Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Experiment No: 1

Aim: To implement signed and unsigned multiplication.

(i) Booth's Algorithm:

Code:

```
def twosComplement(num):
    onesComp=""
    for i in num:
        if i == "0":
            onesComp += "1"
        else:
            onesComp +="0"

    return bin(int(onesComp,2) + int("1",2)).replace('0b',"")

num1 = int(input('Enter number: '))
num2 = int(input('Enter 2nd number: '))

binNum1 = bin(abs(num1)).replace("0b",'')
binNum2 = bin(abs(num2)).replace("0b",'')

if len(binNum1) >= len(binNum2):
    maxlen = len(binNum1)

else:
    maxlen = len(binNum2)


maxlen +=1

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen)
if num2 < 0:
    binNum2 = twosComplement(binNum2)
```

```
if num1 < 0:
   binNum1 = twosComplement(binNum1)

binCompNum1 = twosComplement(binNum1)
binCompNum1 = binCompNum1.zfill(maxlen)
print(binNum1)
print(binNum2)
print(binCompNum1)

count = maxlen
m = binNum1
minusm = binCompNum1
q = binNum2
q1 = '0'
a = "0"
a = a.zfill(maxlen)
rightshift=""
while count > 0:
   if q1 == '1' and q[maxlen-1] == '0':
       a = bin(int(a,2) + int(m,2)).replace('0b','')
       if(len(a) > maxlen):
           a = a[1:]
       a = a.zfill(maxlen)

   elif q1=='0' and q[maxlen-1] == '1':
       a = bin(int(a,2) + int(minusm,2)).replace('0b','')
       if(len(a) > maxlen):
           a = a[1:]

       a = a.zfill(maxlen)

   merged = a+q+q1
   rightshift = merged[0]

   for i in range(len(merged)-1):
       rightshift += merged[i]

   a = rightshift[:maxlen]
   q = rightshift[maxlen:maxlen*2]
```

```
    q1 = rightshift[-1]
    count -=1

ans = a+q
minus = False
if ans[0] == '1':
    ans = twosComplement(ans)
    minus = True
print(ans)
if minus:
    print(int(ans,2) * -1)
else:
    print(int(ans,2))
```

Output:

```
Enter number: 5
Enter 2nd number: -3
0101
1101
1011
1111
-15
```

(ii) Unsigned Multiplication:

Code:

```
def binary(a, b):
    a1 = abs(a)
    b1 = abs(b)
    com = [1, 0, 0, 0, 0, 0, 0, 0]
    anum = [0] * 8
    anumcp = [0] * 8
    bnum = [0] * 8
    acomp = [0] * 8
    bcomp = [0] * 8
    pro = [0] * 8
    res = [0] * 8
    for i in range(8):
        r = a1 % 2
        a1 = a1 // 2
        r2 = b1 % 2
        b1 = b1 // 2
        anum[i] = r
        anumcp[i] = r
        bnum[i] = r2
        if r2 == 0:
            bcomp[i] = 1
        if r == 0:
            acomp[i] = 1
    c = 0
    for i in range(8):
        res[i] = com[i] + bcomp[i] + c
        if res[i] >= 2:
            c = 1
        else:
            c = 0
        res[i] = res[i] % 2
        bcomp[i] = res[i]
    if a < 0:
```

```
        c = 0
        for i in range(8):
            res[i] = 0
        for i in range(8):

            res[i] = com[i] + acomp[i] + c
            if res[i] >= 2:
                c = 1
            else:
                c = 0
            res[i] = res[i] % 2
            anum[i] = res[i]
            anumcp[i] = res[i]
    if b < 0:
        for i in range(8):
            temp = bnum[i]
            bnum[i] = bcomp[i]
            bcomp[i] = temp
    return anum, bnum, bcomp, pro, anumcp
def add(num, pro, anumcp):
    res = [0] * 8
    c = 0
    for i in range(8):
        res[i] = pro[i] + num[i] + c
        if res[i] >= 2:
            c = 1
        else:
            c = 0
        res[i] = res[i] % 2
        pro[i] = res[i]
    return pro, anumcp
def arshift(pro, anumcp):
    temp = pro[7]
    temp2 = pro[0]
    for i in range(1, 8):
        pro[i - 1] = pro[i]
    pro[7] = temp
    for i in range(1, 8):
        anumcp[i - 1] = anumcp[i]
```

```
    anumcp[7] = temp2

    return pro, anumcp


def booth_multiplication(a, b):

    anum, bnum, bcomp, pro, anumcp = binary(a, b)
    q = 0
    result = []
    for i in range(8):
        if anum[i] == q:
            result.append(pro)
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
        elif anum[i] == 1 and q == 0:
            result.append(pro)
            pro, anumcp = add(bcomp, pro, anumcp)
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
        else:
            result.append(pro)
            pro, anumcp = add(bnum, pro, anumcp)
            pro, anumcp = arshift(pro, anumcp)
            q = anum[i]
    final_product = [0] * 16
    for i in range(8):
        final_product[i] = anumcp[i]
    for i in range(8, 16):
        final_product[i] = result[-1][i - 8]
    return final_product
if __name__ == "__main__":
    a = int(input("Enter A: "))
    b = int(input("Enter B: "))
    if abs(a) > 255 or abs(b) > 255:
        print("Both numbers must be integers in the range
             (-256 to 255).")
    else:
        result = booth_multiplication(a, b)
```

```
        print("\nProduct is =", end=" ")
        for i in reversed(result):
            print(i, end="")
        print()
```

