Index

Completed

EXP NO: 01
DATE: 9/01/23

8-BIT ADDITION.

AIM:- To perform the 8-bit addition for the given two numbers.

Procedure:

* Start the program by loading the first data into accumulator.
* Move the data to a register
* Get the second data and load it into the accumulator
* Add the two registers contents
* Check for "carry"
* Store the value of the sum and "carry" in the memory location.

Mnemonics:-

```
LDA 3052
MOV B,A
LDA 3054
ADD B
STA 3056
HLT.
```

Output:-

Address	Data
3052	10
3054	15
3056	25

Result:

Thus, the program for performing 8-bit addition was executed and it is verified.

Exp No: 02
DATE: 9/01/23

8-BIT SUBTRACTION

AIM: To perform the 8-bit subtraction for the given two numbers.

Procedure:

* Start the program by loading the first data into accumulator.
* Move the data to a register (B)
* Get the first data and load into accumulator.
* Get the second data and now subtract the two register contents.
* Check the "carry".
* If the carry is present take 2's complement of Accumulator.
* Store the value of ~~borrow~~ in the memory location.
* Store the different value to a memory
* Location and Terminate the program for the 8-bit subtraction.

Mnemonics:

```
LDA 3052
MOV B,A
LDA 3054
SUB B
STA 3056
HLT.
```

Output:

Addres	Data.
3052	4
3054	15
3056	11

Result:

Thus, the program for performing 8 bit subtracti
was executed and it is verified.

EXP NO:03
DATE:9/01/23

16-BIT ADDITION

AIM: To perform the 16 bit addition by using the GNUSIM 8085.

Procedure:

Load the lower part of the first number in the B registers.

Load the lower part of the second number in A (Accumulator)

Add both the number and store.

Load the higher path of the first number in B. register.

Load the higher path of the second number in A

Add the both the numbers with carrying from the lower bytes (if any) and store them at the location.

Mnemonics:

```
LHLD 2500
XCHG
```

```
LHLD 2504
MOV A,E
ADD L
MOV L,A
MOV A,D
ADC H
MOV H,A
SHLD 2508
HLT.
```

Output:

Address	Data
2500	3
2501	4
2504	5
2505	1
2508	8
2509	5.

Result:

Thus, the program for executing 16 bit addition was executed and output is verified.

EXP NO:04
DATE:9/01/23

16-BIT SUBTRACTION

AIM: To perform the 16 bit subtraction for the given two numbers.

Procedure:

* Get the LSB in L registers and MSB in H register
* Exchange the content of HL register with DE register
* Again get the LSB in L register and MSB in H register of 16 bit number
* Subtract the content of L register from content of E register
* Subtract the content of H register from the content of D register and Borrow from previous step
* Store the result in memory Location.

Mnemonics:

```
LHLD 2500
XCHG
LHLD 2502
MOV A,E
SUB L
```

```
MOV L,A
MOV A,D
ADC H
MOV H,A
SHLD 2504
HLT.
```

Output:

Address	Data
2500	6
2501	0
2502	3
2503	0
2504	3
2505.	0

Result:

Thus the program for performing 16-bit subtraction was executed and it is verified.

EXP NO: 05
DATE: 9/01/23

8-BIT MULTIPLICATION.

Aim: To perform the multiplication of two 8-bit numbers.

Procedure:

* Start the program by loading the first data into Accumulator.
* Move the data to the registers (B)
* Get the second data & Load into accumulator
* Move the data to the register (C)
* Initialize A to 00
* Add the content of A register to register B.
* Check for "Carry"
* Decrement the value of carry
* Check whether related addition is over & store product.
* Terminate the products.

Mnemonics:

```
LDA 4051
MOV 4052 B,A.
MOV LDA 4052
MOV C,A
MVI A,00H
LOOP: ADD B
DCR C
JNZ LOOP
STA 4053
HLT.
```

Output:

Adress	Data
4051	2
4052	3
4053	6

Result:

Thus the program has been executed and output is verified.

EXP NO:06
DATE:10/01/23

8-BIT DIVISION.

Aim: To perform the 8 bit division for the given two numbers using GNUSIM 8085.

