CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY [CHARUSAT]

CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY [CSPIT]

U & P U. PATEL DEPARTMENT OF COMPUTER ENGINEERING

CE258: MICROPROCESSOR AND COMPUTER ORGANIZATION

Practical File

Submitted by: Vatsal Doshi (20CE027)

Practical:2

AIM:

Write a program to convert a given number system to other number system. Task: Ask user to choose one form given number system :Binary, Octal, Decimal, Hexadecimal. Ask user to enter a number (number can be float). Check that number entered for given number system is correct. If not give the error If correct then convert entered number in rest of the three number system. If you can show the conversion then its well and good.

CODE :

```
#include <stdio.h>

#include <math.h>

#include <string.h>


int Binary_to_Decimal(long int);

long int Binary_to_Octal(long int);

long int Binary_to_Hexadecimal(long int);


long int Decimal_to_Binary(long int);

long int Decimal_to_Octal(long int);

long int Decimal_to_Hexadecimal(long int);


long int Octal_to_Binary(long int);

long int Octal_to_Decimal(long int);

long int Octal_to_Hexadecimal(long int);


void Hexadecimal_to_Binary(char []);

void Hexadecimal_to_Decimal(char []);
```

```
void Hexadecimal_to_Octal(char []);



int main() {


int operations,numbers=1,checking;
long int b,o,d;
char h[100];
int x,y,spacebar;


int num;
A:
printf("1.Binary\n2.Decimal\n3.Octal\n4.Hexa-Decimal\n");
printf("Enter Choice: ");
scanf("%d",&num);
if(num<=4)
{
switch(num)
{
case 1:
D:
printf("\nEnter the Number in Binary: ");
scanf("%ld",&b);




checking=b;
```

```
while(checking!=0)
{
numbers=checking%10;
if(numbers>1)
{
printf("\n\n%d IS NOT BINARY NUMBER.",b);
goto D;
}
else
checking=checking/10;
}
Binary_to_Decimal(b);
Binary_to_Octal(b);
Binary_to_Hexadecimal(b);
break;



case 2:

printf("\nEnter the Number in Decimal form (0 to 9): ");
scanf("%ld",&d);
Decimal_to_Binary(d);
Decimal_to_Octal(d);
```

```
Decimal_to_Hexadecimal(d);
break;




case 3:


B:
printf("\nEnter the Number in Octal form (0 to 7): ");
scanf("%ld",&o);
// CHECKING INPUT IS IN OCTAL FORM
checking=o;

while(checking!=0)
{
numbers=checking%10;
if(numbers>7)
{
printf("\n%d IS NOT OCTAL NUMBER.\n",numbers);
goto B;
}
else
{
checking=checking/10;
x++;
}
}
```

```
Octal_to_Binary(o);
Octal_to_Decimal(o);
Octal_to_Hexadecimal(o);
break;




case 4:
X:
printf("\nEnter the Number in Hexa-Decimal form: ");
scanf("%s",&h);
 //check
for(x=strlen(h)-1;x>=0;x--)
{
if(h[x]>'f' && h[x]<='z' || h[x]>'F'&& h[x]<='Z')
{
printf("\nYou have to Enter Hexa-Decimal Number.\n");
printf("'%c' IS NOT Hexa-Decimal Number.\n",h[x]);
goto X;
}
}
Hexadecimal_to_Binary(h);
Hexadecimal_to_Decimal(h);
Hexadecimal_to_Octal(h);
break;
}
}
```

```
else
{
printf("***Enter Valid Choice*** \n");
goto A;
}
return 0;
}
int Binary_to_Decimal(long int bin)
{
int remainder,summation=0,x=0;
while(bin!=0)
{
remainder=bin%10;
bin=bin/10;
summation=summation+remainder*pow(2,x);
x++;
}
printf("\nDecimal Number : %d",summation);
}
long int Binary_to_Octal(long int bin)
{
   int x=0,remainder,sum=0,remaining[100],length=0;

   while(bin!=0)
   {
      remainder=bin%10;
      bin=bin/10;
```

```
        sum=sum+remainder*pow(2,x);
        x++;
    }
    x=0;
    while(sum!=0)
    {
        remaining[x]=sum%8;
        sum=sum/8;
        x++;
        length++;
    }
    printf("\nOctal Number : ");
    for(x=length-1;x>=0;x--)
    {
        printf("%d",remaining[x]);
    }
}

long int Binary_to_Hexadecimal(long int bin)
{
    int remainder,x=0,summation=0,remaining[100],length=0;

    while(bin!=0)
    {
        remainder=bin%10;
        bin=bin/10;
        summation=summation+remainder*pow(2,x);
```

```
        x++;
    }
    x=0;
    while(summation!=0)
    {
        remaining[x]=summation%16;
        summation=summation/16;
        x++;
        length++;
    }
    printf("\nHexa-Decimal Number : ");
    for(x=length-1;x>=0;x--)
    {
        switch(remaining[x])
        {
            case 10:
                printf("A"); break;

            case 11:
                printf("B"); break;

            case 12:
                printf("C"); break;

            case 13:
                printf("D"); break;
```

```
        case 14:
            printf("E"); break;

        case 15:
            printf("F"); break;

        default:
            printf("%d",remaining[x]);
      }

   }
}
long int Decimal_to_Binary(long int dec)
{
   int remainder[50],x,length=0;
   do
   {
      remainder[x]=dec%2;
      dec=dec/2;
      x++;
      length++;
   }
   while(dec!=0);

   printf("\nBinary Number : ");
   for(x=length-1;x>=0;x--)
   {
```

```
        printf("%d",remainder[x]);
    }
}

long int Decimal_to_Octal(long int dec)
{
    int remainder[50],x,length=0;
    do
    {
        remainder[x]=dec%8;
        dec=dec/8;
        x++;
        length++;
    }
    while(dec!=0);

    printf("\nOctal Number : ");
    for(x=length-1;x>=0;x--)
    {
        printf("%d",remainder[x]);
    }
}

long int Decimal_to_Hexadecimal(long int dec)
{
    int remainder[50],x,length=0;
    do
```

```
    {
        remainder[x]=dec%16;
        dec=dec/16;
        x++;
        length++;
    }
    while(dec!=0);

    printf("\nHexa-Decimal Number : ");
    for(x=length-1;x>=0;x--)
    {
        switch(remainder[x])
        {
            case 10:
                printf("A"); break;

            case 11:
                printf("B"); break;

            case 12:
                printf("C"); break;

            case 13:
                printf("D"); break;

            case 14:
                printf("E"); break;
```

```
        case 15:
            printf("F"); break;

        default:
            printf("%d",remainder[x]);
    }

  }
}
long int Octal_to_Binary(long int oct)
{
  int remainder[50],length=0,decimals=0,x=0,numbers,answers;

  while(oct!=0)
  {
    answers=oct % 10;
    decimals = decimals + answers * pow(8,x);
    x++;
    oct = oct/10;
  }

  x=0;
  do
  {
    remainder[x]=decimals%2;
    decimals=decimals/2;
```

```
        x++;
        length++;
    }
    while(decimals!=0);

    printf("\nBinary Number : ");
    for(x=length-1;x>=0;x--)
    {
        printf("%d",remainder[x]);
    }
}

long int Octal_to_Decimal(long int oct)
{
    int decimals=0,x=0,numbers,answers;

    while(oct!=0)
    {
        answers=oct % 10;
        decimals = decimals + answers * pow(8,x);
        x++;
        oct = oct/10;
    }
    printf("\nDecimal Number : %d",decimals);
}

long int Octal_to_Hexadecimal(long int oct)
```

```
{
   int remainder[50],length=0,decimal=0,x=0,numbers,answers=0;
   while(oct!=0)
   {
      answers=oct % 10;
      decimal = decimal + answers * pow(8,x);
      x++;
      oct = oct/10;
   }
   x=0;
   while(decimal!=0)
   {
      remainder[x]=decimal%16;
      decimal=decimal/16;
      x++;
      length++;
   }
   printf("\nHexa-Decimal Number : ");
   for(x=length-1;x>=0;x--)
   {
      switch(remainder[x])
      {
         case 10:
            printf("A"); break;

         case 11:
            printf("B"); break;
```

```
        case 12:
            printf("C"); break;

        case 13:
            printf("D"); break;

        case 14:
            printf("E"); break;

        case 15:
            printf("F"); break;

        default:
            printf("%d",remainder[x]);
    }

  }
}
void Hexadecimal_to_Binary(char hex[])
{
  int x=0;
  printf("\nBinary Number : ");
  for(x=0;x<strlen(hex);x++)
  {
    switch (hex[x])
    {
```

```
    case '0':
        printf("0000"); break;
    case '1':
        printf("0001"); break;
    case '2':
        printf("0010"); break;
    case '3':
        printf("0011"); break;
    case '4':
        printf("0100"); break;
    case '5':
        printf("0101"); break;
    case '6':
        printf("0110"); break;
    case '7':
        printf("0111"); break;
    case '8':
        printf("1000"); break;
    case '9':
        printf("1001"); break;
    case 'A':
    case 'a':
        printf("1010"); break;
    case 'B':
    case 'b':
        printf("1011"); break;
    case 'C':
```

```
        case 'c':
            printf("1100"); break;
        case 'D':
        case 'd':
            printf("1101"); break;
        case 'E':
        case 'e':
            printf("1110"); break;
        case 'F':
        case 'f':
            printf("1111"); break;

        default:
            printf("\n Invalid hexa digit %c ", hex[x]);
        }
    }

}

void Hexadecimal_to_Decimal(char hex[])
{
    int x,numbers=0,powered=0,decimal=0;

    for(x=strlen(hex)-1;x>=0;x--)
    {
        if(hex[x]=='A'||hex[x]=='a')
        {
```

```
        numbers=10;
    }
    else if(hex[x]=='B'||hex[x]=='b')
    {
        numbers=11;
    }
    else if(hex[x]=='C'||hex[x]=='c')
    {
        numbers=12;
    }
    else if(hex[x]=='D'||hex[x]=='d')
    {
        numbers=13;
    }
    else if(hex[x]=='E'||hex[x]=='e')
    {
        numbers=14;
    }
    else if(hex[x]=='F'||hex[x]=='f')
    {
        numbers=15;
    }
    else
    //(a[i]>=0 || a[i]<=9)
    {
        numbers=hex[x]-48;
    }
```

```
        decimal=decimal+numbers*pow(16,powered);
        powered++;
    }
    printf("\nDecimal Number : %d",decimal);

}

void Hexadecimal_to_Octal(char hex[])
{
    int x,length,numbers=0,powered=0,decimal=0,remainder[100];

    for(x=strlen(hex)-1;x>=0;x--)
    {
        if(hex[x]=='A'||hex[x]=='a')
        {
            numbers=10;
        }
        else if(hex[x]=='B'||hex[x]=='b')
        {
            numbers=11;
        }
        else if(hex[x]=='C'||hex[x]=='c')
        {
            numbers=12;
        }
        else if(hex[x]=='D'||hex[x]=='d')
```

```
        {
            numbers=13;
        }
        else if(hex[x]=='E'||hex[x]=='e')
        {
            numbers=14;
        }
        else if(hex[x]=='F'||hex[x]=='f')
        {
            numbers=15;
        }
        else
        //(a[i]>=0 || a[i]<=9)
        {
            numbers=hex[x]-48;
        }

        decimal=decimal+numbers*pow(16,powered);
        powered++;
    }

    x=0,length=0;
    while(decimal!=0)
    {
        remainder[x]=decimal%8;
        decimal=decimal/8;
        x++;
```

```
        length++;
    }
    printf("\nOctal Number : ");
    for(x=length-1;x>=0;x--)
    {
        printf("%d",remainder[x]);
    }
printf("D21CE172 - CHETNA MAKWANA");
}
```

