

**EXP NO: 1****AIM:**

To write an assembly language program to implement 8-bit addition using 8085 processor.

**ALGORITHM:**

- 1) Start the program by loading the first data into the accumulator.
- 2) Move the data to a register.
- 3) Get the second data and load it into the accumulator.
- 4) Add the two 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
```

**INPUT:****OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 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 first data into the accumulator.
- 2) Move the data to a register.
- 3) Get the second data and load it into the accumulator.
- 4) Subtract the two register contents.
- 5) Check for borrow.
- 6) Store the difference and borrow in the memory location.
- 7) Halt.

### PROGRAM:

```
LDA 8000
MOV B, A
LDA 8001
SUB B
STA 8002
RST 1
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 8-BIT MULTIPLICATION

### EXP NO: 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 second data and load it into the accumulator.
- 4) Add the two 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 the memory location.
- 8) Halt.

### PROGRAM:

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

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 8-BIT DIVISION

**EXP NO: 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 the accumulator.
- 4) Subtract the two register contents.
- 5) Increment the value of the carry.
- 6) Check whether the repeated subtraction is over.
- 7) Store the value of quotient and the remainder in the memory location.
- 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
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 16-BIT ADDITION

**EXP NO: 5**

**AIM:** To write an assembly language program to implement 16-bit addition using 8085 processor.

**ALGORITHM:**

- 1) Start the program by loading a register pair with address of 1<sup>st</sup> number.
- 2) Copy the data to another register pair.
- 3) Load the second number to the first register pair.
- 4) Add the two register pair contents.
- 5) Store the result in memory locations.
- 6) Terminate the program.

**PROGRAM:**

```
LHLD 2500
XCHG
LHLD 2502
DAD D
SHLD 2504
HLT
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 16-BIT SUBTRACTION

### EXP NO: 6

**AIM:** To write an assembly language program to implement 16-bit subtraction using 8085 processor.

### ALGORITHM:

- 1) Start the program by loading a register pair with address of 1<sup>st</sup> number.
- 2) Copy the data to another register pair.
- 3) Load the second number to first register pair.
- 4) Subtract the two register pair contents.
- 5) Check for borrow.
- 6) Store the value of difference and borrow in memory locations.
- 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
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 16-BIT MULTIPLICATION

### EXP NO: 7

**AIM:** To write an assembly language program to implement 16-bit multiplication using 8085 processor.

### ALGORITHM:

- 1) Load the first data in HL pair.
- 2) Move content of HL pair to stack pointer.
- 3) Load the second 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 and 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 2056
HLT
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## 16-BIT DIVISION

### EXP NO: 8

**AIM:** To write an assembly language program to implement 16-bit divided by 8-bit using 8085 processor.

### 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 go to 5
- 11) Store upper 8-bit dividend as remainder and lower 8-bit as quotient
- 12) Stop

### 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
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.



## FACTORIAL OF A GIVEN NUMBER

### EXP NO: 9

**AIM:** To find the factorial of a given number using 8085 microprocessor.

### ALGORITHM:

- 1) Load the data into register B
- 2) To start multiplication set D to 01H
- 3) Jump to step 7
- 4) Decrements B to multiply previous number
- 5) Jump to step 3 till value of B>0
- 6) Take memory pointer to next location and store result
- 7) Load E with contents of B and clear accumulator
- 8) Repeatedly add contents of D to accumulator E times
- 9) Store accumulator content to D
- 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
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## **LARGEST NUMBER IN AN ARRAY**

**EXP NO: 10**

**AIM:** To find the largest number from an array using 8085 processor.

**ALGORITHM:**

- 1) Load the address of the first element of the array in HL pair.
- 2) Move the count to B register.
- 3) Increment the pointer.
- 4) Get the first data in A register.
- 5) Decrement the count.
- 6) Increment the pointer.
- 7) Compare the content of memory addressed by HL pair with that of A register.
- 8) If carry=0, go to step 10 or if carry=1 go to step 9
- 9) Move the content of memory addressed by HL to A register.
- 10) Decrement the count.

**PROGRAM:**

