	AC	PC	IR
010	CLA	0000	011	7800
011	ADD 016	C1A5	012	1016
012	BUN 014	C1A5	014	4014
013	HLT	8184	--	7001
014	AND 017	8184	015	0017
015	BUN 013	8184	013	4013
016	C1A5			
017	93C6			

Ex The following program is a list of instruction in its Hex Code. Execution starts from location 100. What are the content of AC & memory word at address 103. When the computer halts?

Location	Instruction
100	5103
101	7200
102	7001
103	0000
104	7800
105	7020
106	C103

→ Answer:

Location	Instruction (Hex Code)	Instruction	AC
100	5103	BSA 103	
101	7200	CMA	→ FFFE
102	7001	HLT	
103	0000	103	
104	7800	CLA	→ 0000
105	7020	INC	→ 0001
106	C103	BUN 103 I	

Ex List the ALP generated by a compiler for the following IF statement:

IF (A-B) 10, 20, 30

The program branches to statement 10 if A-B < 0, to statement 20 if A-B = 0 and to statement 30 if A-B > 0.

```
    ORG 100
    LDA B
    CMA
    INC
    ADD A
    SPA      / Skip if AC = +ve
    BUN N10  / (A-B) < 0, go to N10.
    SZA      / Skip if AC = 0
    BUN N30  / (A-B) > 0, go to N30
    BUN N20  / (A-B) = 0, go to N20
```

* General Register Organization :

→ When a large no. of registers are included in CPU, it is most efficient to connect them through common bus system.

→ It is necessary to provide a common unit that can perform all the arithmatic, logic & shift micro operations.

→ A bus organization for 7 - CPU registers is shown in fig - A

→ The o/p of each register is connected to two mux to form two buses A and B.

1. MUX A Selector (SELA):
To place the content of R2 into bus A

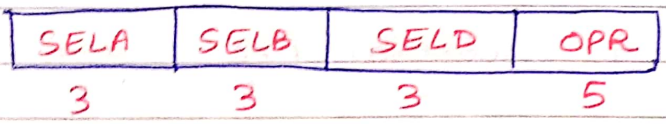
2. MUX B Selector (SELB):
To place the content of R3 into bus B