Procedure:

* Start the program by loading first data into Accumulator.
* Move the data to the register B.
* Get the second data and load into accumulator.
* Move the data to the register (C).
* Add loop for increament of B.
* Subtract two numbers.
* Check whether repeated subtraction is over and store the value of the product and carry in the memory location.
* Terminate the program.

Mnemonics:-

```
lda 2001
mov b,a.
```

```
Lda 2005
mov c,a
mov a,b
mvi b,00h
loop: inr b
 sub c
 jnz loop
 mov a,b
 sta 2010
 hlt.
```

Output:-

Address	Data
2001	30
2005	5
2010	6.

Result:- Thus the program for 8-bit division data is executed and output is verified.

EXP NO: 07
DATE: 10/01/23

16 BIT MULTIPLICATION

AIM:- To perform the 16-bit multiplication by using GNUSIM-8085

Procedure:

* Load the first data in the HL Pair
* Move content of HL Pair to the stack pointer
* Load second data in the HL Pair and move it to DE
* Make the register as 00H and L register as 00H.
* Make ADD HL Pair and stack pointer
* Check for carrying if carry increment it by 1 else moves to next step.
* Then move E to A and perform or operation with Accumulator and register D.
* If the value of the operation is zero, then store the value else go to step3.

Mnemonics:

```
Lhld 2001
xchg
Lhld 2005
```

```
mov c,h
mvi a,00h
loop: add d
dcr c
jnz loop
mov h,a
mov b,l
mvi a,00h
dcr b
jnz lp
mov L,a
shld 2040
hlt.
```

Output:

Address	Data
2001	3
2002	4
2005	6
2006	2
2010	18
2011	8

Result:

Thus the program for performing 16 bit multiplication was executed and output is verified

EXP NO:08

DATE:10/01/23

16-BIT DIVISION.

Aim: To perform the 16-bit division by using GNUSIM 8085.

Procedure:

* Load the first data in HL Pair
* Move content HL pair to stack pointer
* Load second data in HL Pair move it to DE.
* Make the register as 00H and L register as 00H.
* Make SUB HL Pair and stack pointer.
* Compare two numbers to check the "carry"
* Subtract the two numbers each register by its pair.

  Increment of the carry.
* Check whether repeated subtraction is over for "jnz" is attained.
* Terminate the program.

Mnemonics:

```
Lhld 2001
xchg
Lhld 2005
```

```
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 2010
hlt.
```

Output:

Address	Data
2001	6
2002	15
2005	3
2006	5
2010	2
2011	3

Result: Thus the program for performing 16-bit division is succesfully executed.

EXP NO: 09

DATE: 10/01/23

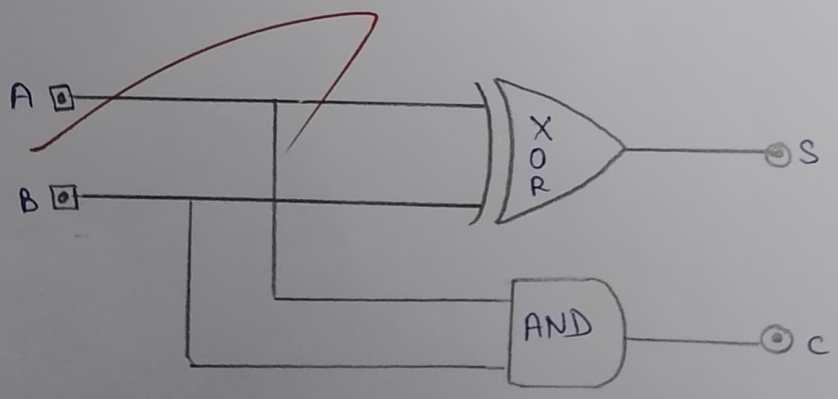
TWO-BIT HALF ADDER.

AIM: To design and impliment the two bit half adder using logism stimulator.

Procedure:

* Insert two inputs in to the canvas
* Label the inputs as A and B by setting the attribute in attribute table
* Insert XOR gate and AND gate from gates table of logisim
* Insert two outputs in canvas
* Label the outputs.