Output:

```
Enter A: 5
Enter B: 3
Product is = 0000000000001111
```

Conclusion:

Booth’s algorithm efficiently multiplies signed binary numbers, handling both positive and negative multipliers uniformly. In contrast, unsigned multiplication is simpler but only deals with positive numbers, limiting its applicability.

Experiment No: 2

(i) Restoring Division:

Code:

```
def twosComplement(num):
    onesComp=""
    for i in num:
        if i == "0":
            onesComp += "1"
        else:

            onesComp +="0"

    return bin(int(onesComp,2) + int("1",2)).replace('0b',"")

num1 = int(input('Enter number: '))
num2 = int(input('Enter 2nd number: '))

binNum1 = bin(abs(num1)).replace("0b",'')
binNum2 = bin(abs(num2)).replace("0b",'')

maxlen = len(binNum1)

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen + 1)

binCompNum2 = twosComplement(binNum2)
binCompNum2 = binCompNum2.zfill(maxlen)

count = maxlen
m = binNum2
minusm = binCompNum2
q = binNum1
a = "0"
```

```
a = a.zfill(maxlen+1)
leftshift=""

while count > 0:
   merged = a+q
   leftshift = merged[1:]
   a = leftshift[:maxlen+1]
   a = bin(int(a,2)+int(minusm,2)).replace("0b","")
   if len(a) > maxlen+1:
       a=a[1:]
   a = a.zfill(maxlen+1)

   if a[0] == "0":
       leftshift = a+q[1:]
       leftshift += "1"

   else:
       a = bin(int(a,2)+int(m,2)).replace("0b","")
       if len(a) > maxlen+1:
           a=a[1:]
       a = a.zfill(maxlen+1)
       leftshift = a+q[1:]
       leftshift += "0"

   a = leftshift[:maxlen+1]
   q = leftshift[maxlen+1:]
   count -=1

if a[0] == "1":

   a = bin(int(a,2)+int(m,2)).replace("0b","")
   if len(a) > maxlen+1:
       a = a[1:]


print("Remainder",int(a,2))
print("Quotient",int(q,2))
```

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Output:

```
Enter number: 11
Enter 2nd number: 3
Remainder 2
Quotient 3
```

(ii) Non-Restoring Division:

Code:

```
def twosComplement(num):
   onesComp=""
   for i in num:
       if i == "0":
           onesComp += "1"
       else:
           onesComp +="0"

   return bin(int(onesComp,2) + int("1",2)).replace('0b',"")

num1 = int(input('Enter number: '))
num2 = int(input('Enter 2nd number: '))


binNum1 = bin(abs(num1)).replace("0b",'')
binNum2 = bin(abs(num2)).replace("0b",'')


maxlen = len(binNum1)

binNum1 = binNum1.zfill(maxlen)
binNum2 = binNum2.zfill(maxlen + 1)

binCompNum2 = twosComplement(binNum2)
binCompNum2 = binCompNum2.zfill(maxlen)

count = maxlen
m = binNum2
minusm = binCompNum2
q = binNum1
a = "0"
a = a.zfill(maxlen+1)
```

```
leftshift=""

while count > 0:
   merged = a+q
   leftshift = merged[1:]
   a = leftshift[:maxlen+1]

   if a[0] == "1":
       a = bin(int(a,2)+int(m,2)).replace("0b","")
       if len(a) > maxlen+1:
           a=a[1:]
       a = a.zfill(maxlen+1)
   else:
       a = bin(int(a,2)+int(minusm,2)).replace("0b","")
       if len(a) > maxlen+1:
           a=a[1:]
       a = a.zfill(maxlen+1)


   leftshift = a+q[1:]

   if a[0] == "1":
       leftshift += "0"
   else:

       leftshift +="1"

   a = leftshift[:maxlen+1]
   q = leftshift[maxlen+1:]
   count -=1


if a[0] == "1":

   a = bin(int(a,2)+int(m,2)).replace("0b","")
   if len(a) > maxlen+1:
       a = a[1:]
```

```
print("Remainder",int(a,2))
print("Quotient",int(q,2))
```