3. ALU operation Selector (OPR):
To provide addition A + B

4. Decoder destination Selector (SELD):
To transfers the content of output bus into R1.

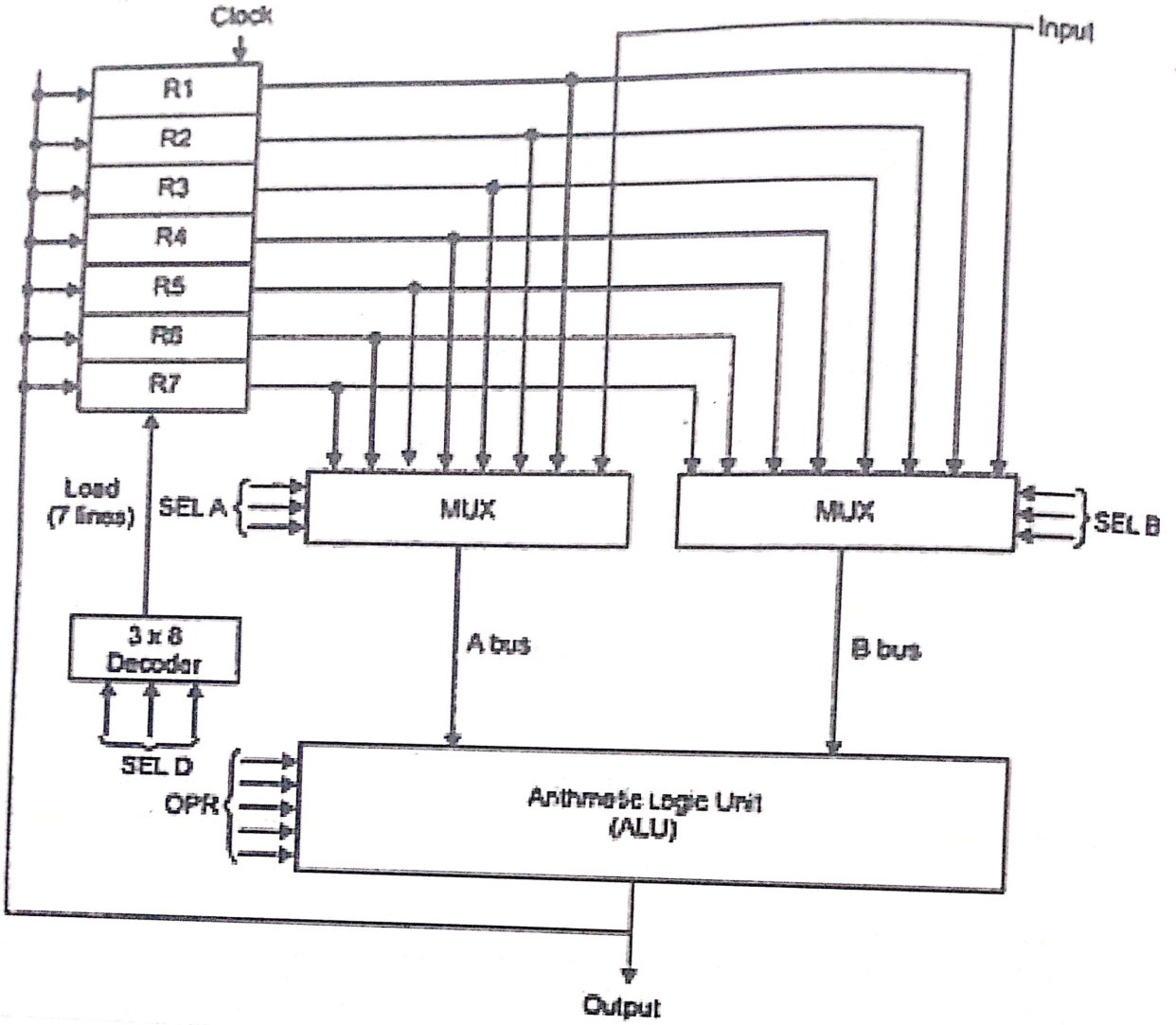
* Control word:

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



[control word]

SELA : It selects source registers for A input of ALU
SELB : It selects registers for B input of ALU
SELD : It selects destination registers using decoders
OPR : It selects one of operations in ALU



[fig-A : General Reg. Organization]

→ As shown in table-1, When SELA and SELB is 000, corresponding MUX selects the external input data.

→ When SELD = 000, no destination register is selected but the content of o/p bus are available in the external o/p.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

[Table-1 : Encoding of Reg selection Fields]

→ As shown in table-2, The OPR field has 5-bits and each operation is designated with symbolic name.

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

[Table-2 : Encoding of ALU operations]

→ Examples of Micro-operations:

→ For example subtract micro-operation given by -

$$R1 \leftarrow R2 - R3$$

→ It specifies R2 for A-input of ALU R3 for B input of ALU, R1 for destination registers., and ALU performs an operation to subtract : A - B

→ So, binary control word for above operation is given by -

Field :	SELA	SELB	SELD	OPR
Symbol :	R2	R3	R1	SUB
Control word :	010	011	001	00101

Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
R1 ← R2 − R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	—	R6	INCA	110 000 110 00001
R7 ← R1	R1	—	R7	TSFA	001 000 111 00000
Output ← R2	R2	—	None	TSFA	010 000 000 00000
Output ← Input	Input	—	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	—	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

[Examples of Micro-operations for CPU]

* Stack Organization :-

→ A useful feature that is included in CPU of most computers is Stack or Last-in first out (LIFO) list.

→ Stack : It is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

→ The operation of stack can be compared to stack of trays. The last tray placed on top of stack is first to be taken off.

→ Stack Pointer (SP) :-

→ The register that holds the address for the stack is called stack pointer.

→ Its value always points at the top item in stack.

→ Two operations of stack are :
① Insertion (PUSH or PUSH-DOWN)
② Deletion (POP or POP-UP)

PUSH : Pushing a new item on top

POP : Removing one item so that the stack pops up.

