1-8-23
Tuesday

8-BIT ADDITION

/

Exp. No:- 1:-

Aim:- To write an assembly language Program to implement 8-b addition using 8085 Processor.

Algorithm:-

1. Start the Program by loading the 1st data into the accumulator.
2. Move the data to a register
3. Get the 2nd data and load it into the accumulator.
4. Add 2-register contents.
5. check for carry
6. Store the value of sum and carry in the memory location
7. Halt...

Program:-

```
LDA 8500
MOV B,A
LDA 8501
ADD B
STA 8502
RST 1.
```

Result:- Thus, the Program was Executed successfully using 8085 Processor Simulator...

Input:-

2134	8500	20
2135	8501	30.

Calculations:-

20 :- 00100000
30 :- 00110000
50 :- 01010000

Output:-

2136	8502	50

1-8-23

8-BIT SUBTRACTION

EXP.NO:- 2:-

Aim:- To write an assembly language program to implement 8-bit subtraction using 8085 processor.

Algorithm:-

1. Start the program by loading the 1st data into the accumulator.
2. Move the data to a register.
3. Get the 2nd data & load it into the accumulator.
4. Subtract 2-register contents.
5. Check for carry.
6. Store the value of sum & carry in the memory location
7. Halt.

Program:-

```
LDA 8500
MOV B,A
LDA 8501
SUB B
STA 8502
RST 1.
```

Result:- Thus, the program was executed successfully using 8085 processor simulator.

Input:-

2134	8500	20	30
2135	8501	30	-20

Calculations:-

20:- 00100000
30:- 00110000
10:- 00010000

8 4 2 1 8 4 2 1
0011 0000
00100000
00010000

20
16

16
20
⇓
20-16.

Output:-

2136 8502 10.

16-20

00010110
11100000
11110110

16 → 00010110
11101001
11101010

20 → 00100000
2's 16 → 11101010
00001010

00100000
1101 0111
11100000

8+2 = 10

1-8-23

8-BIT MULTIPLICATION

EXP.No:-3:-

3

Aim:- To write an assembly language Program to implement 8-bit multiplication using 8085 Processor..

Algorithm:-

1. Start the Program by loading a register Pair with the address of memory location.
2. Move the data to a register.
3. Get the 2nd data and load it into the accumulator.
4. Add 2-register contents.
5. Increment the value of the carry.
6. check whether the repeated addition is over.
7. Store the value of Product and the carry in memory locatio
8. Halt.

Program:-

```
LDA 8500
MOV B, A
LDA. 8501
MOV C, A
CPI 00
JZ LOOP
XRA A
LOOP1; ADDB
DCR C
JZ LOOP
JMP LOOP1
LOOP: STA 8502
RST 1.
```

Result:- Thus, the Program was Executed successfully using 8085 Processor Simulator.

Input:-

2134	8500	3
2135	8501	2

Output:-

2136	8502	6.

Calculation:-

Unsigned Multiplication:- 3:- 0011
2:- 0010

```
      1
     0011
     0010
     ----
     0101
```

```
   0011
   0010
   ----
   0000
  0011x
 0000xx
0000xxx
-------
0000110
   8421
   ----
      6
```

1-8-23

8-BIT DIVISION

EXP.NO:-4:-

4

Aim:- To write an assembly language Program to implement 8-bit division using 8085 Processor..

Algorithm:-

1. Start the Program by loading a register Pair with the address of memory location.
2. Move the data to a register.
3. Get the second data and load it into accumulator.
4. subtract the 2-register contents.
5. Increment the value of the Carry.
6. Check whether the repeated subtraction is over.
7. Store the data of quotient & remainder in memory
8. Halt.

Program:-

```
LDA 8501
MOV B,A
LDA 8500
MVI C,00
LOOP: CMP B
JC LOOP1
SUB B
INR C
JMP LOOP
STA 8503
DCR C
MOV A,C
LOOP1: STA 8502
RST 1.
```

Result:- Thus, the Program was Executed Successfully using 8085 Processor simulator...

Input:-

2134	8500	4
2135	8501	2

Output:-

2136	8502	0. (Remainder)
C	02	(Quotient)

1-8-23

16-BIT ADDITION

5

EXP. No:- 5:-

Aim:- To write an assembly language Program to implement 16-bit addition using Processor.

Algorithm:-

1. Start the Program by loading a register pair with address of 1st number.
2. copy the data to another register pair.
3. Load the second number to the 1st register pair.
4. Add the 2-register pair content.
5. Check for Carry.
6. Store the value sum & Carry in ~~mem~~ memory.
7. Terminate the Program.

Program:-

```
LDA 3050
MOV B,A
LDA 3051
ADD B
STA 3052
LDA 3053
MOV B,A
LDA 3054
ADC B
STA 3055
HLT
LHLD 2500
XCHG
LHLD 2502
DAD D
SHLD 2504
RST 1.
```

Result:- Thus, the Program was Executed successfully using 8085 Processor simulator...

Input:-

09C4	2500	11
09C5	2501	11
09C6	2502	22
09C7	2503	22

Output:-

09C8	2504	33
09C9	2505	33.

1-8-23

16-BIT SUBTRACTION

EXP. NO:- 6:-

6

Aim:- To write an assembly language Program to implement 16bit Subtraction using 8085 Processor.

Algorithm:-

1. Start the Program by loading a register pair with address of 1st number.
2. Copy the data to another register Pair
3. Load the 2nd number to 1st register.
4. Subtract the 2-register Pair Contents
5. Check the / for borrow
6. Store the value of difference & borrow in memory
7. End.