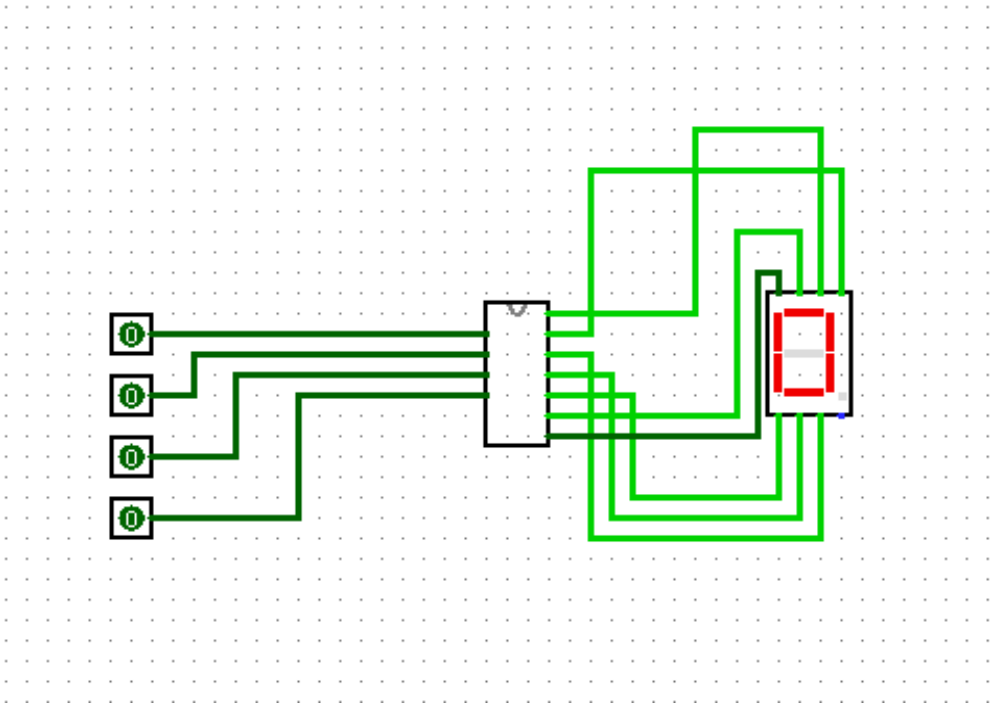
OUTPUT :

```
1.Binary
2.Decimal
3.Octal
4.Hexa-Decimal
Enter Choice: 1

Enter the Number in Binary: 111111

Decimal Number : 63
Octal Number : 77
Hexa-Decimal Number : 3F
```

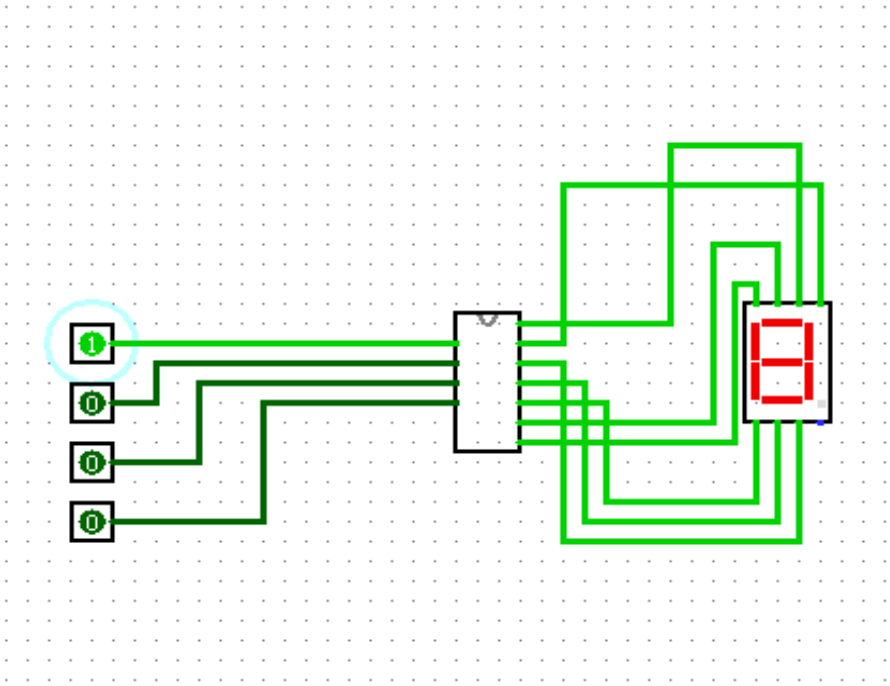
Practical : 3

AIM :

Implement a circuit in Logisim to display given binary number in decimal on to seven segment display.

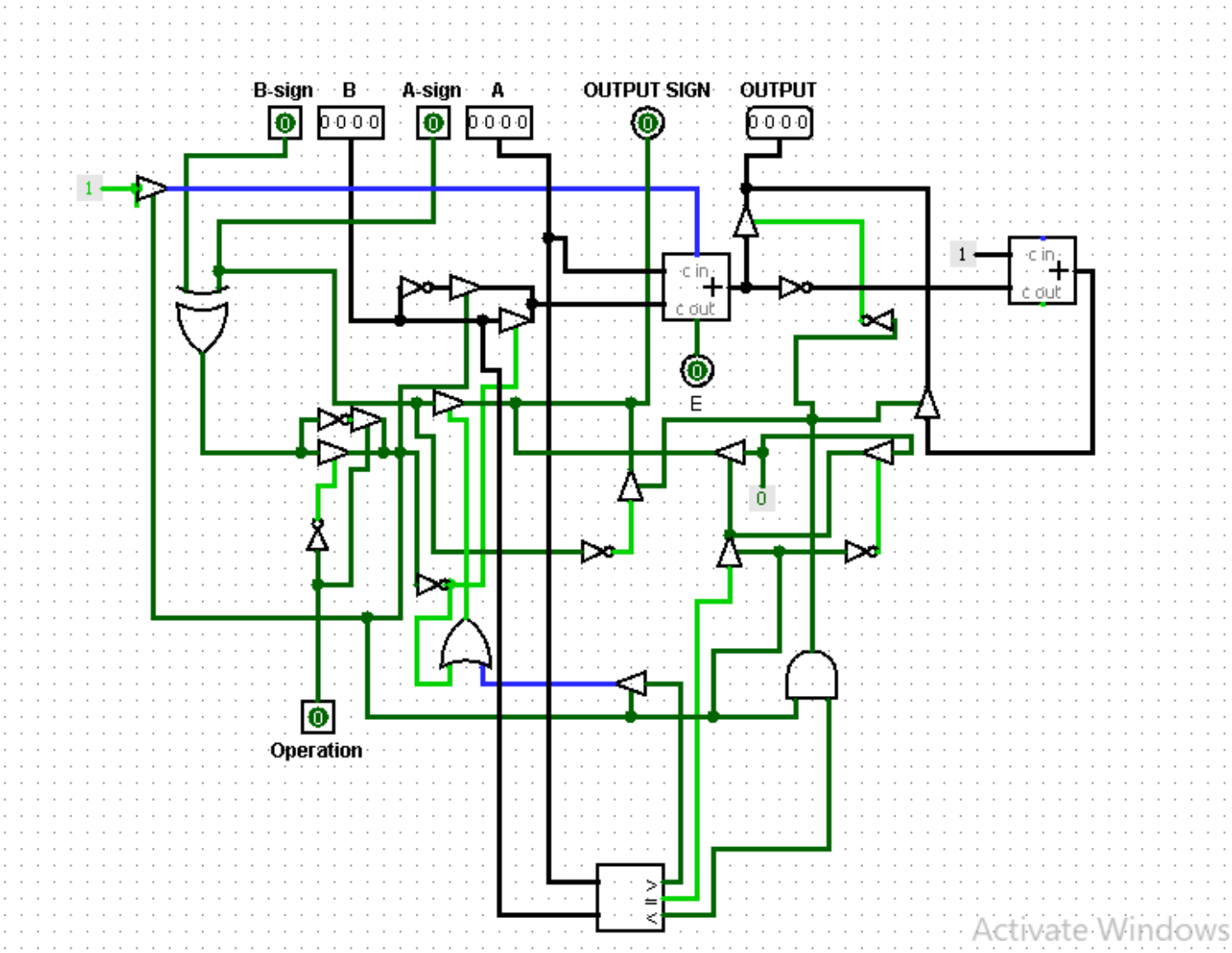


Output :