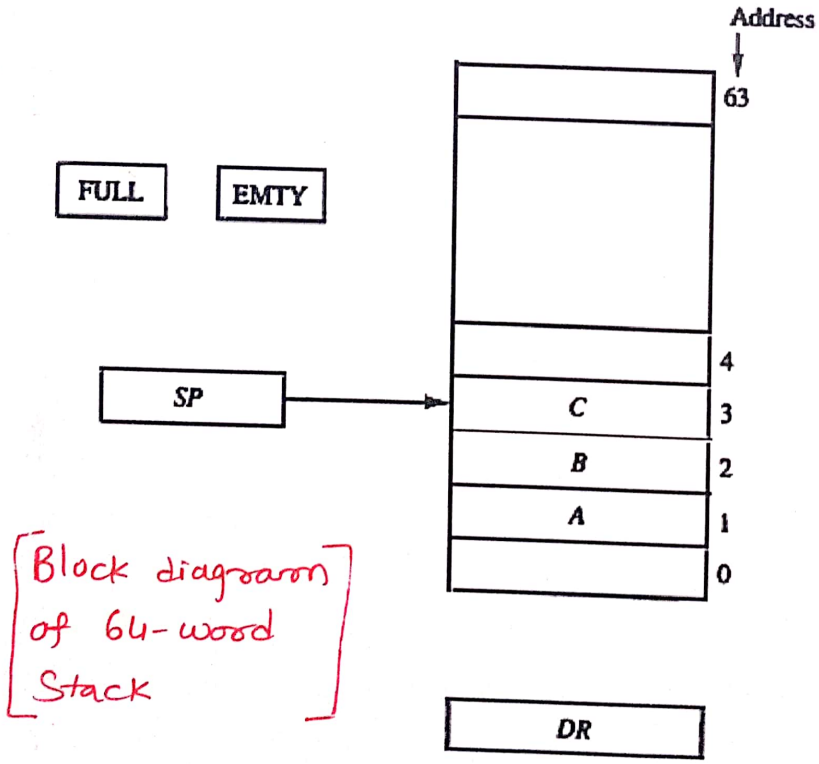
→ Push & Pop operations are simulated by incrementing or decrementing the stack pointer registers.

* Registers Stack :

→ The figure shows the organization of 64-word register stack.

→ SP register contains binary numbers whose value is equal to address of the word that is currently on top of the stack.

→



[Block diagram of 64-word Stack]

→ Three items are placed in the stack A, B and C in the orders.

→ Item C is on top of stack.

→ So, that content of SP is now 3.

→ To remove the top item, the stack is popped by reading the memory word at address 3 & decrementing the content of SP.

→ In 64-word stack, the SP contains 6 bits because $2^6 = 64$.

→ Since SP has only 6 bits, it cannot exceed a number greater than 63. (111111 in binary).

→ When 63 is incremented by 1, the result is 0 because $111111 + 1 = 1000000$ in binary.

→ When 0 is decremented by 1 means 000000 is decremented by 1, the result is 111111.

FULL : It is 1-bit Register.
It sets to 1 when stack is full.

EMTY : It is 1-bit Register.
It sets to 1 when stack is empty of items.

DR : It is the data register that holds binary data to be written into or read out of the stack.

* Sequence of Micro-operations for PUSH Operation:

→ Initially SP is cleared to 0.
EMTY is set to 1.
FULL is cleared to 0.

→ If the stack is not full (FULL = 0) new item is inserted with Push operation.

PUSH

SP ← SP + 1	→ Increment SP
M[SP] ← DR	→ Write items on top of stack.

If (SP = 0) then (FULL ← 1) → Check if Stack is full
EMTY ← 0 → Mark the Stack not empty.