Diagram:

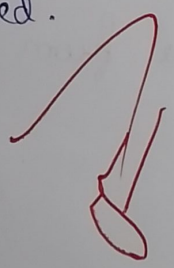


Output:

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

Result:

Half adder is executed and the output is verified.



EXP NO: 10

DATE: 10/01/23

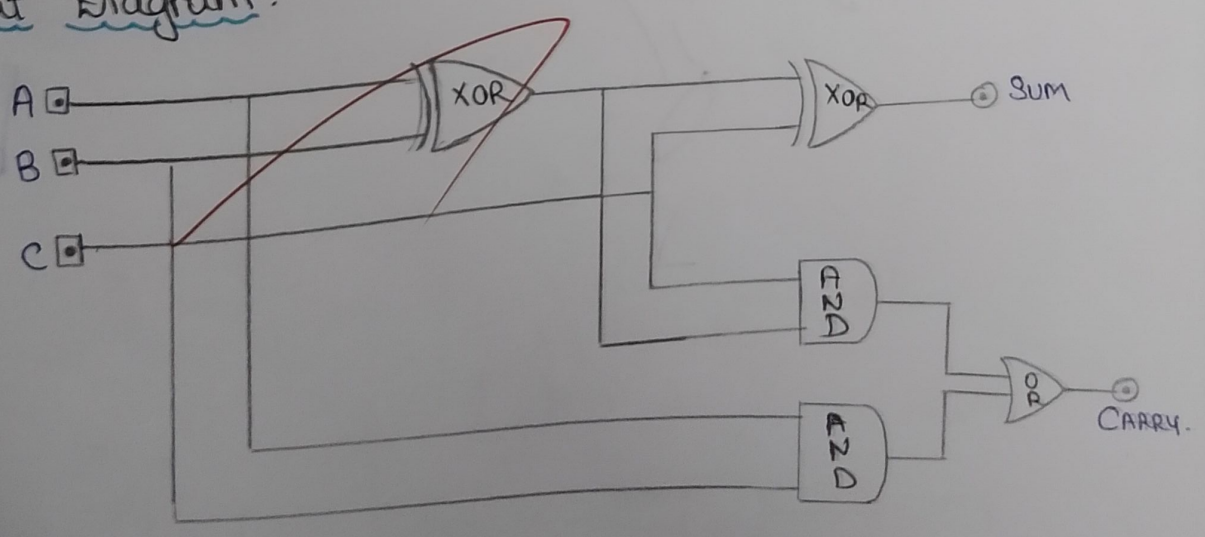
3-BIT FULL ADDER.

AIM: To design and implement 3-bit full adder by Logisim simulator.

Procedure:

* Insert 3 inputs into the canvas
* Label the inputs (A, B & C) by setting the attribute in attribute table.
* Insert two XOR Gates, two AND Gates, one OR Gate from the Gates table of logisim.
* Insert 2 outputs in canvas
* label the outputs.

Input Diagram:



TRUTH TABLE (OUTPUT):

Input			Output	
A	B	C	Sum	Carry.
0	0	0	0	0
0	0	1	01	0
0	1	0	01	0
0	1	1	0	01
1	0	0	01	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Result: "FULL ADDER" is executed and the output is verified.