```
LXI H,2050
MOV C,M
DCR C
INX H
MOV A,M
LOOP1: INX H
CMP M
JNC LOOP
MOV A,M
LOOP: DCR C
JNZ LOOP1
STA 2058
HLT
```

**INPUT:****OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8086 processor simulator.

## SMALLEST NUMBER IN AN ARRAY

### EXP NO: 11

**AIM:** To find the smallest number from an array using 8085 processor.

### ALGORITHM:

- 1) Load the address of the first element of the array in HL pair.
- 2) Move the count to B register.
- 3) Increment the pointer.
- 4) Get the first data in A register.
- 5) Decrement the count.
- 6) Increment the pointer.
- 7) Compare the content of memory addressed by HL pair with that of A register.
- 8) If carry=1, go to step 10 or if carry=0 go to step 9
- 9) Move the content of memory addressed by HL to A register.
- 10) Decrement the count.

### PROGRAM:

```
LXI H,2050
MOV C,M
DCR C
INX H
MOV A,M
LOOP1: INX H
CMP M
JC LOOP
MOV A,M
LOOP: DCR C
JNZ LOOP1
STA 2058
HLT
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## ASCENDING ORDER

### EXP NO: 12

**AIM:** To compute ascending order of an array using 8085 processor.

### ALGORITHM:

- 1) Initialize HL pair as memory pointer.
- 2) Get the count at memory and load it into C register
- 3) Copy it in D register (for bubble sort (N-1)) times required).
- 4) Get the first value in A register.
- 5) Compare it with the value at next location.
- 6) If they are out of order, exchange the contents of A register and memory.
- 7) Decrement D register content by 1
- 8) Repeat step 5 and 7 till the value in D register become zero.
- 9) Decrement the C register content by 1.
- 10) Repeat steps 3 to 9 till the value in C register becomes zero.

### PROGRAM:

```
LOOP: LXI H,3500
MVI D,00
MVI C,05
LOOP1: MOV A,M
INX H
CMP M
JC LOOP2
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
MVI D,01
LOOP2: DCR C
JNZ LOOP1
MOV A,D
RRC
JC LOOP
HLT
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## DESCENDING ORDER

### EXP NO: 13

**AIM:** To compute descending order of an array using 8085 processor.

### ALGORITHM:

- 1) Initialize HL pair as memory pointer.
- 2) Get the count at memory and load it into C register
- 3) Copy it in D register (for bubble sort (N-1)) times required).
- 4) Get the first value in A register.
- 5) Compare it with the value at next location.
- 6) If they are out of order, exchange the contents of A register and memory.
- 7) Decrement D register content by 1
- 8) Repeat step 5 and 7 till the value in D register become zero.
- 9) Decrement the C register content by 1.
- 10) Repeat steps 3 to 9 till the value in C register becomes zero.

### PROGRAM:

```
LOOP: LXI H,3500
MVI D,00
MVI C,05
LOOP1: MOV A,M
INX H
CMP M
JNC LOOP2
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
MVI D,01
LOOP2: DCR C
JNZ LOOP1
MOV A,D
RRC
JC LOOP
HLT
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## ADDITION OF N NUMBERS