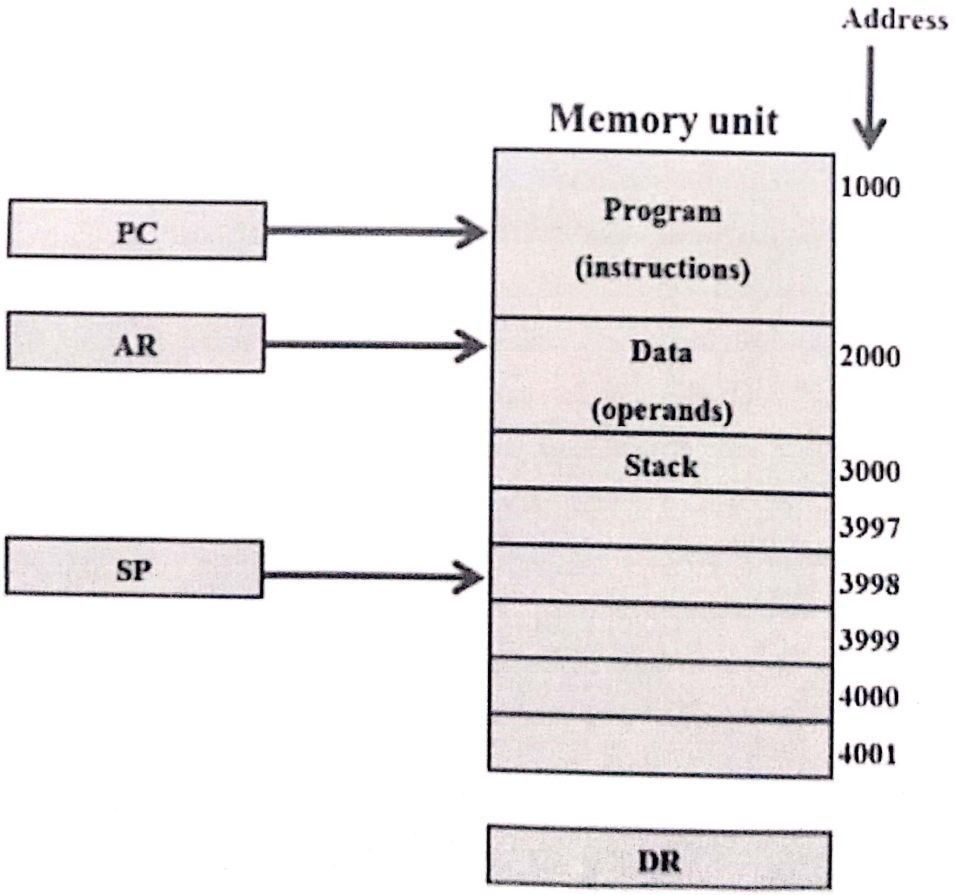
* Sequence of Micro-operations for POP Operation :-

→ When a new item is deleted from Stack when the stack is not empty. (EMTY = 0).

	DR ← M[SP]	→ Read item
	SP ← SP-1	→ Decrement SP
POP	If (SP = 0) then (EMTY ← 1)	Check if stack is empty
	FULL ← 0	→ Mark the stack is not full.

* Memory Stack :

→ A stack can be implemented in random access memory attached to CPU.



[Computer Memory with Program, Data & Stack]

→ The figure shows portion of Computers memory into 3 segments : Program, data and Stack

PC : Prog Counter Points at address of the next instruction in Program.

AR : It points at an array of data.

SP : It points at the top of Stack.

→ These 3 registers are connected to a Common address bus.

→ PC is used during fetch phase to read an instruction.

→ AR is used during execute phase to read an operand.

→ SP is used to push or pop items into or from the Stack.

→ As shown in fig : Initial value of SP = 4001 and Stack grows with decreasing address.

→ So, the first item stored in Stack is at address 4000, the second is at 3999 and last at 3000.

→ A new item is inserted with PUSH operation as follows:

SP ← SP − 1
M[SP] ← DR

→ A new item is deleted with a POP operation as follows:

DR ← M[SP]
SP ← SP + 1

* Reverse Polish Notation : (RPN)

→ A Stack organization is very effective for evaluating arithmatic expressions.

→ Three different ways of writing expressions:

Infix : X + Y

Postfix or RPN : X Y +

Prefix : + X Y

→ Infix : Operations are written in-between their operand

→ Postfix : Operations are written after their operands.

→ Prefix : Operations are written before their operands

→ For example, we have infix notation like: A * (B + C) / D

→ It can be represented in reverse polish or post-fix notation like:

A B C + * D /

→ The order of evaluation of operators is always left to right.

→ Because the '+' is to the left of '*' in above example, addition must be performed before multiplication.

→ RPN leads to faster calculations for a couple of reasons. – One is that there is less information to store.