EXP NO: 11

DATE: 1/10/23

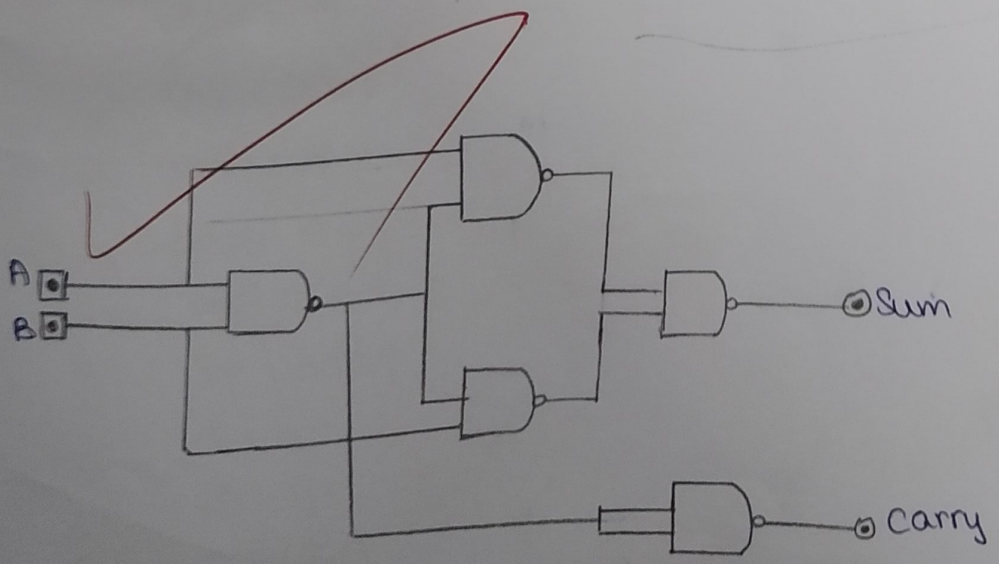
2-BIT HALF ADDER USING NAND GATES.

AIM: To design and implement 2-bit half adder using NAND Gate by Logisim stimulator.

Procedure:

* Insert two inputs A, B.
* Input into the canvas
* Label the inputs (A & B) by setting the attribute table.
* Insert two outputs 0 in canvas
* Insert five NAND gates from gates tab Logisim.
* Label the outputs.

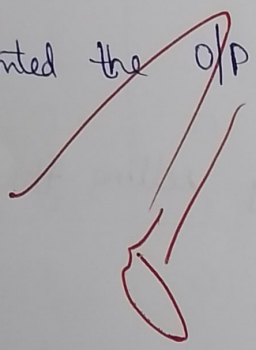
Diagram:



Output:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Result: Thus Half adder using NAND gate is successfully implemented the O/P is also verified.



EXP NO: 12

DATE: 11/01/23

FACTORIAL OF NUMBER

Aim: To perform the factorial of a given number.

Procedure:

* Start program
* Asks the user to enter an integer to find the factorial
* Read the integer and assign it to a variable.
* From the value of int upto 1, multiply each digit and update the final value.
* The final value at the end of the multiplication till 1 is factorial
* End program.

Mnemonics:

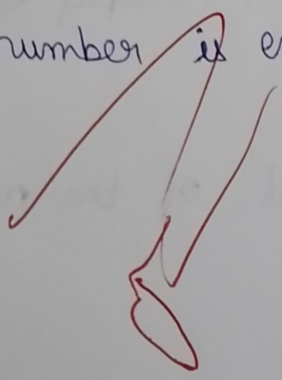
```
Lda 2001
mov b,a
mvi c,#01
mvi a,#01
loop: mov d,c
mvi a,00h
lp: add c
dcr d
jnz lp
mov o,a
```

```
inr c
dcr b
jnz loop
mov a,l
sta 2010
hlt.
```

Output: Enter an integer : 5

factorial of 5 : 120

Result: Thus, the program for performing factorial of a given number is executed and verified.



EXP NO: 13
DATE: 11/01/23

LARGEST NUMBER IN AN ARRAY.

AIM: To find the largest number in an array using GNUSIM 8085.

Procedure:

* Start the program by loading the array size to the pair.
* Copy the array size to B register.
* Increment the memory
* Copy the first data to the accumulator
* Decrement the array size by 1.
* Increment the memory.
* Compare accumulator content & memory
* JUMP on no carry to label AHEAD.
* Copy the memory content to the accumulator.
* Decrement register B by 1.
* Jump on non-zero to label loop
* Store accumulator content to 4300.
* Program ends.

Mnemonics:-

```
LXI H,4
MOV B,M
INX H
MOV A,M
DCR B
INX H
CMP M
JNC AHEAD
MOV A,M
DCR B
JNC Loop
STA 4300
HLT.
```

Output. Input at : 4200: 05 H
4201: 0AH
4202: FH
4203 : 1FH
4204 : 26H
4205: FEH
Output at 4300 : FEH

Result : Thus the program to find the largest number in an array was executed and it is verified.