### EXP NO: 14

**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) Initialize the accumulator with 00.
- 5) Add content of accumulator with content of memory.
- 6) Decrement count.
- 7) Load count value to memory location.
- 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: ADD C
JNC SKIP
INR B
SKIP: DCR C
JNZ LOOP
LXI H,8007
MOV M,A
INX H
MOV M,B
HLT
```



**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## SWAPPING OF NUMBERS

### EXP NO: 15

**AIM:** To compute swapping of numbers using 8085 processor.

### ALGORITHM:

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

### PROGRAM:

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

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## SQUARE OF NUMBER

### EXP NO: 16

**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 location specified into register.
- 4) Add content of memory location with accumulator and 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
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## ONEs AND TWOs COMPLEMENT

### EXP NO: 17

**AIM:** To compute one's and two's complement using 8085 processor.

### ALGORITHM:

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

### PROGRAM:

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

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## ROTATE LEFT OPERATION

### EXP NO: 18

**AIM:** To compute rotation of given data in left without carry using 8085 processor.

### ALGORITHM:

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

### PROGRAM:

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

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## ROTATE RIGHT OPERATION

### EXP NO: 19

**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 pair.
- 2) Move the data from memory location into accumulator.
- 3) Shift right the accumulator content for four times left.
- 4) Store the result in the specified location.

### PROGRAM:

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

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.

## LOGICAL OPERATIONS

**EXP NO: 20**

**AIM:** To compute various logical operations using 8085 processor.

**ALGORITHM:**

- 1) Load data to accumulator.
- 2) Load another data in register
- 3) Perform logical operations like AND, OR and XOR (Use ANA, ORA, XRA) with the accumulator content.
- 4) Store the result in specified memory location.

**PROGRAM:**

**AND OPERATION:**

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

**OR OPERATION:**

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

**XOR OPERATION:**

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

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using 8085 processor simulator.



## DECIMAL TO BINARY CONVERSION

### EXP NO: 21

**AIM:** To write a C program to implement decimal to binary conversion.

### ALGORITHM:

- 1) Check if your number is odd or even.
- 2) If it's even, write 0 (proceeding backwards, adding binary digits to the left of the result).
- 3) Otherwise, if it's odd, write 1 (in the same way).
- 4) Divide your number by 2 (dropping any fraction) and go back to step 1. Repeat until your original number is 0.

### PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a[10],n,i;
    printf("Enter the number to convert: ");
    scanf("%d",&n);
    for(i=0;n>0;i++)
    {
        a[i]=n%2;
        n=n/2;
    }
    printf("\nBinary of Given Number is=");
    for(i=i-1;i>=0;i--)
    {
        printf("%d",a[i]);
    }
    return 0;
}
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using DevC++.

## HEXADECIMAL TO DECIMAL CONVERSION

### EXP NO: 22

**AIM:** To write a C program to implement hexadecimal to decimal conversion.

### ALGORITHM:

- 1) Start from the right-most digit. Its weight (or coefficient) is 1.
- 2) Multiply the weight of the position by its digit. Add the product to the result. (0=0, 1=1, 2=2, ... 9=9, A=10, B=11, C=12, D=13, E=14, F=15)
- 3) Move one digit to the left. Its weight is 16 times the previous weight.
- 4) Repeat 2 and 3 until you go through all hexadecimal digits.

### PROGRAM:

```
#include<stdio.h>

int main()
{
    int n;
    printf("enter the hex decimal number");
    scanf("%x",&n);
    printf("the decimal value is:%d",n);
    return 0;
}
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using DevC++.

## DECIMAL TO OCTAL CONVERSION

### EXP NO: 23

**AIM:** To write a C program to implement decimal to octal conversion.

### ALGORITHM:

- 1) Store the remainder when the number is divided by 8 in an array.
- 2) Divide the number by 8 now
- 3) Repeat the above two steps until the number is not equal to 0.
- 4) Print the array in reverse order now.

### PROGRAM:

```
#include <stdio.h>

int main()
{
    long decimal, remainder, quotient, octal=0;
    int octalnum[100], i = 1, j;
    printf("Enter the decimal number: ");
    scanf("%ld", &decimal);
    quotient = decimal;
    while (quotient != 0)
    {
        octalnum[i++] = quotient % 8;
        quotient = quotient / 8;
    }
    for (j = i - 1; j > 0; j--)
        octal = octal*10 + octalnum[j];
    printf("Equivalent octal value of decimal no %d is: %d ", decimal, octalnum);
    return 0;
}
```

### INPUT:

### OUTPUT:

**RESULT:** Thus the program was executed successfully using DevC++.

## BINARY TO DECIMAL CONVERSION

### EXP NO: 24

**AIM:** To write a C program to implement binary to decimal conversion.

### ALGORITHM:

- 1) Start
- 2) Read the binary number from the user, say 'n'
- 3) Initialize the decimal number, d=0
- 4) Initialize i=0
- 5) Repeat while n != 0:
  - i. Extract the last digit by: remainder = n % 10
  - ii. n = n/10
  - iii. d = d + (remainder \* 2<sup>i</sup>)
  - iv. Increment i by 1
- 6) Display the decimal number, d
- 7) Stop