→ For example we have an expression:
((5 - 3) * 2)

→ In above expression, instead of using 9 characters, computers with RPN only need to store 5 characters that is: 53 - 2 *

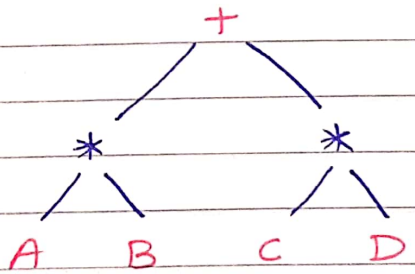
⇒ Considers an expression: A * B + C * D

In RPN: A B * C D * +

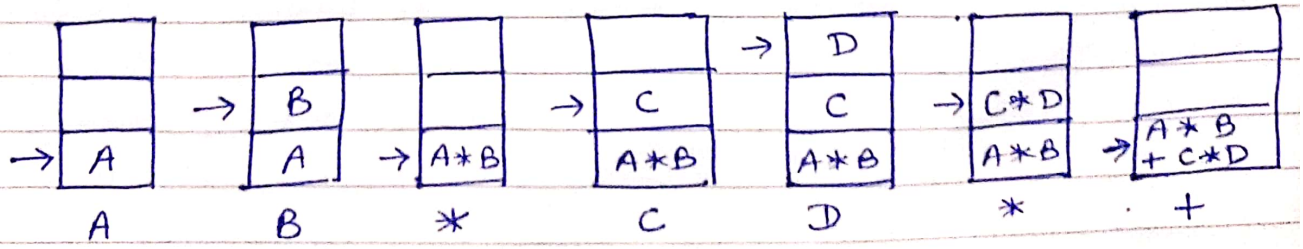
→ The evaluation of expression AB * CD* + is as follows :

1. PUSH A into the Stack.
2. PUSH B into the Stack. (It is on the position above A, i.e top of the Stack).
3. Apply the multiplication operation and result A * B Stored back to Stack.
4. PUSH C into Stack
5. PUSH D into Stack.
6. Apply the multiplication & C * D is available at top of Stack.
7. Apply addition and Store the result back in the Stack : A * B + C * D.

→ The above operation can be represented in another way :



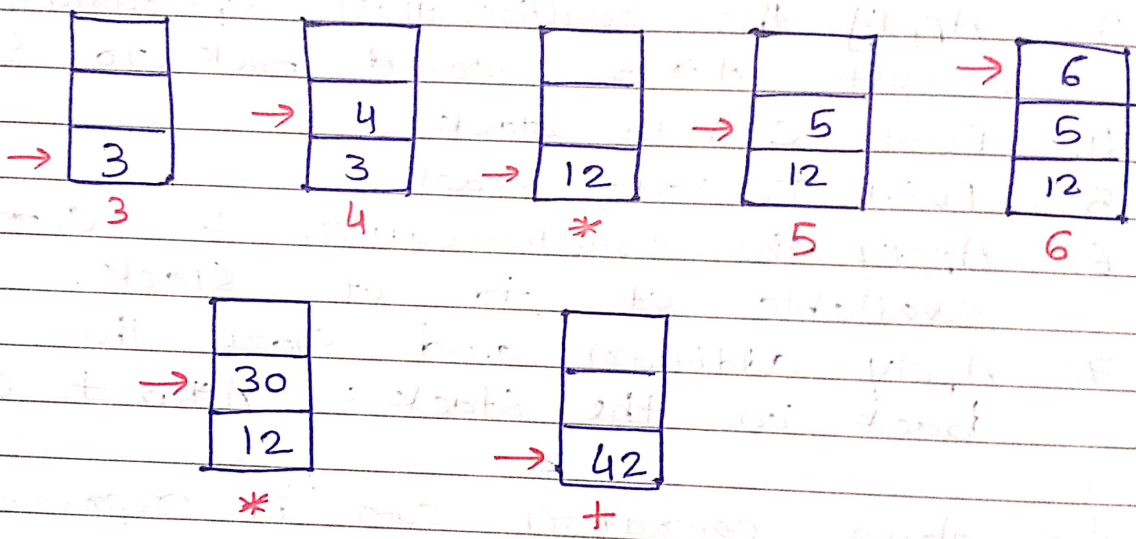
⇒ Stack Operation to evaluate : AB * CD* +