Program:-

```
LHLD 2050
XCHG
LHLD 2052.
MVI C,00
MOV A,E
SUB L
STA 2054
MOV A,D
SUB H
STA 2055
HLT
```

Result:- Thus, the Program was Executed Successfully using 8085 Processor Simulator.

Input:-

0802	2050	10
0803	2051	10
0804	2052	25
0805	2053	5

Output:-

0806	2054	5
0807	2055	5

1-8-23

16-BIT MULTIPLICATION.

EXP.No: 7

7

Aim:- To write an assembly language program to implement 16-bit multiplication using processor.

Algorithm:-

1. Load the 1st data in HL Pair
2. Move content of HL Pair to stack pointer.
3. Load the 2nd data in HL Pair and move it to DE.
4. Make H register as 00H and L register as 00H
5. ADD HL Pair and stack pointer.
6. Check for carry if carry increment it by 1 else move to next step.
7. Then move E to A & perform OR operation with accumulator and register D.
8. The value of operation is zero, then store the value else go to step-3.

Program:-

```
LHLD 2050
SPHL
LHLD 2052
XCHG
LXI H; 0000H
LXI B, 0000H
AGAIN: DAD SP
JNC START
INX B
START: DCX D
MOV A, E
ORA D
JNZ AGAIN
SHLD 2054
MOV L, C
MOV H, B
SHLD 2055
HLT.
```

Result:- Thus, the program was executed successfully using 8085 processor simulator.

Input :-

0802	2050	12
0803	2051	12
0804	2052	2
0805	2053	2

Output :-

0806	2054	24.
0807	2055	24

2050 = 12
2052 = 2
2054 = 24
2051 = 12
2053 = 2
2055 = 24

16-BIT DIVISION

8

Exp. No:- 8:-

Aim:- To Compute 16-Bit-division using Ginusim-8085

Algorithm:-

① Read dividend (16 bit) ② Read divisor ③ count ← 8.
④ Left shift dividend
⑤ subtract divisor from upper 8-bits of dividend.
⑥ If CS = 1 go to 9.
⑦ Restore dividend
⑧ Increment lower 8-bits of dividend.
⑨ count ← count − 1.
⑩ If Count = 0 goto 5.
⑪ store 8-bit (upper) as remainder & lower as quotie
⑫ HLT..

Program:-

```
LHLD 2000.
XCHG
LHLD 2020.
MOV A,D
MOV B,H
MVI C,00H
LOOP: INR C
SUB B
JNZ loop
MOV H,C
MOV A,E
MOV B,L
MVI C,00H
LP: INR C
SUB B
JNZ LP
MOV L,C
SHLD 2070
HLT.
```

Result:- Thus the Program was Executed//.

INPut:-

07D0 2000 12

07E4 2020 6

output:-

0802 2050 2.

$\Rightarrow$

$$\begin{array}{r} 00010010 \\ 00000110 \\ \hline 00011000 \end{array}$$

$$\begin{array}{r} 00010010 \\ \times\ 0110 \\ \hline 00000000 \\ 00010010 \\ 00010010 \\ 00000000 \\ \hline 00001100 \\ \hline \end{array}$$

$2^{7}\ 2^{6}\ 2^{5}\ 2^{4}\ 2^{3}\ 2^{2}\ 2^{1}\ 2^{0}$

64 32 8 4

1-8-23

SWAPPING OF NUMBERS

Exp. No 19:-

Aim:- To compute swapping of numbers using 8085 Processor

Algorithm:-

1. Load 8-bit number from memory location into accumulator.
2. Move value of accumulator into register H.
3. Load 8-bit number from next memory location into accumulator.
4. Move value of accumulator into register D.
5. Exchange both the register pairs.
6. Halt.

Program:-

```
LDA 2001
MOV B,A
LDA 2002
MOV C,A
STA 2003
MOV A,B
STA 2004
HLT.
```

Result:- Thus, the program was executed successfully using 8085 Processor simulator.

Input:

07D1 2001 5

07D2 2002 6

Output:-

07D3 2003 6

07D4 2004 5

ADDITION OF N NUMBERS

EXP.NO:-10:-

Aim:- To compute addition of N numbers using 8085 Processor

Algorithm:-

1. Load the base address of the array in HL register Pair.
2. Load the memory with data to be added.
3. Take it as Count.
4. Initialise the accumulator with 00.
5. Add Content of accumulator with Content of memory
6. Decrement Count.
7. Load Count value.
8. Repeat Step-5
9. Check whether Count has become 0.
10. Halt.

Program:-

```
LXI H,8000.
MOV C,M
MVI A,00
MOV B,A
LOOP: ADDC
JNC SKIP
INR B
SKIP: DCRC
JNZ LOOP → LXI H,8007
MOV M,A
INX H
MOV M,B
HLT.
```

Result:- Thus, the program was Executed Successfully.

Input:

IF40	8000	5
IF41	8001	12
IF42	8002	2
IF43	8003	4
IF44	8004	3
IF45	8005	2
IF46	8006	15 → output

$$\frac{5(5+1)}{2}$$

$$= \frac{5(6)}{2}$$

$$= \frac{30}{2}$$

$$= 15$$

SQUARE OF NUMBER

11

Exp. No :- 11 :-

Aim :- To compute square of number using 8085 Processor

Algorithm :-

1. Load the base address of the array in HL register Pair.
2. Assign accumulator as 0
3. Load the content of memory specified into register.
4. Add content of memory with accumulator & decrement register content by 01.
5. check if register holds 00, if so store the value of accumulator in memory location.