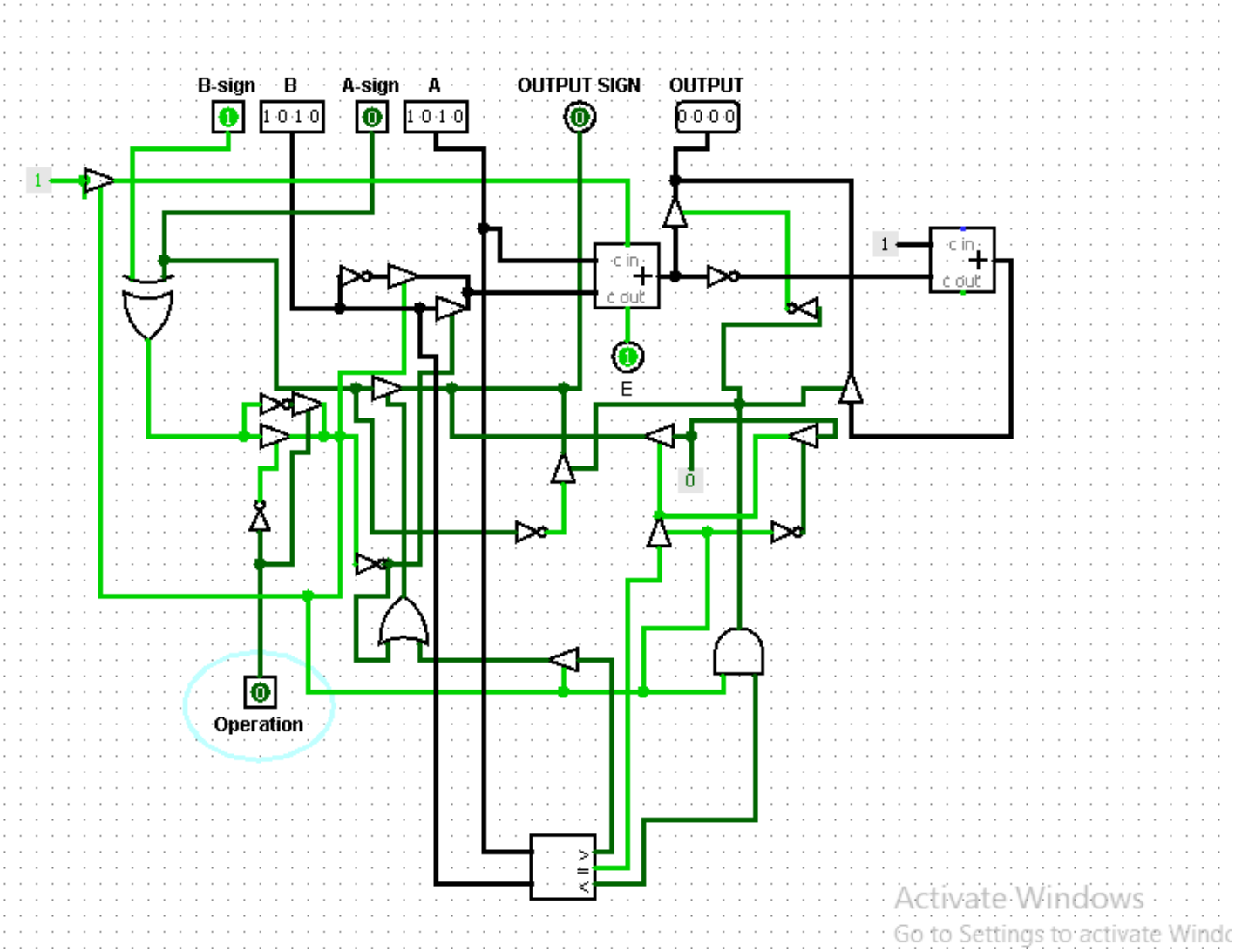

1
0
0
0

Practical : 4

AIM : Implement a circuit in Logisim which perform Addition and Subtraction of sign number.



OUTPUT:


B-sign
B
A-sign
A
OUTPUT SIGN
OUTPUT
1 0 1 0
1 0 1 0
0 0 0 0
1
c in
c out
E
0
Operation
Activate Windows
Go to Settings to activate Windo

PRACTICAL : 5

AIM : Write a program which perform multiplication using booth algorithm.

CODE:

```
#include <bits/stdc++.h>
using namespace std;

class BoothMul
{
   int multiplicand[20], multiplier[20], shift[20], arr[20],
par_pro[20], temp[20], i, j, sq = 0, len_a, len_b, q_n1, q_n,
carry;

public:
   BoothMul(string a, string b) // class constructor
   {
      len_a = a.length(); // length of multiplicand
      len_b = b.length(); // length of multiplier
      sq = len_b;        // storing the sequence counter

      for (i = 0; i < len_a; i++)
      {
```

```
        multiplicand[i] = arr[i] = int(a.at(i)) - 48; // storing the
value entered by user into multiplicand array
    }

    for (i = 0; i < len_b; i++)
    {
        multiplier[i] = int(b.at(i)) - 48; // storing the value
entered by user into multiplier array
    }

    // doing 2's complement of multiplicand and storing in
arr[] array
    for (j = 0; j < len_a; j++)
    {
        if (arr[j] == 0)
        {
            arr[j] = 1;
        }
        else
        {
            arr[j] = 0;
```

```
        }
    }

    for (j = len_a - 1; j > 0; j--)
    {
        if (arr[j] == 0)
        {
            arr[j] = 1;
            break;
        }
        else
        {
            arr[j] = 0;
        }
    }
}

void logic() // function to implement the main logic of Booth Multiplication
{
    for (i = 0; i < len_a; i++)
```

```
        {
            par_pro[i] = 0; // setting the partial product at initial to
all 0's
        }

        i = len_b - 1;
        while (sq != 0)
        {
            if (i == len_b - 1)
            {
                q_n1 = 0;
                q_n = multiplier[i];
            }
            else
            {
                q_n1 = multiplier[i + 1];
                q_n = multiplier[i];
            }

            if (q_n1 == q_n)
            {
```

```
            //  direct right shift operation
        }
        else if (q_n == 1 && q_n1 == 0)
        {
            ;
            j = len_a - 1;
            carry = 0;
            while (j >= 0)
            {
                temp[j] = par_pro[j];                    // storing
partial product in a temporary array to count carry
                par_pro[j] = (par_pro[j] + arr[j] + carry) % 2; //
subtraction operation on partial product
                carry = (temp[j] + arr[j] + carry) / 2;        //
counting the carry for next stage
                j--;
            }
        }
        else if (q_n == 0 && q_n1 == 1)
        {
            j = len_a - 1;
```

```
            carry = 0;
            while (j >= 0)
            {
                temp[j] = par_pro[j];                          // storing
partial product in a temporary array to count carry
                par_pro[j] = (par_pro[j] + multiplicand[j] + carry)
% 2; // addition operation on partial product
                carry = (temp[j] + multiplicand[j] + carry) / 2;
// counting the carry for next stage
                j--;
            }
        }

        for (j = len_a - 1; j > 0; j--)
        {
            if (j == len_a - 1)
            {
                shift[i] = par_pro[j]; // storing the LSB of partial
product into the shift array
            }
            par_pro[j] = par_pro[j - 1]; // performing the
arithmetic right shift operation
```

```
        }

        i--;  // decreamenting the index counter
        sq--; // decreamenting the sequence counter
    }
  }

  void print() // function to print multiplicated final product
  {
    cout << "\nMultiplied Binary Number : ";
    for (i = 0; i < len_a; i++)
    {
        cout << par_pro[i];
    }

    for (i = 0; i < len_b; i++)
    {
        cout << shift[i];
    }
  }
};
```