### PROGRAM:

```
#include <stdio.h>

void main()
{
    int num, binary_num, decimal_num = 0, base = 1, rem;
    printf (" Enter a binary number with the combination of 0s and 1s \n");
    scanf ("%d", &num);
    binary_num = num;
    while ( num > 0)
    {
        rem = num % 10;
        decimal_num = decimal_num + rem * base;
        num = num / 10;
        base = base * 2;
    }

    printf ( " The binary number is %d \t", binary_num);
    printf (" \n The decimal number is %d \t", decimal_num);
}
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using DevC++.

## TWO STAGE PIPELINE

### EXP NO: 25

**AIM:** To write a C program to implement two stage pipelining.

### PROCEDURE:

- Step 1: Start
- Step 2: Initialize the counter variable to 1.
- Step 3: Prompt the user to enter the first number (a).
- Step 4: Read the first number (a) from the user.
- Step 5: Increment the counter by 1.
- Step 6: Prompt the user to enter the second number (b).
- Step 7: Read the second number (b) from the user.
- Step 8: Increment the counter by 1.
- Step 9: Display the menu of operations: Addition, Subtraction, Multiplication, and Division.
- Step 10: Prompt the user to select an operation (choice).
- Step 11: Read the choice from the user.
- Step 12: Use a switch statement to perform the operation based on the selected choice:
  - 12.1 For choice 1: Perform addition ( $res = a + b$ ). Increment the counter by 1.
  - 12.2 For choice 2: Perform subtraction ( $res = a - b$ ). Increment the counter by 1.
  - 12.3 For choice 3: Perform multiplication ( $res = a * b$ ). Increment the counter by 1.
  - 12.4 For choice 4: Perform division ( $res = a / b$ ). Increment the counter by 1.
  - 12.5 For any other choice: Display "Wrong input".
- Step 13: Display the value of the counter (the number of cycles taken).
- Step 14: Prompt the user to enter the number of instructions (ins).
- Step 15: Read the number of instructions (ins) from the user.
- Step 16: Calculate the performance measure by dividing the number of instructions (ins) by the counter and store it in the performance measure variable.
- Step 17: Display the performance measure
- Step 18: End

### 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-Addition:\n2-Subtraction:\n3-Multiplication:\n4-Division:");
scanf("%d",&choice);
switch(choice)
{
    case 1: printf("Performing addition\n");
            res = a+b;
            counter = counter+1;
            break;
    case 2: printf("Performing subtraction\n");
            res = a-b;
            counter = counter+1;
            break;
    case 3: printf("Performing Multiplication\n");
            res = a*b;
            counter = counter+1;
            break;
    case 4: printf("Performing Division\n");
            res = a/b;
            counter = counter+1;
            break;
    default: printf("Wrong input");
            break;
}
printf("The cycle value is:%d\n",counter);
printf("Enter the number of instructions:");
scanf("%d",&ins);
int performance_measure = ins/counter;
printf("The performance measure is:%d\n",performance_measure);
return 0;
}

```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using DevC++.



## CPU PERFORMANCE

### EXP NO: 26

**AIM:** To write a C program to implement CPU performance measures.

### ALGORITHM:

Step 1: start

Step 2: Declare the necessary variables: cr (clock rate), p (number of processors), p1 (a copy of the number of processors), i (loop variable), and cpu (array to store CPU times).

Step 3: Initialize the cpu array elements to 0.

Step 4: Prompt the user to enter the number of processors (p).

Step 5: Store the value of p in p1.

Step 6: Start a loop from 0 to p-1:

- a. Prompt the user to enter the cycles per instruction (cpi) for the current processor.
- b. Prompt the user to enter the clock rate (cr) in GHz for the current processor.
- c. Calculate the CPU time (ct) using the formula:  $ct = 1000 * cpi / cr$ .
- d. Display the CPU time for the current processor.
- e. Store the CPU time in the cpu array at index i.

Step 7: Set max as the first element of the cpu array.

Step 8: Start a loop from 0 to p1-1:

- a. If the CPU time at index i is less than or equal to max, update max to the current CPU time.

Step 9: Display the processor with the lowest execution time (max).

Step 10: Exit the program.

### PROGRAM:

```
#include <stdio.h>

int main()
{
    float cr;
    int p,p1,i;
    float cpu[5];
    float cpi,ct,max;
    int n=1000;
    for(i=0;i<=4;i++)
    {
        cpu[i]=0;
    }
    printf("\n Enter the number of processors:");
    scanf("%d",&p);
    p1=p;
    for(i=0;i<p;i++)
    {
        printf("\n Enter the Cycles per Instrction of processor:");
```

```
scanf("%f",&cpi);
printf("\n Enter the clockrate in GHz:");
scanf("%f",&cr);
ct=1000*cpi/cr;
printf("The CPU time is: %f",ct);
cpu[i]=ct;
}
max=cpu[0];
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;
}
```

**INPUT:**

**OUTPUT:**

**RESULT:** Thus the program was executed successfully using DevC++.

## HALF ADDER

**EXP.NO: 27**

**AIM:**

To design and implement the two bit half adder using Logisim simulator.