Program :-

```
LXI H, 8000,
XRA A
MOV B,M
LOOP: ADD M
DCR B
JNZ LOOP
STA 8001
HLT.
```

Result :- Thus, the Program was executed successfully.

Input :-

IF40 8000 6

Output :-

IF41 8001 36

ONE'S AND TWO'S COMPLEMENT

EXP.NO:- 12 :-

Aim:- To compute 1's & 2's Complement using 8085 Processor

Algorithm:

① Load the base address of the array in a register
② Move the data from memory location into accumulator
③ Convert all ones into zeros & zeros into ones.
④ Add 01 to the accumulator content.
⑤ Store the results of 1's & 2's complement.

Program:-

```
LDA 3000.
CMA
STA 3001
ADI 01
STA 3002
HLT.
```

Result:- Thus, the Program was Executed Successfully.

Input:-

0BB8 3000 250

Output:-

0BB9 3001 5

0BBA 3002 6

ROTATE LEFT OPERATION

EXP. NO:- 13:-

Aim:- To compute rotation of data in left without carry using 8085 process.

Algorithm:-

1. Load the base address of array in HL register pair
2. Move the data from memory location into accumulator
3. Shift left accumulator content for 4-times.
4. Store the result.

Program:-

```
MVI A,02
RLC
RLC
RLC
RLC
STA 2000
HLT.
```

Result:- Thus, the program was executed successfully...

Input:-

MVI A, 02

MVI A, 03 (or)

Output:-

07D0 2000 32

(or)

07D0 2000 48.

Calculations:-

02:- 00000010

00000001

10000000

01000000

00100000

$2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

$\boxed{2^5 = 32}$

ROTATE RIGHT OPERATION

14

EXP. NO:- 14:-

Aim:- To compute rotation of given data in right without carry using 8085 processor.

Algorithm:-

1. Load the base address of the array in HL register
2. Move the data from memory location into accumulator
3. Shift register the accumulator content for four times left
4. Store the result in specified location.

Program:-

```
MVI A,03
RRC
RRC
RRC
RRC
STA 2000.
HLT.
```

Result:- Thus, the Program was Executed Successfully.

Input:-

MVI A, 03

output:-

07D0 2000 48.

Calculations:-

03:- 00000011

10000001

11000000

01100000

00110000

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

32
16
48

2-8-23

LARGEST NUMBER IN AN ARRAY

EXP. No) 15:

15

Aim: To compute largest number in an array using 8085 processor.

Algorithm:

1. Load the address of the element of array in HL Pair.
2. move the count to B-register.
3. Increment the Pointer
4. Get the 1st data in A.
5. Decrement count
6. increment Pointer.
7. Compare the content of memory addressed by HL Pair with that of A.
8. if carry = 0, go to step-10 (or) if carry = 1 go to step-9
9. Move the content by HL to A-register.
10. Decrement the count.
11. check for zero of the count, if ZF = 0, go to Step-6 (or) if ZF = 1 go to next.
12. Store the largest data
13. Halt.

Program:

```
LXI H, 4200
MOV B, M.
INX H
MOV A, M
DCR B
LOOP: INX H
CMP M
JNC AHEAD
MOV A, M.
AHEAD: DCR B.
JNZ LOOP
STA 4300.
HLT.
```

Result: Thus the Program was Executed successfully.

Input :-

1068	4200	5
1069	4201	1
106A	4202	3
106B	4203	7
106C	4204	9
106D	4205	5.

Output :-

10CC	4300	9.

2-8-23 SMALLEST NUMBER IN AN ARRAY.

Exp. No:- 16:-

Aim:- To compute smallest number in an array.

Algorithm:-

1. Load the address of element
2. Move the count to B.
3. Increment Pointer.
4. Get 1st data in A.
5. Decrement Count
6. Increment Pointer.
7. Compare the content of memory addressed by HL pair with that of A
8. If carry = 0, go to step-10 (or) if carry = 1 go to step-9
9. Move the content by HL to A.
10. Decrement Count.
11. Store the smallest number
12. HALT.

Program:-

```
LXI H,4200
MOV B,M
INX H
MOV A,M
DCR B
LOOP: INX H
CMP M
JC AHEAD
MOV A,M
AHEAD: DCR B
JNZ LOOP
STA 4300
HLT
```

Result:- Thus the program was executed successfully.

Input:

1068	4200	6
1069	4201	7
106A	4202	2
106B	4203	1
106C	4204	3
106D	4205	4

Output:

10CC	4300	1

2-8-23

ASCENDING ORDER

EXP.NO:- 17 :-

17

Aim:- To compute ascending order. by using ~~ascending~~ 8085 Processor.

Algorithm:-

1. Initialise HL pair as memory pointer.
2. Get the count at 4200 into C
3. Copy it in D
4. Get the 1st value in 'A'
5. Compare it with the value at next
6. If they are out of order, exchange the contents of A
7. Decrement D content by 1.
8. Repeat step-5 & 7 till D becomes zero.
9. Decrement C content by 1.
10. repeat step-3 & 9 till C becomes zero.

Program:-

```
LXI H,4200
MOV C,M
DCR C
REPEAT : MOV D,C
LXI H, 4201
LOOP : MOV A,M
INX H
CMP M
JC SKIP
MOV B,M.
MOV M,A
DCX H
MOV M,A
INX H
SKIP : DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT.
```

Result:- Thus, the program was Executed.