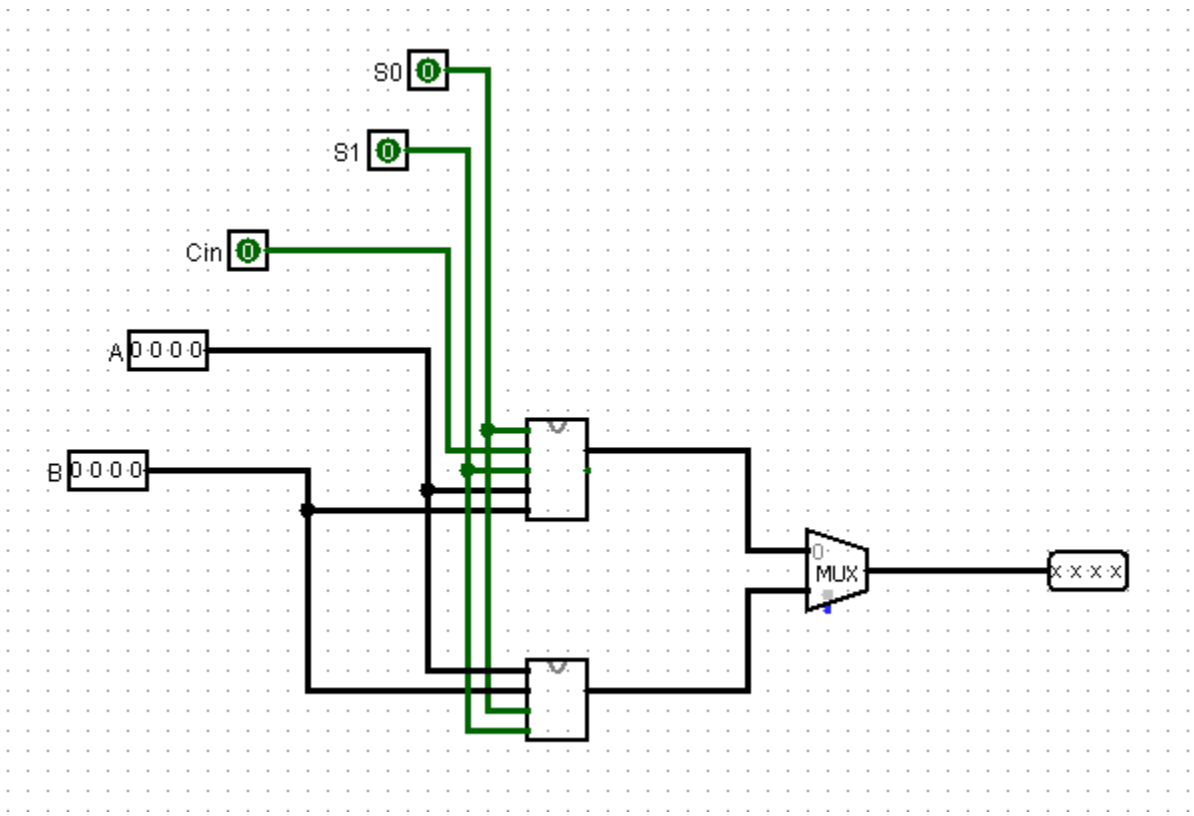
```
int main()
{
   string a, b;
   cout << "Please Enter two Binary Numbers(in 2's complement form) to Multiply :" << endl;
   cout << "Multiplicand : ";
   cin >> a;
   cout << "Multiplier : ";
   cin >> b;
   BoothMul obj(a, b); // calling class constructor
   obj.logic();
   obj.print();
   return 0;
}
```

OUTPUT :

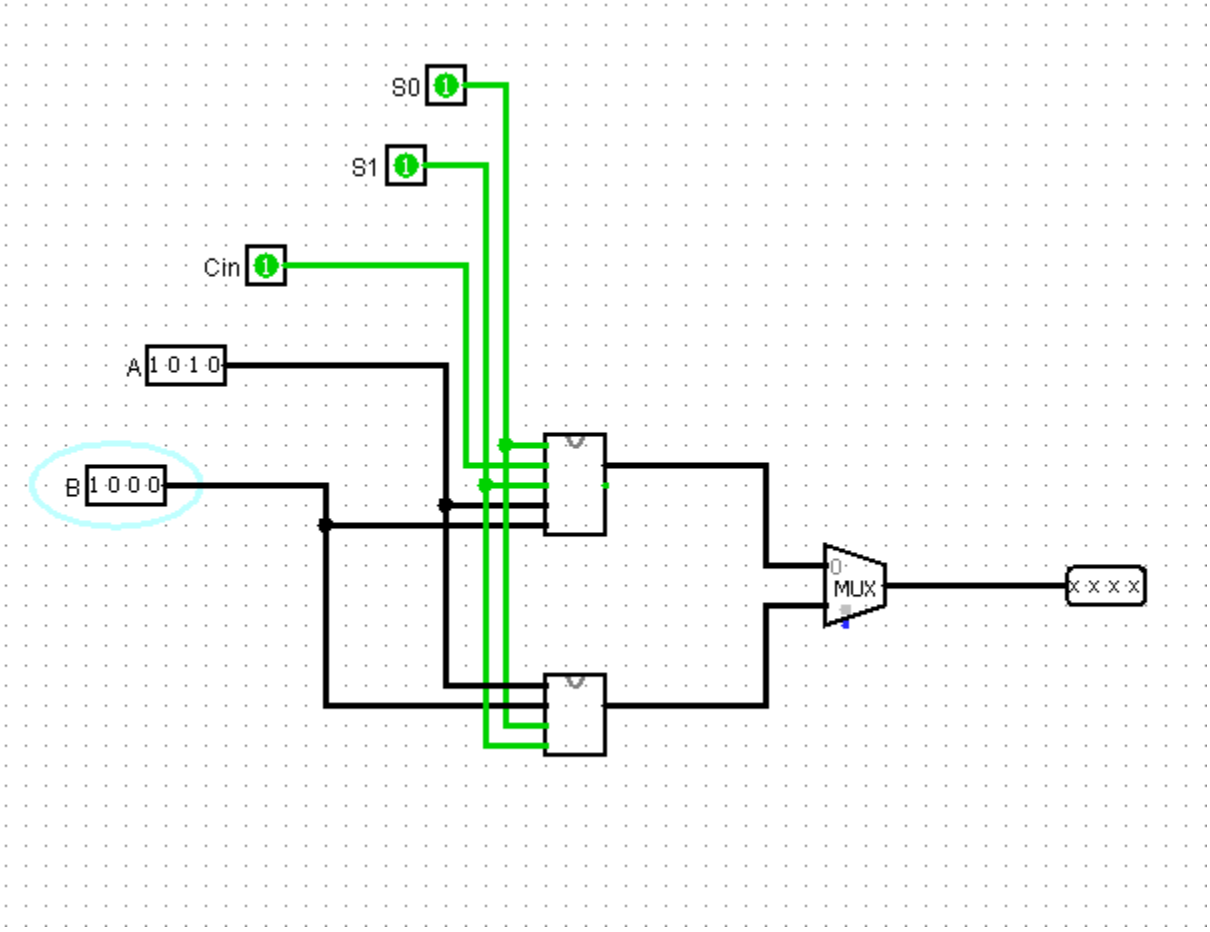
```
Please Enter two Binary Numbers(in 2's complement form) to Multiply :
Multiplicand : 10
Multiplier : 11

Multiplied Binary Number : 0000

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL : 6

AIM: Implement a circuit in Logisim which perform Arithmetic and Logic unit.



Output:


S0
S1
Cin
A
1 0 1 0
B
1 0 0 0
MUX
x x x x

PRACTICAL: 7

AIM: **Complete the following task with the help of Emulator.**

Perform following

(1) addition and subtract of two 64-bits data available at offset 20 and 30 and save result at 40 and 50 offset respectively.

```
01 org 100h
02 mov [0020],123h
03 mov [0022],5678h
04 mov [0024],9876h
05 mov [0026],7777h
06 mov [0030],1122h
07 mov [0032],3455h
08 mov [0034],1515h
09 mov [0036],7575h
10 mov ax,[0020]
11 mov bx,[0030]
12 add ax,bx
13 mov [0040],ax
14 mov ax,[0022]
15 mov bx,[0032]
16 adc ax,bx
17 mov [0042],ax
18 mov ax,[0024]
19 mov bx,[0034]
20 adc ax,bx
21 mov [0044],ax
22 mov ax,[0026]
23 mov bx,[0036]
24 adc ax,bx
25 mov [0046],ax
26 mov ax,[0040]
27 mov bx,[0042]
28 mov cx,[0044]
29 mov dx,[0046]
30 ret
```

```
org 100h
mov [0020],123h
mov [0022],5678h
mov [0024],9876h
mov [0026],7777h
mov [0030],1122h
mov [0032],3455h
mov [0034],1515h
mov [0036],7575h
mov ax,[0020]
mov bx,[0030]
sub ax,bx
mov [0040],ax
mov ax,[0022]
mov bx,[0032]
sbb ax,bx
mov [0042],ax
mov ax,[0024]
mov bx,[0034]
sbb ax,bx
mov [0044],ax
mov ax,[0026]
mov bx,[0036]
sbb ax,bx
mov [0046],ax
mov ax,[0040]
mov bx,[0042]
mov cx,[0044]
mov dx,[0046]
ret
```

(2) multiply two 8-bits data available at offset 70 and 74 and save the result at 78 offset

```
org 100h
mov [0070],05h
mov [0074],03h
mov ax,[0070]
mov bx,[0074]
mul bx
mov [0078],ax
ret
```

(3) divide one 16-bits number from offset 100 by 8-bits number from offset 104 and save quotient at offset 106 and remainder at offset 108.

```
mov [0100],0100h
mov [0104],20h
mov ax,[0100]
mov dl,[0104]
div dl
mov [0106],al
mov [0108],ah
ret
```

(4) and, or , xor of two 16-bits data available at offset 150 and 152 and store respective results at 155, 160 and 165 offsets.