**PROCEDURE:**

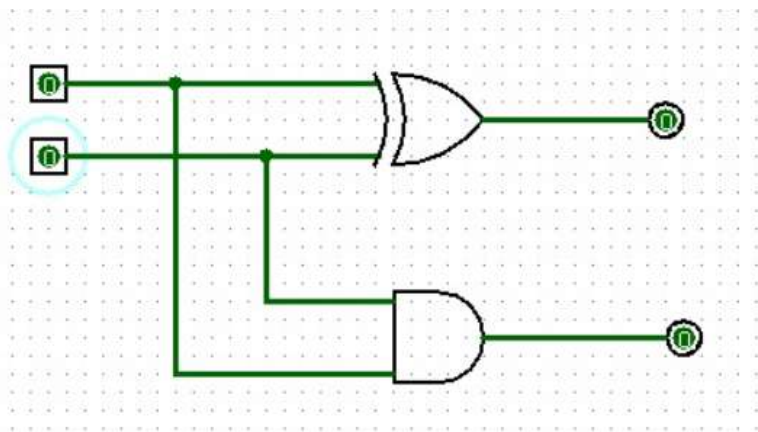
- 1) Pick and place the necessary gates.
- 2) Insert 2 inputs into the canvas.
- 3) Connect the inputs to the XOR gate and AND gate.
- 4) Insert 2 outputs into the canvas.
- 5) Make the connections using the connecting wires.
- 6) Verify the truth table.

**TRUTH TABLE:**

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

$$S = A \text{ XOR } B \quad C = A \text{ AND } B$$

**OUTPUT**



**RESULT:** Thus 2-bit half adder has been designed and implemented successfully using logisim simulator.

## TWO BIT HALF SUBTRACTOR

**EXP.NO: 28**

**AIM:**

To design and implement the two bit half subtractor using Logisim simulator.

**PROCEDURE:**

- 1) Pick and place the necessary gates.
- 2) Insert 2 inputs into the canvas.
- 3) Connect the inputs to the OR gate, AND gate and NOT gate.
- 4) Insert 2 outputs into the canvas.
- 5) Make the connections using the connecting wires.
- 6) Verify the truth table.

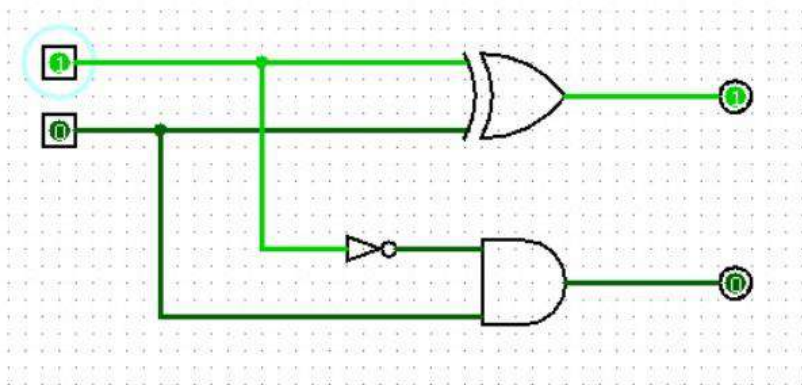
**TRUTH TABLE:**

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$\text{Diff} = A'B + AB'$$

$$\text{Borrow} = A'B$$

**OUTPUT**



**RESULT:** Thus 2-bit half subtractor has been designed and implemented successfully using logisim simulator.

## FULL ADDER

**EXP.NO: 29**

**AIM:**

To design and implement the full adder using Logisim simulator.

**PROCEDURE:**

- 1) Pick and place the necessary gates.
- 2) Insert 3 inputs into the canvas.
- 3) Connect the inputs to the XOR gate, AND gate and OR gate.
- 4) Insert 2 outputs into the canvas.
- 5) Make the connections using the connecting wires.
- 6) Verify the truth table.