Input:-

1068	4200	5
1069	4201	7
106A	4202	9
106B	4203	1
106C	4204	2
106D	4205	3

output:-

1068	4200	5
1069	4201	1
106A	4202	2
106B	4203	3
106E	4204	7
106D	4205	9

2-8-23

DESCENDING ORDER

EXP.NO:-18:

18

Aim:- To compute descending order using 8085 Processor

Algorithm:-

1. Initialise HL Pair as memory Pointer.
2. Get the count at 4200 into C.
3. Copy it in 'D'.
4. Get the 1st value in 'A'.
5. Compare it with the value at next.
6. If they are out of order, Exchange the Contents of A
7. Decrement D content by 1.
8. Repeat Step-5 & 7 till D becomes 0.
9. Decrement C content by 1.
10. Repeat step-3 & 9 till C becomes 0.

```
LXI H, 4200.
MOV C, M
DCR C
REPEAT : MOV D, C
LXI H, 4201
LOOP : MOV A, M
INX H
CMP M
JNC SKIP
MOV B, M
MOV M, A
DCX H
MOV M, A
INX H
SKIP : DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT.
```

Result:- Thus, the Program was Executed.

Input:-

1068	4200	5
1069	4201	7
106A	4202	9
106B	4203	8
106C	4204	1
106D	4205	2

output:-

1068	4200	5
1069	4201	9
106A	4202	8
106B	4203	7
106C	4204	2
106D	4205	1

2-8-23

FACTORIAL

Exp. No:- 19:-

19

Aim:- To compute factorial by using 8085 Processor.

Algorithm:-

1. Load the data into register B.
2. To start multiplication set C to 01H.
3. Jump to step-7.
4. Decrement D to multiply previous number.
5. Jump to step-3 till of D>0.
6. Take memory pointer to next & store.
7. Load E with content of D & clear 'A'.
8. Repeteadly add content of E to accumulator. B times
9. Store ~~accumulator~~ E Content to 'A'.
10. Go to step-4.

Program:-

```
LDA 2001
MOV B,A
MVI C,#01
MVI E,#01
LOOP: MOV D,C
MVI A,00H
LP: ADD E
DCR D
JNZ LP
MOV E,A
INR C
DCR B
JNZ LOOP
MOV A,E
STA 2010
HLT.
```

Result:- Thus the Program was Executed.

Input:-

07D1 2001 5

output:-

07DA 2010 120

2-8-23

BINARY TO DECIMAL

EXP.NO:- 20

20

Aim:- To convert Binary to decimal number.

Algorithm:-

1. open dev c++ app
2. Type the code
3. check whether the code is correct (or) not
4. if it is correct it shows output
5. Else it shows error and rectify it & get output

Program:-

```
# include <stdio.h>
# include <Math.h>
int Convert (long long);
int main() {
long long n;
Printf ("Enter binary number: ");
Scanf ("%.11d", &n);
printf ("%.11d in binary = %.d in decimal" n, Convert(n))
return 0;
}
int Convert (long long n) {
int dec = 0, i=0, rem;
while (n!=0){
rem = n% 10;
n/ =10;
dec+ =rem * Pow(2, i);
++i;
}
return dec;
}
```

```
# include <stdio.h>
int main()
{int temp, binary num,
decimal number = 0, base=1,
remainder;
Printf ("Enter line :");
scanf ("%d", &binary num);
temp=binary number;
while (temp>0){
remainder =temp % 10;
decimal = dec + rem * base
temp / = 10;
base *=2;
}
printf ("in decimal num
: %d \n binary
number : %d,
decimal num, binary num)
return 0;
}
```

Input:- Enter binary number =100010

Output:- ~~Binary~~ 34

Result:- Thus the Program was Executed...

2-8-23

DECIMAL TO BINARY.

EXP.No:- 29:-

Aim:- To convert decimal to Binary number.

Algorithm:-

① open devc++ app ② Type the code
③ check whether the code is correct (or) not
④ If it is correct it shows output
⑤ If it is incorrect it shows Error and rectify it and get output...

Program:-
```
#include <stdio.h>
int main(){
int decimal_num, binary_num = 0, i=1, remainder;
Printf("Enter a decimal number: ");
scanf("%d", &decimal_num);
while (decimal_num!=0){
    remainder = decimal_num % 2;
    decimal_num /=2;
    binary_num += remainder * i;
    i *=10;
}
printf("Binary number: %d \n", binary_num);
return 0;
}
```

Input:-
Enter a decimal number: 65.

Output:- Binary number: 1000001.

Result:- Thus the Program was Executed..

2-8-23

HEXADECIMAL TO DECIMAL

Exp.No:- 22:-

Aim:- To convert Hexadecimal to decimal.

Algorithm:-

1. open devc++ app
2. type the code.
3. run the code
4. If it is correct code, it shows output
5. If it is incorrect code, it shows error
6. rectify the error and run the code and get output..

Program:-

```
#include <stdio.h>
int main()
{
 int n;
 scanf("%x", &n);
 printf("%d", n);
 return 0;
}
```

A=10 B=11 C=12 D=13

Input:- 4d

4×13 = 52

4 13
2 2 1 0
→ 4×16 = 16

4×16¹ = 4×16 = 64
13×16⁰ = 13×1 = 13
~~8×16⁰~~ = 77

Output:- 77

Result:- Thus the program executed...

2-8-23

DECIMAL TO HEXADECIMAL

EXP.NO:- 23:-

Aim:- To convert decimal to Hexadecimal.

Algorithm:-

1. open dev c++ app
2. type the code
3. run the code
4. if it is correct code, it shows output
5. if it is incorrect code, it shows Error.
6. rectify Error & run the code get output.

Program:-

```
# include <stdio.h>
int main();
long int decimal number;
Printf ("Enter decimal number:");
scanf ("%d", & decimal number);
Printf ("hexadecimal is: %x", decimal number);
return 0;
}.
```

Input:-

Enter any decimal number = 45

16 | 45
2 → 13

Output:-

hexadecimal number is : 2D.