* Evaluation of Arithmatic Expression using RPN :-

⇒ Consider (3 * 4) + (5 * 6)

In RPN: 34 * 56 * +



Examples:-

8.1 Specify the control word to implement the following micro-operations:

(a) $R1 \leftarrow R2 + R3$

Field :	SELA	SELB	SELD	OPR
Symbol :	R2	R3	R1	ADD
Control word :	010	011	001	00010

(b) $R4 \leftarrow \overline{R4}$

Field :	SELA	SELB	SELD	OPR
Symbol :	R4	–	R4	COMA
Control word :	100	XXX	100	01110

(c) $R6 \leftarrow shl\ R1$

Field :	SELA	SELB	SELD	OPR
Symbol :	R1	–	R6	SHLA
Control word :	001	XXX	110	11000

(d) $R7 \leftarrow input$

Field :	SELA	SELB	SELD	OPR
Symbol :	Input	–	R7	TSFA
Control word :	000	XXX	111	00000

Q.2 Determine the micro-operation that will be executed in processor when 14-bit control words are applied :

(a) 00101 00111 00101

001 010 011 00101

R1 R2 R3 SUB

Micro-operation: R3 ← R1 − R2

(b) 000 000 000 00000

000 000 000 00000

Input Input None TSFA

Micro-operation : Output ← Input

(c) 111 100 011 10000

111 100 011 10000

R7 R4 R3 SHRA

Micro-operation : R3 ← Shr R7

Q.3 Let SP = 000000 in Stack. How many items are there in Stack if :

(a) FULL = 1 , EMTY = 0

(b) FULL = 0 , EMTY = 1

Ans-(a): Due to FULL = 1, Stack is full of items. Total items = 64

Ans-(b): Due to EMPTY = 1, Stack is empty

Q.3 Convert the following arithmetic expressions from infix to reverse polish notation.

1. A * B + C * D + E * F

= AB * + CD * + EF *

= AB * CD * EF * + +

2. A * B + A * (B * D + C * E)

= AB * + A * (BD * CE * +)

= AB * + ABD * CE * + *

= AB * ABD * CE * + * +

3. A + B * [C * D + E * (F + G)]

= AB + * [CD * + EFG + *]

= AB + * CD * EFG + * +

= ABCD * EFG + * + * +

4. $\frac{A * [B + C * (D+E)]}{F * (G+H)}$

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

$= \frac{A * [B C D E + * +]}{F G H + *}$

$= \frac{A B C D E + * + *}{F G H + *}$

$= A B C D E + * + * F G H + * /$

Q.4 Convert the following arithmatic expression from RPN to infix notation.