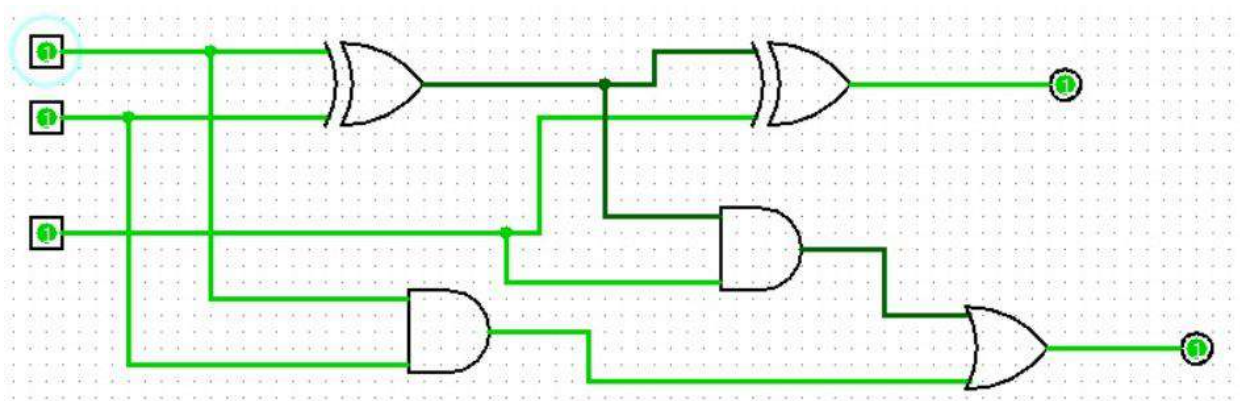
**TRUTH TABLE:**

Inputs			Outputs	
A	B	C <sub>in</sub>	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

$$\text{Sum} = (A \oplus B) \oplus C_{in}$$

$$\text{Carry} = A.B + (A \oplus B).C_{in}$$

**OUTPUT**



**RESULT:** Thus full adder has been designed and implemented successfully using logisim simulator.



**EXP.NO: 30**

**AIM:**

To design and implement the full subtractor using Logisim simulator.

**PROCEDURE:**

- 1) Pick and place the necessary gates.
- 2) Insert 3 inputs into the canvas.
- 3) Connect the inputs to the XOR gate, AND gate and OR gate.
- 4) Insert 2 outputs into the canvas.
- 5) Make the connections using the connecting wires.
- 6) Verify the truth table.

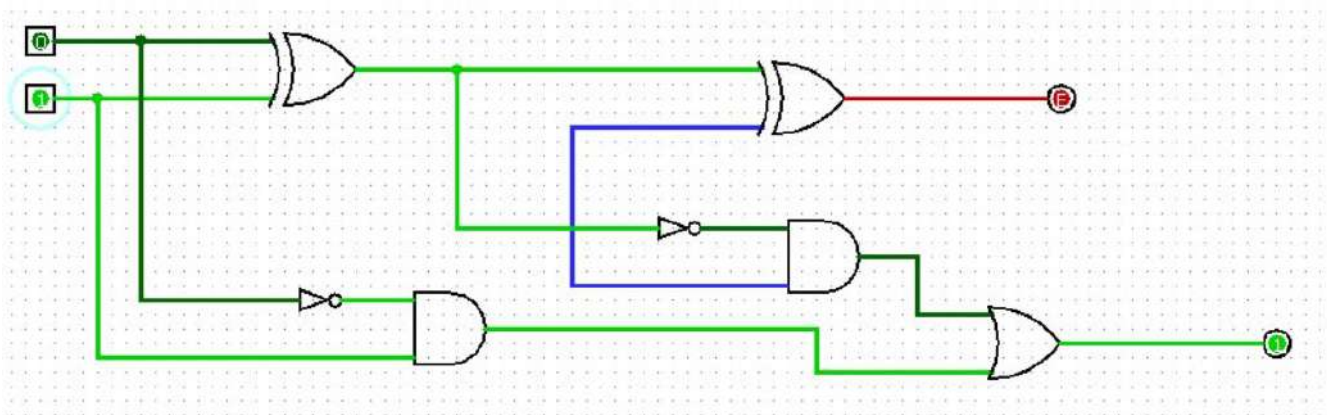
**TRUTH TABLE:**

Inputs			Outputs	
A	B	Borrow <sub>in</sub>	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

$$\text{Diff} = (A \oplus B) \oplus \text{'Borrow}_{in}$$

$$\text{Borrow} = A'.B + (A \oplus B)'$$

**OUTPUT**



**RESULT:** Thus full subtractor has been designed and implemented successfully using logisim simulator.