Result:- Thus the Program was Executed.

DECIMAL TO OCTAL

EXP.NO:- 24:

Aim:- To compute decimal to octal.

Algorithm:- ① open dev c++ app
② Type the program
③ rectify the errors
④ get output.

Program:-
```
#include <stdio.h>
#include <math.h>.
int ConvertDecimalToOctal;
int main(){
 int decimalnumber;
 Printf("Enter decimal number:");
 Scanf("%d", & decimal number);
 Printf("%d in decimal = %d in octal", decimalnumber,
        ConvertDecimalTOOctal (decimal number));
 return 0;
}
int ConvertDecimalTOOctal (int decimal number){
int octal number = 0, i = 1;
while (decimalnumber != 0){
octal number += (decimal number % 8) * i;
decimalnumber /= 8;
i *= 10;
}
return octalnumber;
}.
```

8 | 45
8 | 5 → 5

Input:- Enter a decimal number: 45

Output:- decimal = 55 in octal.

55

Result:- Thus, the Program was Executed.

OCTAL TO DECIMAL

25

EXP.No:-25

Aim:- To compute octal to decimal.

Algorithm:

1. open dev c++ app.
2. Type the Program.
3. rectify the errors
4. get output.

Program:-

```
# include stdio.h>
# include <math.h>
int main()
{
long int octal ;decimal = 0;
int i = 0;
Printf("Enter octal number;");
Scanf ("%ld ", & octal);
while (octal != 0) {
decimal = decimal + (octal % 10) * Pow(8, i++);
octal = octal / 10;
}
Printf ("decimal value: % ld", decimal);
return 0;
}.
```

$8^1 \times 5 = 40$

$8^0 \times 6 = 6$

$= 46$

Input:- Enter octal number = 56

Output:- decimal value = 46.

Result:- Thus, the program was executed....

3-8-23

CPU PERFORMANCE

Exp. No:- 26:-

Aim:- To compute CPU Performance in c.

Algorithm:-

1. Identify the task you want to measure the CPU Performance for.
2. Implement the code to perform the task in c.
3. Measure the execution time using suitable functions. You can use functions like clock() for basic time measurement.
4. Run the task multiple times to get average execution time and ensure consistency.
5. Calculate performance metrics like instructions per second (IPS) (or) time taken per operation depends on the task.

Program:-

```
#include <stdio.h>
int main{
float cr;
int P,Pl,i;
float CPU[5];
float CPi,ct,max;
int n=1000;
for (i=0; i<=4; i++)
{
CPU[5]=0;
}
Printf("\n Enter no.of Processors:");
scanf("%d",&P);
Pl=P;
for(i=0; i<P; i++){
Printf("\n Enter the cycles per instructions of
                Processors:");
scanf("%f",&CPi);
Printf("\n Enter the Clock rate in GHz:");
scanf("%f",&cr);
ct=1000* CPi/cr;
Printf("The CPU time is: %f",ct);
```

```
cpu[i] = ct;
}
max = cpu[0];
printf("%f", max);
for(i=0; i<p1; i++)
{
if (cpu[i]<=max)
max = cpu[i];
}
printf("\n The Processor has lowest Execution
time is: %f", max);
return 0;
}
```

Output:-

```
Enter the no. of Processors : 3
Enter the cycles per instructions of Processor = 1.5.
Enter the clockrate in GHz : 3
The CPU time is : 500.000000
Enter the cycles per instructions of Processor = 1.
Enter the clockrate in GHz : 2
The CPU time is : 454.545441
Enter the cycles per instruction of Processor : 2
Enter the clockrate in GHz = 4
The CPU time is : 500.000000
The Processor has lowest Execution time
is : 454.545441
```

Result:- Thus the Program was Executed.

8-8-23:- SINGLE PRECISION FORMAT

2

EXP. NO:- 27:-

Aim:- To compute Single Precision format in C.

Algorithm:-

1. Determine the sign of number & set the sign bit accordingly.
2. Convert the absolute value of decimal number to binary
3. Normalise the binary representation so, that there is a single 1 to the left of binary. Adjust Exponent
4. Remove the leading 1.
5. Pad (or) truncate fractional part to 23 bits
6. Round the fraction according to IEEE 754 rounding rules
7. Convert exponents fits within 8-bits
8. Combine sign bit, Exponent & fraction to create the 32-bit single precision representation.

Program:-

```
# include <stdio.h>
void PrintBinary(int n, int i){
int K;
for(K=i-1; K>=0; k--){
if((n>>K)&1)
printf("1");
Else
printf("0");
}
}
typedef union{
float f;
Struct{
unsigned int mantissa : 23;
unsigned int Exponent : 8;
Unsigned int sign : 1; }
raw; } my float;
void PrintIEEE(myfloat var)
{printf("%d !", var.raw.sign);
PrintBinary(var.raw.Exponent, 8);
printf("1!");
PrintBinary(var.raw.Exponent, 23);
Printf("\n");
}int main()
{ my float var;
var.f = 1259.125;
Printf("IEEE 754 representation of %f is:\n",
var.f);
PrintIEEE(var);
return 0;
}
```

Output:- IEEE 754 representation of 1259.125000 is:

0|10001001|00111010100 10000000000.

Result:- Thus, the Program was executed.

3-8-23

TWO STAGE PIPELINE

(Exp.No:28)

Aim: To compute two stage pipeline in C.

Algorithm: ① I/P data is fed into the 1st stage of pipeline

② Perform data preprocessing & feature engineering to prepare next stage.