Output:

```
Enter number: 11
Enter 2nd number: 3
Remainder 2
Quotient 3
```

Conclusion:

Restoring division, though relatively more complex and slower due to the need for restoration steps, offers a straightforward method for performing division. It is suitable for both hardware and software implementations where simplicity is not a primary concern, making it a viable choice in many computing systems. On the other hand, non-restoring division is computationally efficient, primarily suited for hardware-based division units. It employs a more intricate mechanism by avoiding restoration steps and complementing the divisor when necessary, resulting in faster execution. However, it might be less intuitive for software implementations. The choice between these algorithms depends on the specific requirements of the application, where factors like hardware constraints, execution speed, and ease of implementation play a significant role in determining which algorithm is more suitable.

Department of Computer Engineering

Class: T.Y. B.Tech. Semester: V

Course Code: DJ19CEL502 Course Name: Processor Organization & Architecture

Experiment No: 3

(i) Best Fit:

Code:

```
#include<iostream>
using namespace std;
void bestFit(int blockSize[], int m, int processSize[], int n)
{
   int allocation[n];
   for (int i = 0; i < n; i++)
       allocation[i] = -1;
   for (int i = 0; i < n; i++)
   {
       int bestIdx = -1;
       for (int j = 0; j < m; j++)
       {
           if (blockSize[j] >= processSize[i])
           {
               if (bestIdx == -1)
                   bestIdx = j;
               else if (blockSize[bestIdx] > blockSize[j])
                   bestIdx = j;
           }
       }
       if (bestIdx != -1)
       {
           allocation[i] = bestIdx;
           blockSize[bestIdx] -= processSize[i];
       }}
   cout << "\nProcess No.\tProcess Size\tBlock no.\n";
   for (int i = 0; i < n; i++)
   {
       cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
```

```
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    bestFit(blockSize, m, processSize, n);
    return 0 ;
}
```

Output:

```
Process No. Process Size    Block no.
 1         212      4
 2         417      2
 3         112      3
 4         426      5
```

(ii) First Fit:

Code:

```
#include<stdio.h>
void firstFit(int blockSize[], int m, int processSize[], int n)
{
   int i, j;
   int allocation[n];
   for(i = 0; i < n; i++)
   {
       allocation[i] = -1;
   }
   for (i = 0; i < n; i++)
   {
       for (j = 0; j < m; j++)
       {
           if (blockSize[j] >= processSize[i])
           {
               allocation[i] = j;
               blockSize[j] -= processSize[i];
               break;
           }

       }
   }
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (int i = 0; i < n; i++)
   {
       printf(" %i\t\t\t", i+1);
       printf("%i\t\t\t\t", processSize[i]);
       if (allocation[i] != -1)
           printf("%i", allocation[i] + 1);
       else
           printf("Not Allocated");
       printf("\n");
   }
}
```

```
int main()
{
    int m;
    int n;
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    m = sizeof(blockSize) / sizeof(blockSize[0]);
    n = sizeof(processSize) / sizeof(processSize[0]);
    firstFit(blockSize, m, processSize, n);
    return 0 ;
}
```

Output:

```
Process No. Process Size    Block no.
 1          212             2
 2          417             5
 3          112             2
 4          426             Not Allocated
```

(iii) Next Fit:

Code:

```
#include <bits/stdc++.h>
using namespace std;
void NextFit(int blockSize[], int m, int processSize[], int n)
{
   int allocation[n], j = 0, t = m - 1;
   memset(allocation, -1, sizeof(allocation));
   for(int i = 0; i < n; i++){
       while (j < m){
           if(blockSize[j] >= processSize[i]){
               allocation[i] = j;
               blockSize[j] -= processSize[i];
               t = (j - 1) % m;
               break;
           }
           if (t == j){
               t = (j - 1) % m;
               break;
           }
           j = (j + 1) % m;
       }
   }
   cout << "\nProcess No.\tProcess Size\tBlock no.\n";
   for (int i = 0; i < n; i++) {
       cout << " " << i + 1 << "\t\t\t\t" << processSize[i]

           << "\t\t\t\t";
       if (allocation[i] != -1)
           cout << allocation[i] + 1;
       else
           cout << "Not Allocated";
       cout << endl;
   }
}
```

```
int main()
{
    int blockSize[] = { 5, 10, 20 };
    int processSize[] = { 10, 20, 5 };
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    NextFit(blockSize, m, processSize, n);
    return 0;
}
```