```
mov [0150],1010h
mov [0152],0010h
mov ax,[0150]
and ax,[0152]
mov bx,[0150]
or bx,[0152]
mov cx,[0150]
xor cx,[0152]
mov [0155],ax
mov [0160],bx
mov [0165],cx
ret
```

(5) arithmetic and logical shift of 16-bits data available at offset 170 and store respective results at offsets 172, 174, 176, 178.

emu8086 - assembler and microprocessor emulator 4.08

file edit bookmarks assembler help

NEW OPEN SAVE ASSEMBLE RUN

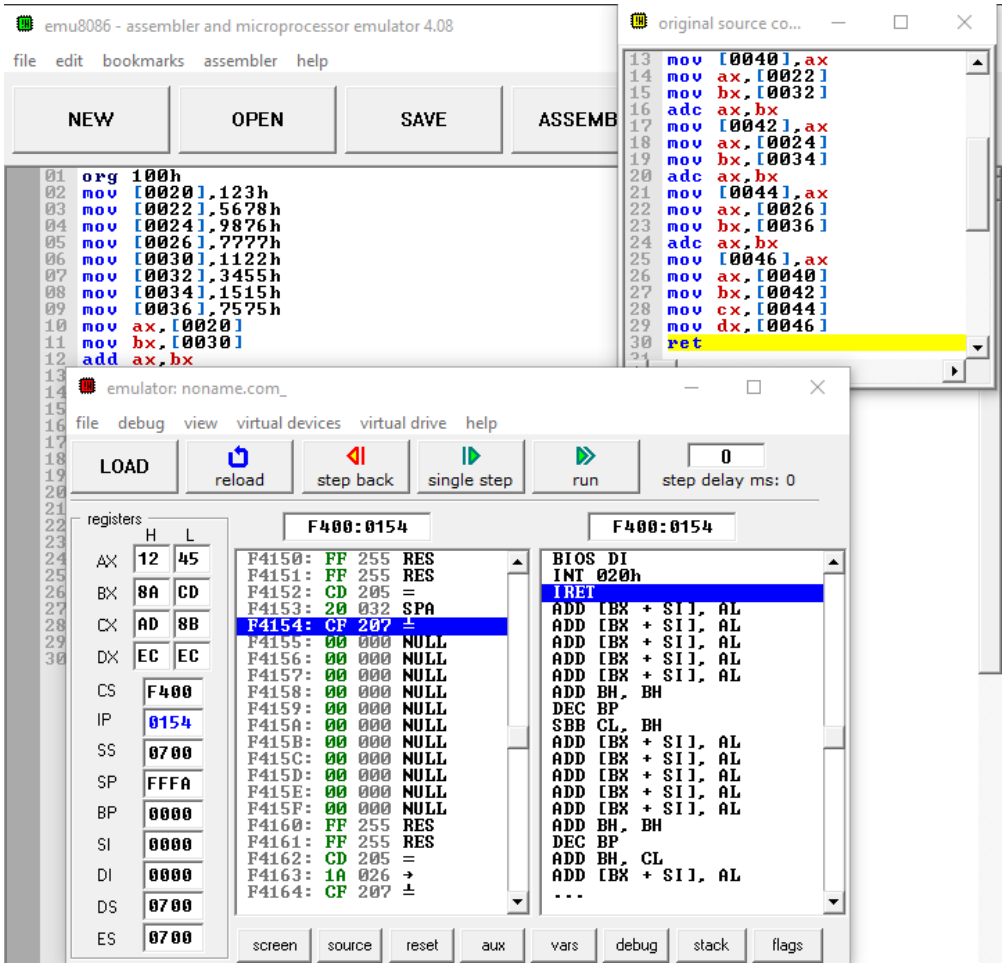
```
.model shift_bits
.data
sr dw 1 dup(0)
ar dw 1 dup(0)
sl dw 1 dup(0)
al_ dw 1 dup(0)
to_s dw 1 dup(0d441h)
.code
mov cx,@data
mov ds,cx
;2 is the number of bits shifted
;shift right
mov ax,to_s
shr ax,2
mov [0172],ax
mov sr,ax
;arithmetic shift right
mov ax,to_S
sar ax,2
mov [0174],ax
mov ar,ax
;shift left
mov ax,to_s
shl ax,2;
mov [0176],ax
mov sl,ax
;shift arithmetic left
mov ax,to_s
sal ax,2
mov [0178],ax
mov al_,ax
int 20h
```

(6) rotate left and right with and without carry of 1 and 3 bit of 8-bits data available at offset 200 and store respective result at offsets 201, 202, 203, 204, 205, 206, 207, 208.

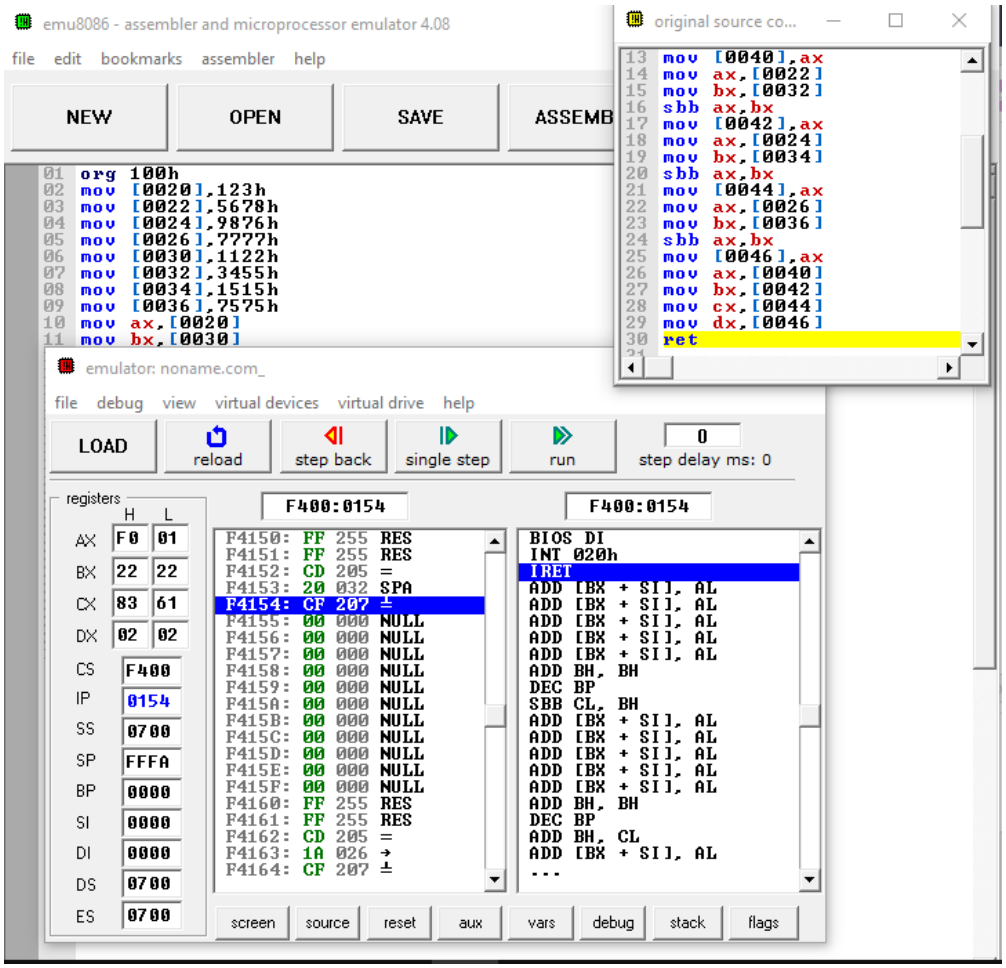
```
mov [0001],28h
mov ah,[0001]
rol ah,1
mov [0003],ah
mov ah,[0001]
rol ah,3
mov [0005],ah