③ result from stage-1 serve as i/p for stage-2.

④ Apply algorithm to process & generate final o/p.

Program:

```
#include<stdio.h>
int main() {
int counter=1,a,b,choice,res,ins;
Printf("Enter number 1: ");
scanf("%d",&a);
Counter=Counter+1;
Printf("Enter number 2: ");
scanf("%d",&b);
counter=counter+1;
printf("1-Add:\n2-sub:\n
        3-mult:\n4-Div:");
scanf("%d",&choice);
switch (choice) {
case 1: Printf("performing add\n");
res a+b;
counter=counter+1;
break;
Case 2: Printf("performing sub\n");
res a-b;
Counter=Counter+1;
break;
case3: Printf("performing mult\n");
res a*b;
Counter=Counter+1;
break;
case4: Printf("performing div\n");
res a/b;
counter=counter+1;
break;
default: Printf("wrong i/p");
break;
Printf("The cycle value is %d
\n", Counter);
Printf("Enter no. of instructions:");
scanf("%d",&ins);
int performance_measure=
ins/counter;
Printf("The performance
measure is: %d\n",
performance_measure);
return 0;
}.
```

output: Enter number 1: 56
Enter number 2: 67
1. ADD
2. SUB
3. Mult
4. Div: 7
wrong i/p the cycle
value is: 3
Enter the no. of Instructions: 5
The performance
measure is: 1.

Result: Thus the Program was Executed.

3-8-23

4-STAGE PIPELINE:

EXP. NO:- 29:-

Aim:- To compute 4-stage pipeline in C.

Algorithm: ① Fetch the next instruction from memory.
② Decode stage, the instruction decoded & extract the necessary control signals & operands.
③ Execute stage: Perform operation.
④ write back stage: write result of execution back to appropriate register.

Program:-

```
#include <stdio.h>
void main() {
int counter=0;
int i/p;
int num1,num2;
int op;
int res;
int ins;
int performance_measure=0;
Printf("\n Enter 1st value: ");
scanf("%d",&num1);
counter+=1;
Printf("\n Enter the 2nd value: ");
scanf("%d",&num2);
counter+=1;
Printf("\n Enter the option: \n
1) ADD 2) SUB 3) MUL 4) DIV");
scanf("%d",&op);
Switch (op) {
case 1:
Printf("performing add operation");
res = num1 + num2;
counter+=1;
break;
case 2:
Printf("performing sub operation");
res = num1 - num2;
counter+=1;
break;
case 3:
Printf("performing mul operation");
res = num1 * num2;
counter+=1;
break;
case 4:
if (num2==0){
Printf("\n Denominator can't be
zero");
}
else
{
Printf("performing divi
operation");
res = num1 / num2;
counter+=1;
break;
}
default;
Printf("Invalid case...");
counter+=3;
break;
}
printf("\n cycle
value is: %d",
counter);
printf("Enter no. of
instructions");
```

```
scanf("%d", &ins);
performance_measure = ins/counter;
printf("\n Performance measure is: %d",
        performance_measure);
}.
```

Output:

Enter 1st value = 43
Enter 2nd value = 45.
Enter the option:
1) ADD
2) SUB
3) MUL
4) DIV - 5
Invalid case...
Cycle value is : 5
Enter the no. Instruction - 5.
Performance measure is : 1.

3-8-23

HALF ADDER

EXP. NO:- 30:-

Aim:- To design & implement 2-bit half adder using logicsim.

Algorithm:-

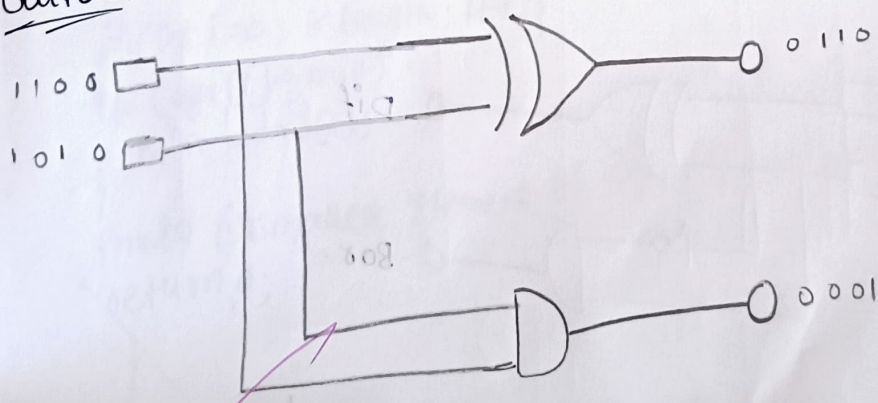
1. Pick and Place necessary gates
2. Insert 2-i/p's into the canvas.
3. connect the I/P's to the XOR & AND gate.
4. Insert 2-o/p's into canvas.
5. Make the connections using connecting wires.
6. verify truth table.

Truth Table:-

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = A XOR B
C = A AND B.

Output:-



Result:- Thus the Program was designed & implemented.

3-8-23

HALF SUBTRACTOR

EXP.NO:- 31:-

Aim:- To design & implement the 2-bit half subtractor using logicsim.