EXP NO: 14

DATE: 11/01/23

2-STAGE PIPELINING.

Aim :- To implement the 2-stage pipelining using C compiler.

Algorithm:

1. The instruction is fetched from the memory and stored in the instruction register.
2. The instruction is moved to the decoder which decodes the instruction. It activates the appropriate
3. Control signals and takes the necessary steps for the next execution stage.

Program:

```
# 2 stage pipeline  main() {
    Counter=1 ;
    a=int (input ("enter number-1-"));
    Counter = counter+1 ;
    Printf(" 1-Addition, 2-subtraction, 3-multiplication, 4-Division) ;
    Printf ("enter your choice");
    Choice = int (input()) ;
    if choice ==1 :
    printf (" performing addition");
    res=a+b ;
    Counter=counter+1 ;
    if choice ==2 :
```

```
Printf ("performing subtraction")
res = a-b;
counter = counter+1;
if choice==3 ;
Printf ("Performing multiplication");
res=a*b;
Counter = counter+1;
if choice ==4;
if b=0;
Printf ("Denominator can't be zero");
Printf (" Performing division");
res{ = a/b;
counter = counter+1;
if choice ==5: }
Printf ("Correct input_enter"); }
```

Output:- Enter number 1:

Enter number 2:

1- Addition : 2- subtraction : 3- multiplication : 4- Division.

choice : 3- multiplication.

The cycle value is : 4

Enter the number of instructions : 90

The performance measure is : 22

Result: Thus, the program for implementing 2-stage pipelining is executed and output is verified.

EXP NO: 15

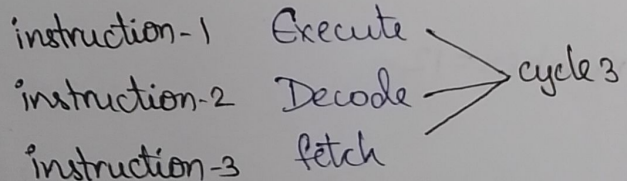
DATE: 11/01/23

3-STAGE PIPELINING.

Aim: To implement 3- stage pipelining

Algorithm:

* **fetch:** instruction is fetched from the memory and stored in the instruction register.
* **Decode:** The instruction is moved to the decoder which decodes the instruction. It activates the appropriate control signals and takes the nessasary steps for the next execution stage.
* **Execute:** The instruction is executed, Data transfer, logical and arithematic operations all take place during this stage.



Program:

```
#include <stdio.h>
int main()
{
long decimalnum, remainder, quotient, octalnum=0;
int octalnum[100], i=1, j;
```

```
Printf("Enter the decimal number:");
Scanf("%ld", &decimalnum);
quotient = decimalnum;
while (quotient != 0)
{
octal num [i++] = quotient % 8;
quotient = quotient 18;
}
for (j = i - 1; j > 0; j--) {
octal num = octal num * 10 + octal num[j];
Printf("Equivalent octal value of decimal num %d is: %d",
                    decimal num, octal num); }
return 0;
}
```

Output:

Enter the decimal number = 12

octal number = 4

Result:

Thus the program for implementing 3 stage pipelining is executed and output is verified.

EXP NO: 16

DATE: 12/01/23

4-STAGE PIPELINING

Aim: To implement 4 stage pipelining using C-program.

Algorithm:

* The instruction is moved to the decoder which decodes the instruction
* The instruction is executed, data tranfers, logical and arithematic operations and takes place during the stage.
* Instruction 2 → Decoder
  Instruction 3 → Decoder
  ] - cycle 4.

Program:

```
#include <stdio.h>
void main()
{
int counter=0;
int input;
int num1, num2, op, res, ins, perf_mis=0;
printf("Enter first value");
scanf("%d", &num1);
Counter += 1;
Scanf ("%d", & num2);
Counter += 1;
```

```
printf("Enter the option");
scanf("%d", &op);
switch [op]
{
case 1:
    printf(" perform addition");
    res = num1 + num2;
    counter += 1
    break;
    ------
    ------
default+;
    ------
    break;
}
```

Output: Enter number 1-4

Enter number 2-4

Enter your choice : 1

performing addition:

cycle value is : 4

enter no of instructions : 3

Performance measure is 0.75

Result:

Thus the program for performing 4-stage pipelining was executed and output was verified.