mov ah,[0001]
ror ah,1
mov [0007],ah
mov ah,[0001]
ror ah,3
mov [0009],ah


mov ah,[0001]
rcl ah,1
mov [0011],ah
mov ah,[0001]
rcl ah,3
mov [0013],ah

mov ah,[0001]
rcr ah,1
mov [0015],ah
mov ah,[0001]
rcr ah,3
mov [0017],ah
```

Output:

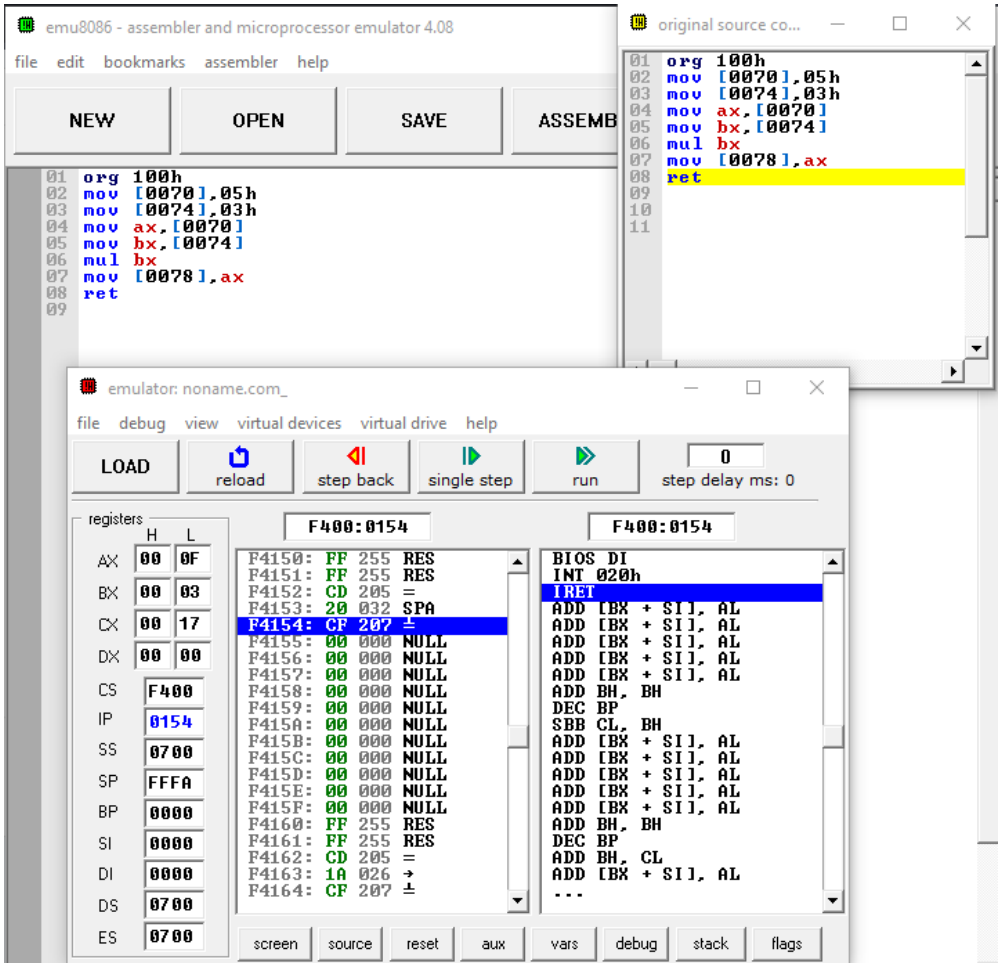
1)



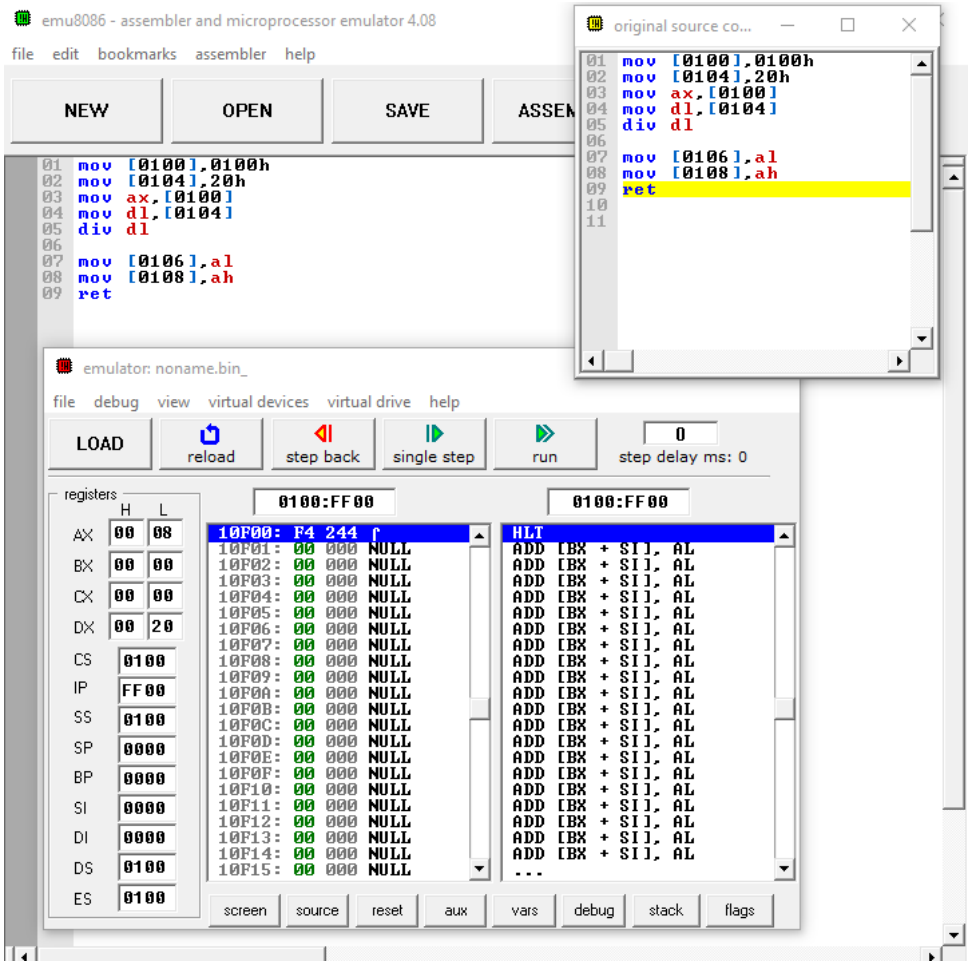
1 - SUB


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMB
01 org 100h
02 mov [0020],123h
03 mov [0022],5678h
04 mov [0024],9876h
05 mov [0026],7777h
06 mov [0030],1122h
07 mov [0032],3455h
08 mov [0034],1515h
09 mov [0036],7575h
10 mov ax,[0020]
11 mov bx,[0030]
original source co...
13 mov [0040],ax
14 mov ax,[0022]
15 mov bx,[0032]
16 sbb ax,bx
17 mov [0042],ax
18 mov ax,[0024]
19 mov bx,[0034]
20 sbb ax,bx
21 mov [0044],ax
22 mov ax,[0026]
23 mov bx,[0036]
24 sbb ax,bx
25 mov [0046],ax
26 mov ax,[0040]
27 mov bx,[0042]
28 mov cx,[0044]
29 mov dx,[0046]
30 ret
emulator: noname.com_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX F0 01
BX 22 22
CX 83 61
DX 02 02
CS F400
IP 0154
SS 0700
SP FFFA
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ±
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ±
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
...
screen
source
reset
aux
vars
debug
stack
flags

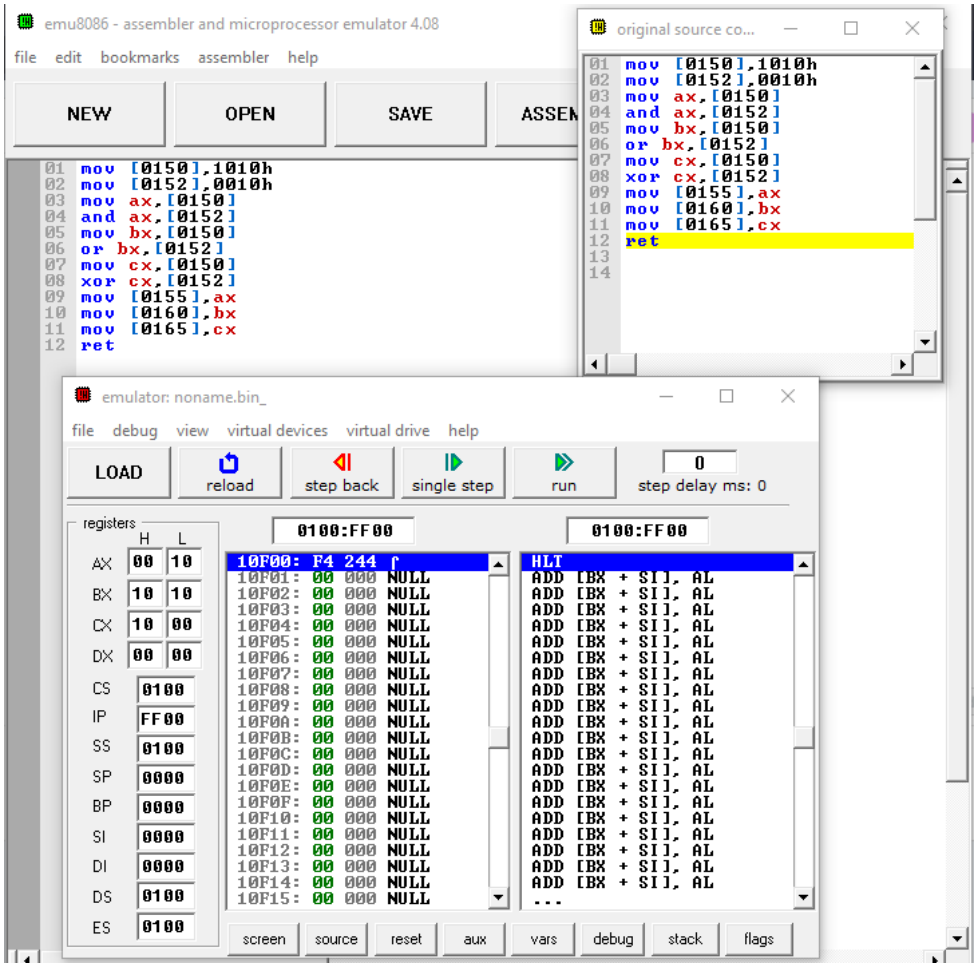
2)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMB
01 org 100h
02 mov [0070],05h
03 mov [0074],03h
04 mov ax,[0070]
05 mov bx,[0074]
06 mul bx
07 mov [0078],ax
08 ret
09
original source co...
01 org 100h
02 mov [0070],05h
03 mov [0074],03h
04 mov ax,[0070]
05 mov bx,[0074]
06 mul bx
07 mov [0078],ax
08 ret
09
10
11
emulator: noname.com_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX 00 0F
BX 00 03
CX 00 17
DX 00 00
CS F400
IP 0154
SS 0700
SP FFFA
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ±
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ±
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
ADD [BX + SI], AL
...
screen
source
reset
aux
vars
debug
stack
flags

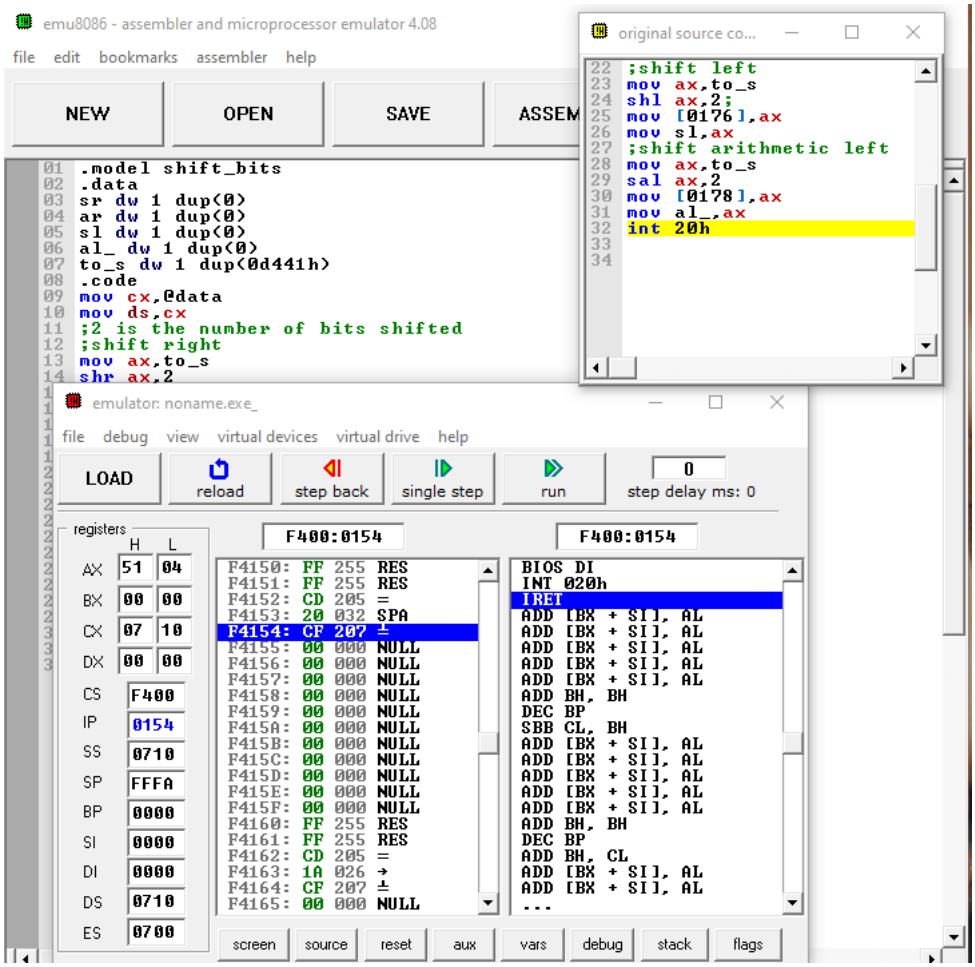
3)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
01 mov [0100],0100h
02 mov [0104],20h
03 mov ax,[0100]
04 mov dl,[0104]
05 div dl
07 mov [0106],al
08 mov [0108],ah
09 ret
original source co...
emulator: noname.bin_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay ms: 0
registers
AX 00 08
BX 00 00
CX 00 00
DX 00 20
CS 0100
IP FF00
SS 0100
SP 0000
BP 0000
SI 0000
DI 0000
DS 0100
ES 0100
0100:FF00
10F00: F4 244 HLT
screen source reset aux vars debug stack flags

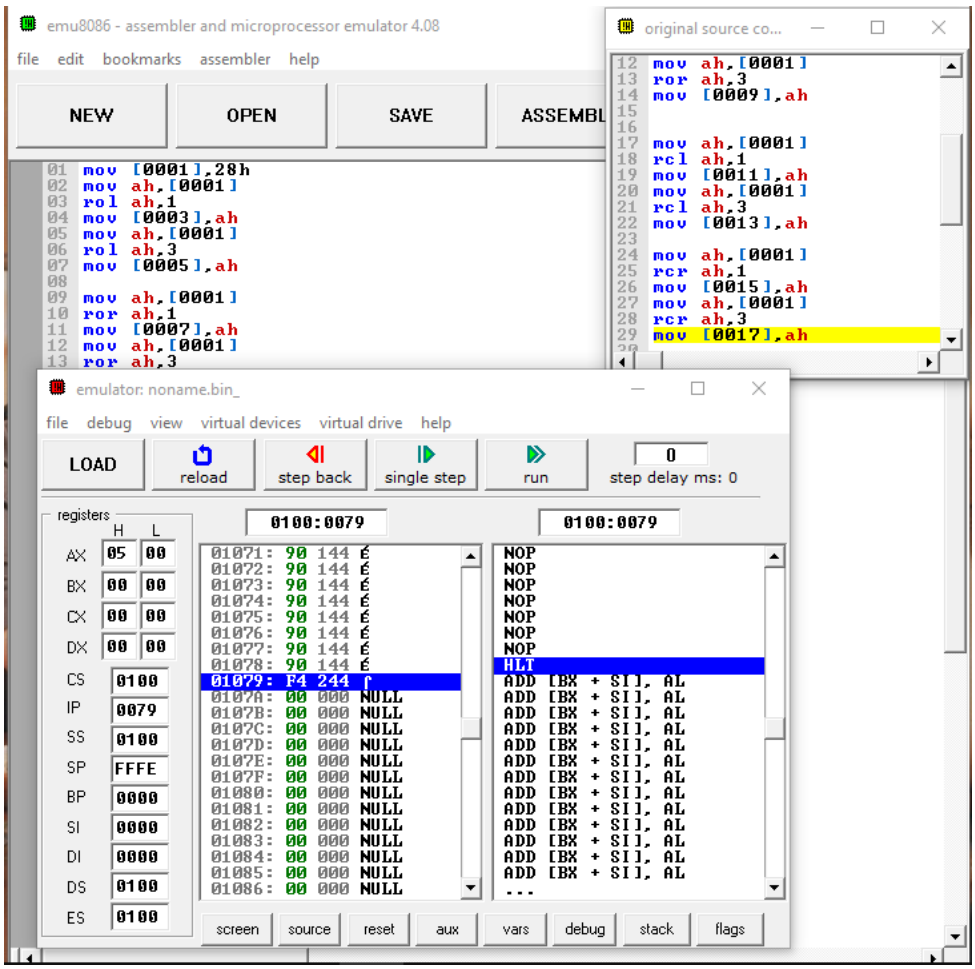
4)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEM
01 mov [0150],1010h
02 mov [0152],0010h
03 mov ax,[0150]
04 and ax,[0152]
05 mov bx,[0150]
06 or bx,[0152]
07 mov cx,[0150]
08 xor cx,[0152]
09 mov [0155],ax
10 mov [0160],bx
11 mov [0165],cx
12 ret
original source co...
emulator: noname.bin_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX 00 10
BX 10 10
CX 10 00
DX 00 00
CS 0100
IP FF00
SS 0100
SP 0000
BP 0000
SI 0000