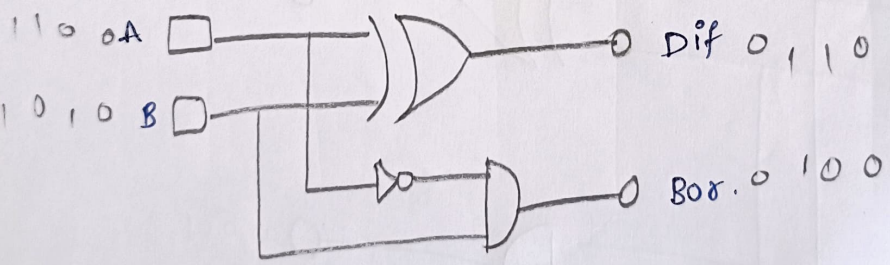
Algorithm:-

1. Pick & Replace the necessary gates.
2. Insert 2-i/p's into Canvas.
3. Connect the i/p's to the OR gate, AND & NOT gate.
4. Insert 2-o/p's into Canvas.
5. Make the connection using the connecting wires.
6. verify the truth table..

Truth Table:-

A	B	Dif	Bor
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$Diff = A'B + AB'$
$Borrow = A'B$



Result:- Thus the Program was designed & implemented successfully.

3-8-23

SMALLEST NUMBER

EXP.NO:- 32 :-

Aim:- To Find Smallest number in C.

Algorithm:-

① Start ② Initialise arr[] = {18, 45, 66, 23, 19, 10, 52, 71}
③ length = size of (arr)/size of (arr[0]);
④ Min = arr[0].
⑤ set i=0. Repeat step-6 & 7 until i<length.
⑥ if (arr[i] < min) min = arr[i]
⑦ i=i+1.
⑧ print.
⑨ Return 0 ⑩ END.

Program:-

```
#include<stdio.h>
int main()
{ int arr[] = { 18, 45, 66, 23, 19, 10, 52, 71 };
  int length = size of (arr)/size of (arr[0]);
  int min = arr[0];
  for(int i=0; i<length; i++){
  if (arr[i] < min).
     min = arr[i];
  }
  printf ("smallest Element in array: %d \n", min);
  return 0;
}
```

~~output:-~~

Input:- arr[] = {18, 45, 66, 23, 19, 10, 52, 71}

output:- Smallest number/Element = 10.

Result:- Thus, the Program was executed.

4-8-23

FULL ADDER

EXP. NO: 33 :

Aim:- To design & implement full adder using logic sim.

Algorithm:-

1. Pick and Place the necessary gates
2. Insert 3-i/p's into Canvas.
3. Connect the i/p to XOR, AND, OR gate
4. Insert 2-o/p's into the Canvas.
5. Make the Connections using Connecting wires.
6. verify the Truth Table.

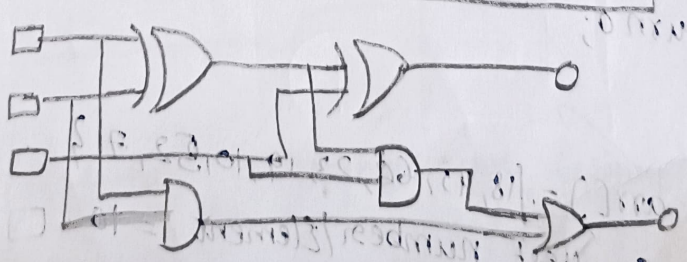
Truth Table:-

I/P			O/P	
A	B	Cin	Sum	carry
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

Sum = $(A \oplus B) \oplus C$.

Carry = $AB + (A \oplus B)$.

Output:-



Result:- Thus the Program was designed & implemented.

4-8-23

FULL SUBTRACTOR

9

EXP.NO:- 34:-

Aim:- To design & implement full subtractor using logicsim.

Algorithm:-

① Pick & Place the necessary gates ② Insert 3-i/p into Canvas ③ Connect the I/Ps to XOR, AND, OR gate. ④ Insert 2-o/P's into canvas. ⑤ make the connections using connecting wires. ⑥ verify the truth Table.

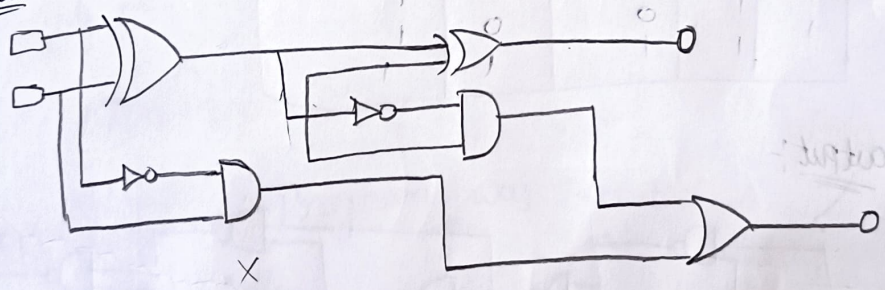
Truth Table:-

I/P's			O/P's	
A	B	Borrowin	Diff	Borrow.
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

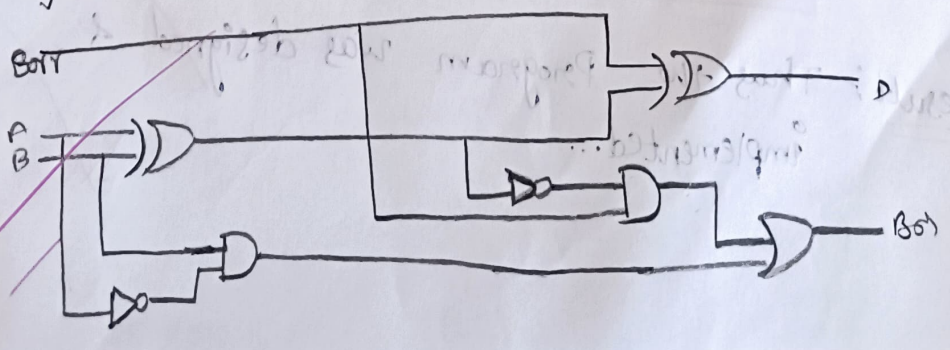
$Diff = (A \oplus B) \oplus$ Borrowin

$Borrow = A'B + (A \oplus B)'$

output:-



Result:- Thus the Program was designed & implemented..