EXP NO: 17

DATE: 12/01/23

BOOTH ALGORITHM.

AIM: To implement Booth algorithm using C program.

Procedure: [Put multiplicant in BR and multiplier in QR. the algorithm works as per the following conditions]:

* If $2n$ and $2n+1$ are same i.e 00 or 11 perform arithmatic shift by 1 bit
* If $2n$ and $2n+1 = 10$ do $A = A + BR$ and perform arithematic shift by 1 bit.
* If $2n$ and $2n+1 = 01$ do $A = A - BR$ and perform arithematic shift by 1 bit.

Program:

```
#include <stdio.h>
#include <math.h>
int a=0, b=0, c=0, a1=0, b1=0, com[5]={1,0,0,0,0};
int anum[5]={0}, numcp[5]={0}, bnum[5]={0};
int acomp[5]={0}, bcomp[5]={0}, prov[5]={0}, res[5]=0;
void binary(){
a1=fabs(a);
b1=fabs(b);
int r1, r2, i, temp;
```

```
for(i=0; i<8; i++){
r = a1 %2;
a1 = a1/2;
r2 = b1%2
b1 = b1/2;
anum[i] = r;
anumcp[i] = r;
bnum[i] = r2;
    :
    :
}
```

Output: Enter two numbers to multiply:

Booth must be less than 16.

Enter A = 6

Enter B = 12

Expected product = 72

Binary Equivalents are:

A = 00110 ; B = 01100 ; B' + 1 = 10100

AR-SHIFT : 00010 : 01000

Product is = 0001001000

Result: Thus the program for performing Booth algorithm for multiplication was executed and output was verified.

EXP NO: 18
DATE: 12/01/23

C PROGRAM TO IMPLEMENT RESTORING DIVISION.

Aim: To implement integer restoring division using Cprogram.

Program&Algorithm:

*First the registers and initialized with cores ponding values (Q = Divident, M = Divisor, A = 0, n)
*They do left shift for Q & A.
*Then register M is subtracted from A & stored.
*Then it most significant bit of A is 0 then LSB of Q is 1 else LSD of Q is 0.
*The values of counter n is decremented.
*If n=0 we get loop else repeat step.
*finally Q contains quotient and A contain remainders.

Program:

```
# include <stdio.h>
int acum[100] = {0};
void add (int accum[], int b[], int (n));
int q [100], b[100];
int main()
{
int x,y;
```

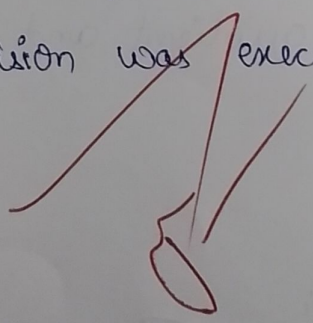
```
printf("enter the number:");
Scanf ("%d %d", &x, &y);
int i=0
while (x>0||y>0)
{
if (x>0)
{
q(i) = x%2
    x = x/2
}
else
{
  q[i]=0
}
}
```

Output:- Enter the number = 8, 3

Quotient = 0010

Remainder = 0010

Result:- Thus, the program for performing integer restoration division was executed and verified.



EXP NO: 19

DATE: 13/01/23

C PROGRAM TO CALCULATE CACHE HIT RATIO.

Aim: To write a C program for calculating cache HIT Ratio

Procedure:

* Start the program...
* Enter the number of cache Hits
* Enter the number of cache misses
* Calculate Cache HIT ratio
* Stop the program.