DI 0000
DS 0100
ES 0100
0100:FF00
10F00: F4 244 ∫
10F01: 00 000 NULL
10F02: 00 000 NULL
10F03: 00 000 NULL
10F04: 00 000 NULL
10F05: 00 000 NULL
10F06: 00 000 NULL
10F07: 00 000 NULL
10F08: 00 000 NULL
10F09: 00 000 NULL
10F0A: 00 000 NULL
10F0B: 00 000 NULL
10F0C: 00 000 NULL
10F0D: 00 000 NULL
10F0E: 00 000 NULL
10F0F: 00 000 NULL
10F10: 00 000 NULL
10F11: 00 000 NULL
10F12: 00 000 NULL
10F13: 00 000 NULL
10F14: 00 000 NULL
10F15: 00 000 NULL
0100:FF00
HLT
ADD [BX + SI], AL
...
screen source reset aux vars debug stack flags

5)


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
.model shift_bits
.data
sr dw 1 dup(0)
ar dw 1 dup(0)
sl dw 1 dup(0)
al_ dw 1 dup(0)
to_s dw 1 dup(0d441h)
.code
mov cx,@data
mov ds,cx
;2 is the number of bits shifted
;shift right
mov ax,to_s
shr ax,2
original source co...
;shift left
mov ax,to_s
shl ax,2;
mov [0176],ax
mov sl,ax
;shift arithmetic left
mov ax,to_s
sal ax,2
mov [0178],ax
mov al_,ax
int 20h
emulator: noname.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
step delay ms: 0
registers
AX 51 04
BX 00 00
CX 07 10
DX 00 00
CS F400
IP 0154
SS 0710
SP FFFA
BP 0000
SI 0000
DI 0000
DS 0710
ES 0700
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ±
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
F4164: CF 207 ±
F4165: 00 000 NULL
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD BH, CL
...
screen
source
reset
aux
vars
debug
stack
flags

6)

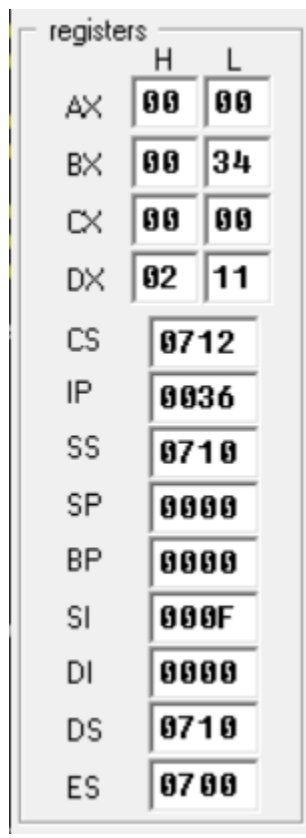

emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler help
NEW
OPEN
SAVE
ASSEMBL
01 mov [0001],28h
02 mov ah,[0001]
03 rol ah,1
04 mov [0003],ah
05 mov ah,[0001]
06 rol ah,3
07 mov [0005],ah
08
09 mov ah,[0001]
10 ror ah,1
11 mov [0007],ah
12 mov ah,[0001]
13 ror ah,3
original source co...
12 mov ah,[0001]
13 ror ah,3
14 mov [0009],ah
15
16
17 mov ah,[0001]
18 rcl ah,1
19 mov [0011],ah
20 mov ah,[0001]
21 rcl ah,3
22 mov [0013],ah
23
24 mov ah,[0001]
25 rcr ah,1
26 mov [0015],ah
27 mov ah,[0001]
28 rcr ah,3
29 mov [0017],ah
emulator: noname.bin_
file debug view virtual devices virtual drive help
LOAD
reload
step back
single step
run
0
step delay ms: 0
registers
H L
AX 05 00
BX 00 00
CX 00 00
DX 00 00
CS 0100
IP 0079
SS 0100
SP FFFE
BP 0000
SI 0000
DI 0000
DS 0100
ES 0100
0100:0079
01071: 90 144 É
01072: 90 144 É
01073: 90 144 É
01074: 90 144 É
01075: 90 144 É
01076: 90 144 É
01077: 90 144 É
01078: 90 144 É
01079: F4 244 ⌠
0107A: 00 000 NULL
0107B: 00 000 NULL
0107C: 00 000 NULL
0107D: 00 000 NULL
0107E: 00 000 NULL
0107F: 00 000 NULL
01080: 00 000 NULL
01081: 00 000 NULL
01082: 00 000 NULL
01083: 00 000 NULL
01084: 00 000 NULL
01085: 00 000 NULL
01086: 00 000 NULL
0100:0079
NOP
NOP
NOP
NOP
NOP
NOP
NOP
HLT
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
screen
source
reset
aux
vars
debug
stack
flags

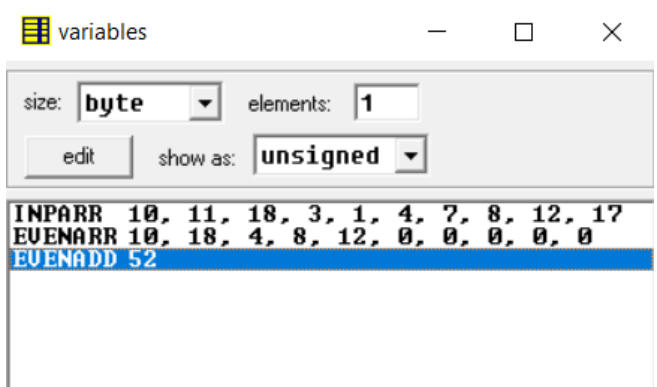
Practical : 8

AIM : Create an array. Perform addition of all even numbers from array and save answer in one variable.

model pract-8 array even number addition