4-8-23

FULL ADDER USING NAND GATES

Exp. No:- 35:-

Aim:- To design & implement the full adder using NAND gate.

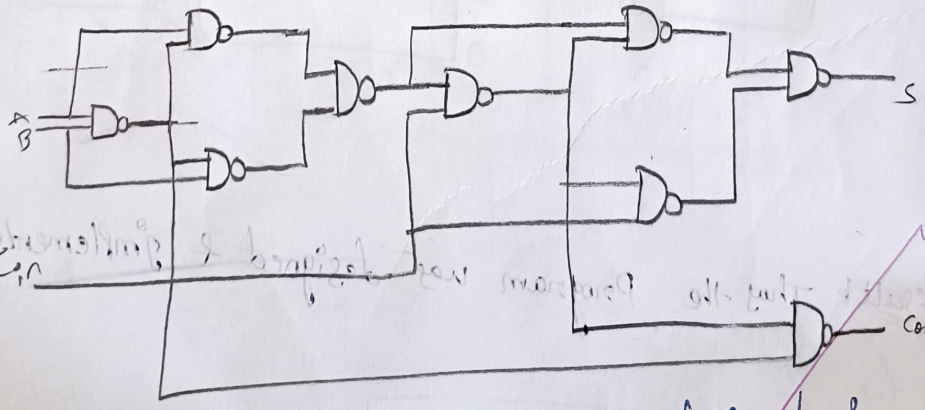
Algorithm:-

① Pick and place the necessary gates ② Insert 3/ips
③ Connect circuits ④ Insert 2-O/P's.
⑤ Make connections using the wires.
⑥ Verify truth Table..

Truth Table:-

I/P			O/P	
A	B	Cin	Sum	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

output:-



Result:- Thus the Program was designed & implemented...

4-8-23

AND OPERATION

EXP.NO:- 36:-

Aim:- To compute AND operation using 8085 GNUSim.

Algorithm:- ① open GNUSIM & create a new circuit.

② From the left Panel, locate "logic gate.

③ Drag AND gate component onto the Canvas. Write the Code.

④ After typing code save it and Execute.

⑤ It will store data in 2500.

Program:-

```
MVI A, 06
MVI B, 04
ANA B
STA 2500
HLT
```

Input:- MVI A, 06
MVI B, 04

output:- 07D0 2000 7.

Result:- Thus the Program was Executed Successfully.

4-8-23

OR OPERATION

Exp.No:- 37:-

Aim:- To compute OR operation in Gnusim.

Algorithm:-

1. open the Gnusim-8085
2. Type the code
3. save the code
4. Run the code and get output.

Program:-

```
MVI A, 07
MVI B, 06
ORA B
STA 2000
HLT.
```

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	1

```
0000 0111
0000 0110
---------
0000 0111
```

Input:- MVI A, 07
MVI B, 06.

output:- 07D0 2000 7.

Result:- Thus the Program was Executed successfully.

4-8-23

XOR OPERATION

EXP. NO :- 38 :-

Aim :- To compute XOR-operation using GNUsim.

Algorithm :-

1. Open the Gnusim-8085 app
2. Type the Code
3. Save it
4. Run the code after saving
5. and Get output.

Program :-

```
MVI A, 03
MVI B, 04
XRA B
STA 2000
HLT.
```

Input :- MVI A, 03
MVI B, 04.

Output :- 07D0 2000 7.

Result :- Thus the Program was Executed successfully

4-8-23

LCM

Exp. No:- 39:-

Aim:- To Find LCM of 12 & 16. in C-Program.

Algorithm:-

1. Initialise 2 - numbers for 12 & 16.
2. Find & Store maximum of num1 & num2 to Separate variables.
3. Max (16).
4. If max is divisible by num(16 % 12 == 0?) and num2 (16 % 16 == 0?), max is the LCM (16)
5. Hence Print it.

Program:-

```
# include gcd(int a, int b){
 if (b==0){
  return a;
 }
 return gcd (b, a% b);
 }
int lcm (int a, int b)
{ return (a * b) / gcd (a,b);
}
int main(){
 int num1 =12;
 int num2 =16;
 int result = lcm (num1, num2);
Printf("The LCM of %d & %d is %d,\n",
        num1, num2, result);
return 0;
}.
```

Input:-

num1:- 12
num2:- 16.

Output:-

LCM of 12 & 16 is 48.

Result:- Thus the Program was Executed...

4-8-23

GCD

Exp.No:- 40:-

Aim:- To Find GCD of 16 & 12 in C.

Algorithm:-

1. Start
2. Declare Variable n_1, n_2, gcd=1, i=1.
3. I/P n_1 & n_2.
4. Repeat until $i <= n_1$ & $i <= n_2$.
5. if $n1 \% i == 0$ && $n_2 \% i == 0$;
6. gcd = i.
7. Print gcd.
8. Stop.

Program:-

```
int gcd (int a, int b){
while (b!=0){
  int temp=b;
  b=a%b;
  a=temp;
}
return a;
}
int main(){
  int num1=16;
  int num2=12;
  int result=gcd(num1,num2);
  Printf("The GCD of %d & %d is %d.\n", num1,num2,
          result);
  return 0;
}
```

Input:- num1 = 16
num2 = 12.

Output:- The GCD of 16 & 12 is 4.

Result:- Thus the Program was Executed.