Program:

```
#include <stdio.h>
int main(){
float cache Hit, Cache miss, Cache Hit ratio;
Printf("\n enter the total no.of cache hits");
Scanf("%f" & Cache Hit);
Printf("\n enter the no.of Cache miss:");
Scanf("%f", & Cache miss);
Cache Hit ratio = Cache hit/(Cache hit + Cache miss)
Printf("\n Cache Hit ratio: %f", Cache hit ratio);
return 0;
}
```

Output:

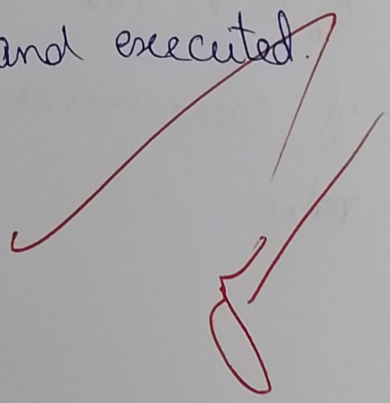
Enter the number of cache hits = 43

Enter the no. of cache miss = 11

Cache Hit ratio : 0.796296.

Result:

Thus, program to calculate cache ~~h~~it ratio is successfully verified and executed.



EXP NO: 20
DATE: 12/01/23

1's and 2's COMPLEMENT OF 8-BIT NUMBER

AIM: To perform 1's and 2's complement of 8-bit number GNUSIM-8085, Microprocessor simulator.

Procedure:

* Start the program by loading the first data into accumulator
* Move the data into register.
* Add or to accumulator content.
* Store content of accumulator in memory.
* Stop.

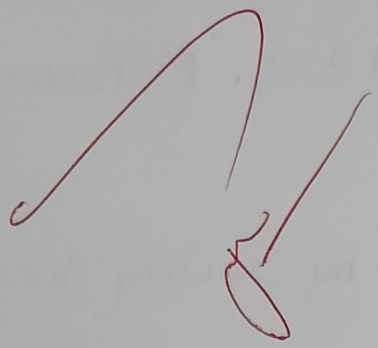
Mnemonics:

```
LDA 3000H
LMA
STA 3001
ADI 01
STA 3002
HLT
```

Output

Address	Data
3000	50
3001	255

Result: Thus, the program for performing 1's & 2's complement for 8 bit was executed and output is verified.



EXP NO:21
DATE:13/01/23

C PROGRAM TO CONVERT DECIMAL TO BINARY.

Aim: To convert a decimal number to Binary number using C.

Procedure:

* Take a decimal number as input
* Divide the number by 2 through %, and store the remainder.
* Divide the number by 2 through '/' division operator
* Repeat the step2 until number is greater than 0.

Program:

```
#include<stdio.h>
int main(){
int a[10],n,i;
System("cls");
Printf("Enter the number to convert:");
Scanf("%d",&n);
for(i=0;n>0;i++)
 {
 a[i]=n%2;
 n=n/2;
}
```

```
printf("\n Binary of given number is:");
for(i=1 ; i>=0; i--)
{
  printf ("%d",&a[i]);
}
return0;
}
```

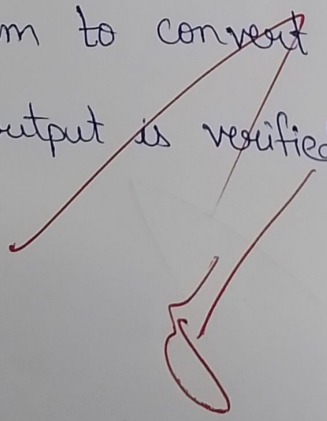
Output :-

enter the number to convert =5

Binary of a given number is=101.

Result :-

Thus, the program to convert decimal to Binary is executed and output is verified.



EXP NO: 22
DATE: 13/01/23

C PROGRAM TO CONVERT DECIMAL TO OCTAL.

Aim: To convert a decimal number to octal number using C.

Procedure:

* Take a decimal number as an input.
* Store the remainder when the number is divided by 8 in an array.
* Repeat the above two steps until the number is not equal to 0.
* Print the array in reverse order.