Output:

```
Process No. Process Size    Block no.
 1               10              2
 2               20              3
 3               5               1
```

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**
Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

(iv) Worst Fit:
Theory:

Code:

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

void worstFit(int blockSize[], int m, int processSize[], int n)
{
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));
    for (int i=0; i<n; i++)
    {
        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }
        if (wstIdx != -1)
        {
            allocation[i] = wstIdx;
            blockSize[wstIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
```

```
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }}
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);
    worstFit(blockSize, m, processSize, n);
    return 0 ;
}
```

Output:

```
Process No. Process Size    Block no.
 1       212      5
 2       417      2
 3       112      5
 4       426      Not Allocated
```

Department of Computer Engineering

Class: T.Y. B.Tech. Semester: V

Course Code: DJ19CEL502 Course Name: Processor Organization & Architecture

Experiment No: 4

Page Replacement Algorithms:

(i) Optimal:

Code:

```
#include <bits/stdc++.h>
using namespace std;
bool search(int key, vector<int>& fr)
{
   for (int i = 0; i < fr.size(); i++)
       if (fr[i] == key)
           return true;
   return false;
}
int predict(int pg[], vector<int>& fr, int pn, int index)
{
   int res = -1, farthest = index;
   for (int i = 0; i < fr.size(); i++) {
       int j;
       for (j = index; j < pn; j++) {
           if (fr[i] == pg[j]) {
               if (j > farthest) {
                   farthest = j;
                   res = i;
               }

               break;
           }}
       if (j == pn)
           return i; }

     return (res == -1) ? 0 : res;
}
void optimalPage(int pg[], int pn, int fn)
{
```

```
    vector<int> fr;
    int hit = 0;
    for (int i = 0; i < pn; i++) {
        if (search(pg[i], fr)) {
            hit++;
            continue;
        }
        if (fr.size() < fn)
            fr.push_back(pg[i]);
        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i]; } }
    cout << "No. of hits = " << hit << endl;
    cout << "No. of misses = " << pn - hit << endl;
}
int main()
 {
    int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
    int pn = sizeof(pg) / sizeof(pg[0]);
    int fn = 4;
    optimalPage(pg, pn, fn);
    return 0;
}
```

Output:

```
No. of hits = 7
No. of misses = 6
```

Department of Computer Engineering

Class: T.Y. B.Tech. Semester: V
Course Code: DJ19CEL502 Course Name: Processor Organization & Architecture

(ii) FIFO:

Code:

```
#include<bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
    unordered_set<int> s;
    queue<int> indexes;
    int page_faults = 0;
    for (int i=0; i<n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i])==s.end())
            {
                s.insert(pages[i]);
                page_faults++;
                indexes.push(pages[i]);
            }
        }
        else
        {
            if (s.find(pages[i]) == s.end())
            {
                int val = indexes.front();
                indexes.pop();
                s.erase(val);

                s.insert(pages[i]);
                indexes.push(pages[i]);
                page_faults++;
            }
        }
    }
    return page_faults;
}
```

```
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    int x =pageFaults(pages, n, capacity);
    cout << "No of miss " <<x<<endl;
    cout << "No of hits "<< n-x;
    return 0;
}
```

Output:

```
No of miss 7
No of hits 6
```

(iii) LFU:

Code:

```
#include <bits/stdc++.h>
using namespace std;
int pageFaults(int n, int c, int pages[])
{
    int count = 0;
    vector<int> v;
    unordered_map<int, int> mp;
    int i;
    for (i = 0; i <= n - 1; i++) {
        auto it = find(v.begin(), v.end(), pages[i]);
        if (it == v.end()) {
            if (v.size() == c) {
                mp[v[0]]--;
                v.erase(v.begin());
            }
            v.push_back(pages[i]);
            mp[pages[i]]++;
            count++;
        }
        else {
            mp[pages[i]]++;
            v.erase(it);
            v.push_back(pages[i]);
        }
        int k = v.size() - 2;
        while (mp[v[k]] > mp[v[k + 1]] && k > -1) {
            swap(v[k + 1], v[k]);
            k--;
        }
    }
    return count;
}
```