1. $A B C D E + * - /$

$= A B C (D+E) * - /$

$= A B C * (D+E) - /$

$= A (B - C * (D+E)) /$

$= A / [B - C * (D+E)]$

$= \frac{A}{B - (D+E) * C}$

2. A B C D E * / − +

= A B C (D * E) / − +

= A B C / (D * E) − +

= A [B − C / (D * E)] +

= $A + B - \frac{C}{D * E}$

3. A B C * / D − E F / +

= A / B * C − D + E / F

= $\frac{A}{B * C} - D + \frac{E}{F}$

4. A B C D E F G + * + * + *

= A B C D E (F + G) * + * + *

= A B C (D + E) * (F + G) * + *

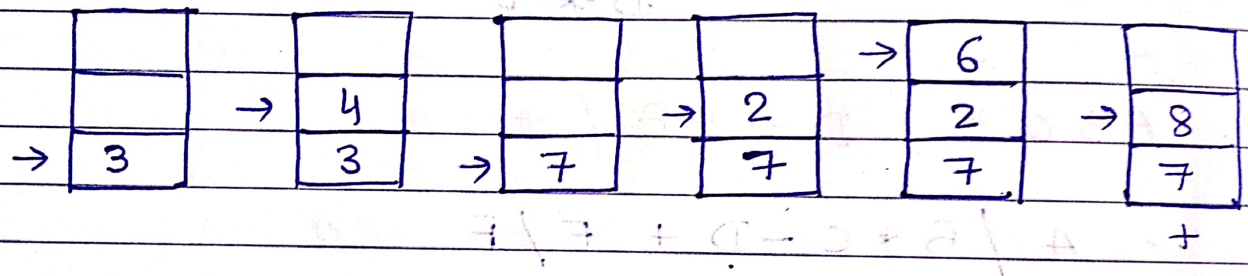
= A (B + C) * (D + E) * (F + G) *

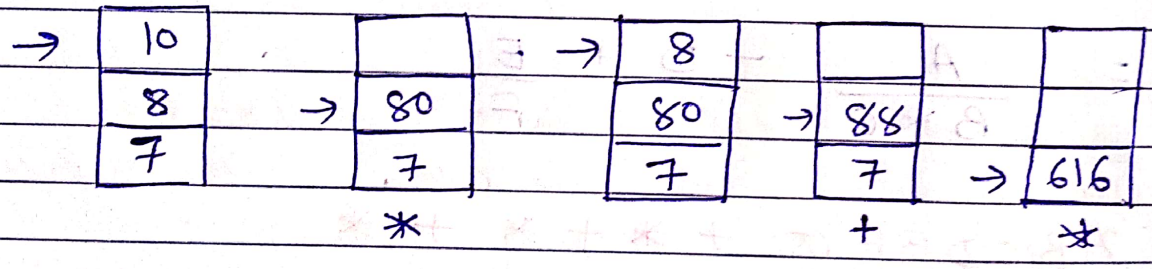
= A * (B + C * (D + E * (F + G)))

Q.5 Convert the following numerical arithmatic expression into RPN and show the Stack operations for evaluating the numerical result.

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

RPN = 34 + 26 + 10 * 8 + *





Q.6 Find out the postfix form of the expression:

(A + B) * (C * D − E) * F / G

= (AB +) * (CD * E −) * F / G

= AB + CD * E − * F * G /

Q.7 Write ALP for the following pseudo Code.

```
SUM = 0
SUM = SUM + A + B
DIF = DIF - C
SUM = SUM + DIF
```

```
        ORG 100
        CLA          } SUM = 0
        STA SUM      }
        LDA SUM      }
        ADD A        }
        ADD B        } SUM = SUM + A + B
        STA SUM      }
        LDA C        }
        CMA          }
        INC          } DIF = DIF - C
        ADD DIF      }
        STA DIF      }
        LDA SUM      }
        ADD DIF      } SUM = SUM + DIF
        STA SUM      }
        HLT
SUM,    HEX 0
  A,    DEC 5
  B,    DEC 4
  C,    DEC 10
DIF,    DEC 50
        END
```

Q.8 Write ALP to logically OR the two numbers.

```
ORG 100
LDA A        / Load A
CMA
STA T        / Store at T
LDA B        / Load B
CMA
AND T
CMA
STA RES      / Store result at RES
HLT
A,  DEC 5
T,  HEX 0
B,  DEC 4
RES, HEX 0
END
```

OR Operation

$Y = A + B$

$= \overline{\overline{A+B}}$

$= \overline{\overline{A} \cdot \overline{B}}$

Q.9 Write ALP for arithmatic right shift operation.

```
ORG 100
CLE          / Clear E to 0
LDA A        / Load AC with A
SPA          / Skip if AC is +ve
CME          / AC is -ve, Set E to 1
CIR          / Circulate right E & AC
A, HEX F234
HLT
A, H
```

Q.10 Write a Program loop using pointers and Counters to clear the contents of hex locations 500 to 5FF with 0.

```
     ORG 100
100  CLA
101  LOP, STA PTR I
102  ISZ PTR
103  ISZ CTR
104  BUN LOP
105  HLT
106  PTR, HEX 500
107  CTR, DEC -256
     END
```

Q.11 Write an ALP to Compare two memory words.

```
        ORG  100
100     LDA  WD1      / Load 1st word
101     CMA
102     INC           / 2's comp
103     ADD  WD2      / Add 2nd word
104     SZA           / Compare both num
105     BUN  UEQ      / Branch to unequal routin
106     BUN  EQL      / Branch to equal routin
107     HLT
108  WD1, ----
109  WD2, ----
        END
```