```
.data
InpArrdb 10,11,18,3,1,4,7,8,12,17
EvenArrdb 10 dup(?)
EvenAdddb 0

.code
mov ax,@data
mov ds,ax

mov bx,offsetInpArr
mov si,offsetEvenArr

mov cx,10
mov dh,02
mov ah,00

next:
    mov al,[bx]
    mov dl,al
    div dh
cmp ah,00
    je even
inc bx
    Loop next
jmp finish

even:
    mov [si],dl
    add EvenAdd,dl
incsi
```





```
inc bx
    Loop next

finish:


  mov ax,0000
    mov bx,0000
    mov bl,EvenAdd
hlt
```

PRACTICAL : 9

AIM : String Handling in Assembly level language) Find out whether the given string is palindrome or not and print appropriate message. Don't use procedure.

```
org 100h
jmp start
m1:
s db 'abc'
s_size = $ - m1
 db 0Dh,0Ah,'$'
start:
; first let's print it:
mov ah, 9
mov dx, offset s
int 21h
lea di, s
mov si, di
add si, s_size
dec si ; point to last char!
mov cx, s_size
```

```
cmp cx, 1
je is_palindrome ; single char is always palindrome!
shr cx, 1 ; divide by 2!
next_char:
 mov al, [di]
 mov bl, [si]
 cmp al, bl
 jne not_palindrome
 inc di
 dec si
loop next_char
is_palindrome:
 ; the string is "palindrome!"
 mov ah, 9
 mov dx, offset msg1
 int 21h
jmp stop
not_palindrome:
 ; the string is "not palindrome!"
 mov ah, 9
 mov dx, offset msg2
 int 21h
stop:
; wait for any key press:
```

```
mov ah, 0
int 16h
ret
;DATA
msg1 db " this is a palindrome!$"
msg2 db " this is not a palindrome!$"
```

OUTPUT :

PRACTICAL :10

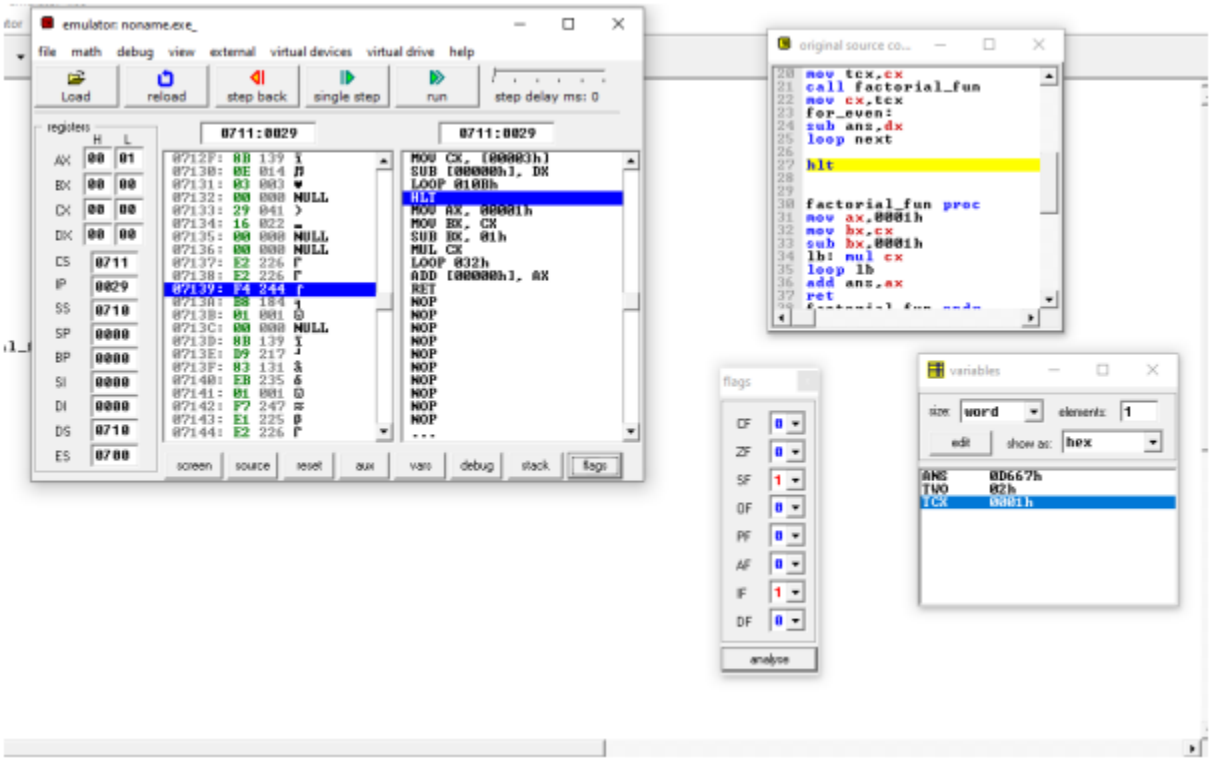
AIM: Wrioute an assembly code to evaluate the answer of blow given series and store the answer in ANS variable. Program shld have only one procedure to compute factorial of number. Series: 1! -2+3!-4+5!-6+7!-8+9!-10.

Code :

```
.model factorial
.data
ans dw 1 dup(0)
two db 1 dup(2)
tcx dw 1 dup(2)
.code
mov ax,@data
mov ds,ax
mov cx,0010h
mov ax,0001h
next: mov dx,cx
 mov ax,dx
 div two
 cmp ah,01h
 jne for_even
 mov tcx,cx
```

```
 call factorial_fun
 mov cx,tcx
 for_even:
 sub ans,dx
 loop next
hlt
factorial_fun proc
 mov ax,0001h
 mov bx,cx
 sub bx,0001h
 lb: mul cx
 loop lb
 add ans,ax
 ret
factorial_fun endp
```

Output :

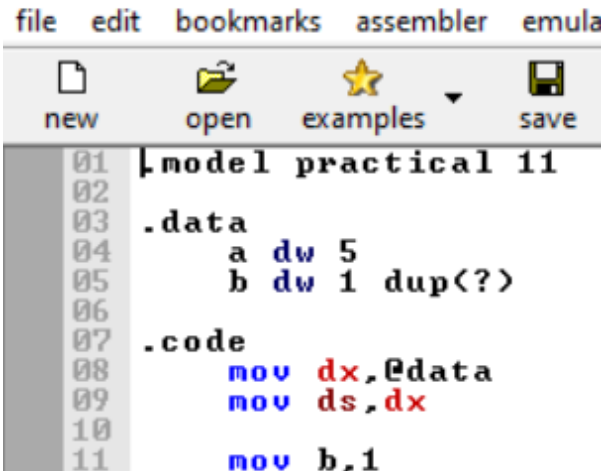


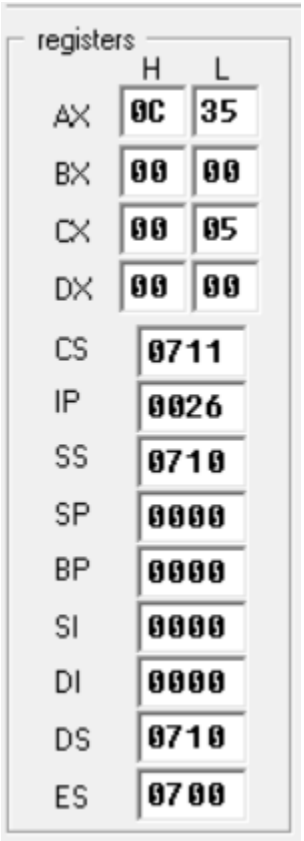
PRACTICAL : 11

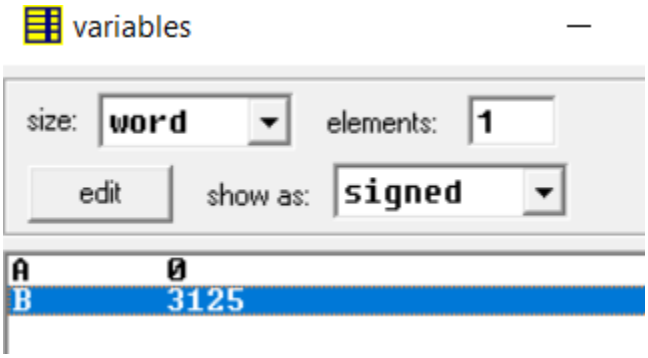
AIM: Write a assembly level code for given c program.

Task: Convert below C program in assembly level code

```
code.voidmain()
{
int a=5,b;b =1;
next : b = b * 5;a=a- 1;
if (a>=1)gotonext;
printf("%d",b);
}
```




registers
H
L
AX
0C
35
BX
00
00
CX
00
05
DX
00
00
CS
0711
IP
0026
SS
0710
SP
0000
BP
0000
SI
0000
DI
0000
DS
0710
ES
0700


variables
size:
word
elements:
1
edit
show as:
signed
A
0
B
3125