Program:

```
# include < stdio.h>
int main() {
long decimal num, remainder, quotient, octal num=0;
int octal number [100], i=1, j;
Printf ("Enter the decimal number:");
Scanf ("%d", & decimal num);
quotient = decimal num;
while (quotient != 0) {
octal number [i++] = quotient %8
```

```
quotient = quotient /8;
}
for (j=i-1; j>0; j--)
octal num = octal num* 10 + octal number[j];
printf("Equivalent octal value of decimal number:");
return 0;
}
```

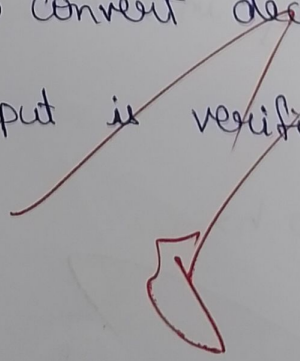
Output:-

Enter a Decimal number: 16

The Decimal number is 16

The octal number is 20

Result:- The program to convert decimal to octal is executed and output is verified.



EXP NO: 23
DATE: 13/01/23

C PROGRAM TO CONVERT BINARY TO DECIMAL.

Aim: To convert a binary number to decimal number using C.

Procedure:

* Take a binary number as the input.
* Divide the number by 10 and store the remainder into variable remainder.
* Decimal_num = decimal_num + remai· base.
* Divide the quotient of the original number by 10.
* Multiply for base by 2 and print the decimal number.

Program:

```
#include <stdio.h>
void main(){
int num, int binary_num, decimal_num=0, base=1, rem;
printf("Enter a binary number\n");
Scanf("%d", &num);
binary_num = num;
while(num>0){
rem = num%.10;
decimal_num = decimal_num + rem + base;
```

```
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);
}
```

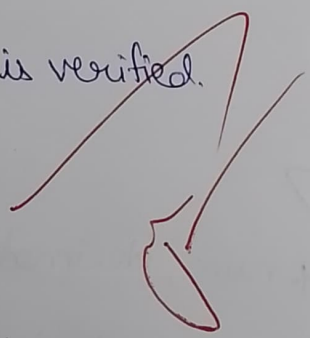
Output:

Enter a binary number = 1011

Decimal number of binary is 11

Result:-

Thus, the program to convert binary to decimal is executed and output is verified.



EXP NO: 24

DATE: 13/01/23

CPU PERFORMANCE.

AIM: To find the CPU performance of a processor.

Algorithm:

1. Start the program and load all the variables.
2. Initialize an array to get no. of processors, cycles, per instruction and clock rate.
3. Initialize a formula to calculate the cpu time
4. Initialize another array to get processor with lowest execution time.
5. End the program and load the results.

Program:-

```
# include <stdio.h>
int main() {
float cr;
int P1, P, i;
float CPU[5];
float cpi, ct, max;
int n=1000;
for(i=0; i<=4; i++)
CPU[5]=0;
```

```
printf(" enter the no of processors");
scanf("%d",&p);
p1=p;
for(i=0; i<p; i++)
{
printf(" enter the cycles per instruction of processor:");
scanf("%f" &cpi);
printf(" Enter the clockrate in GHz");
scanf("%f",& cr);
ct=1000*cpi/cr;
printf(" the cpu time : %f", ct);
cpu[i]=ct;
}
max=cpu[0]
for(i=0; i<p; i++){
if(cpu[i] <=max)
 max=cpu[i]; }
printf("The processor has lowest execution time :%f", max);
return 0; }
```

Output: Enter no.of processors: 1
Enter the cycles per instruction: 2
Enter the clock rate in GHz: 1
The cpu time is: 2000

Result: Thus, the program to find CPU performance of processor was executed and output was verified.

Exp No: 25
DATE: 13/01/23

SWAP TWO 8-BIT DATA

Aim: To perform swapping of two 8- bit data

Procedure:

* Load a 8-bit value from memory 2500 into accumulator
* Move value of accumulator into register H.
* Load the 8-bit number from memory 2501 into the accumulator
* Move the value of accumulator into register D.
* Exchange both the registers pairs
* "Stop"

Mnemonics:

```
LDA 2500
MOV B,A
LDA 2501
STA 2500
MOV A,B
STA 2501
HLT.
```

Output:

Address	Data
2501	31
2500	12

→

Address	Data
2501	12
2500	31

Result :

Thus the program to swap two 8-bit data is executed and the output is verified