```
int main()
{

    int pages[] = { 1, 2, 3, 4, 2, 1, 5 };
    int n = 7, c = 3;

    cout << "Page Faults = " << pageFaults(n, c, pages)
        << endl;
    cout << "Page Hits = " << n - pageFaults(n, c, pages);
    return 0;
}
```

Output:

```
Page Faults = 6
Page Hits = 1
```

(iv) LRU:

Code:

```
#include<bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
   unordered_set<int> s;
   unordered_map<int, int> indexes;
   int page_faults = 0;
   for (int i=0; i<n; i++)
   {
       if (s.size() < capacity)
       {
           if (s.find(pages[i])==s.end())
           {
               s.insert(pages[i]);
               page_faults++;
           }
           indexes[pages[i]] = i;
       }
       else
       {
           if (s.find(pages[i]) == s.end())
           {
               int lru = INT_MAX, val;
               for (auto it=s.begin(); it!=s.end(); it++)
               {
                   if (indexes[*it] < lru)
                   {
                       lru = indexes[*it];
                       val = *it;
                   }
               }
               s.erase(val);
               s.insert(pages[i]);
               page_faults++;
```

```
            }
            indexes[pages[i]] = i;
        }
    }
    return page_faults;
}
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    int x =pageFaults(pages, n, capacity);
    cout << "No of miss " <<x<<endl;
    cout << "No of hits "<< n-x;
    return 0;
}
```

Output:

```
No of miss 6
No of hits 7
```

Conclusion: Thus we implemented page replacement algorithms for the given question to assign pages to frames in memory.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Experiment No: 5

Code:

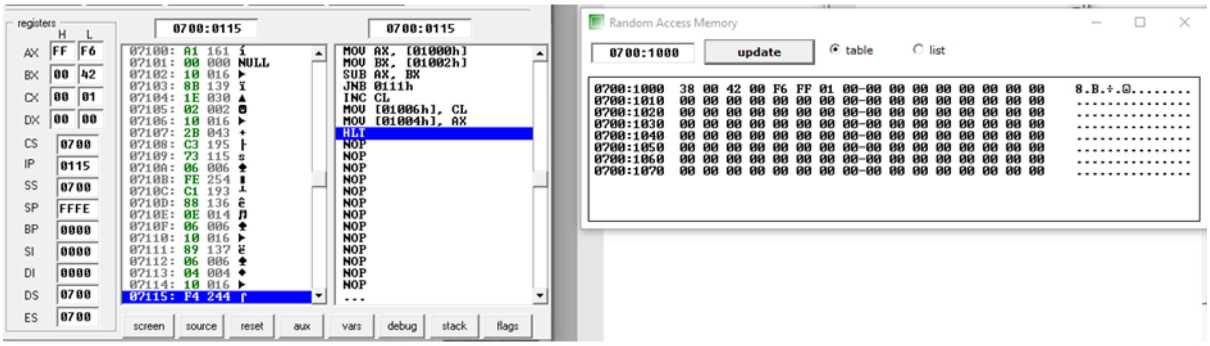
(i) Addition-

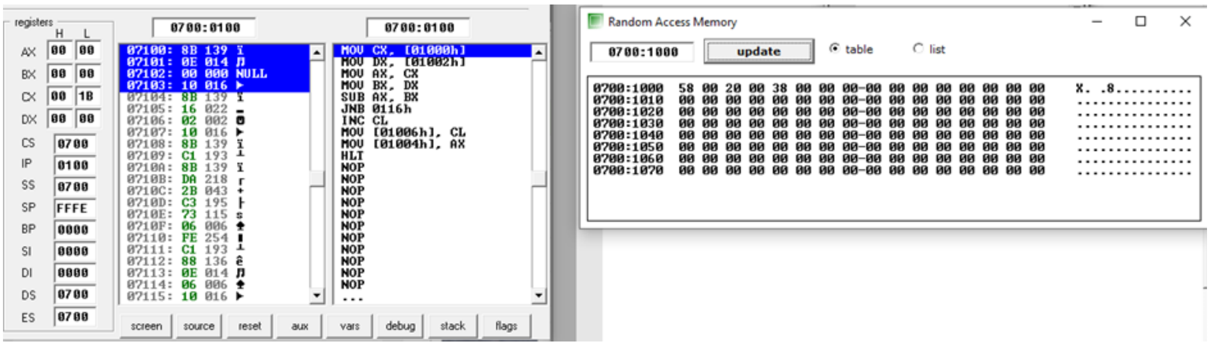
```
data segment

a dw 0202h

b dw 0408h

c dw ?

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov ax,a

mov bx,b

add ax,bx

mov c,ax

int 3

code ends

end start
```

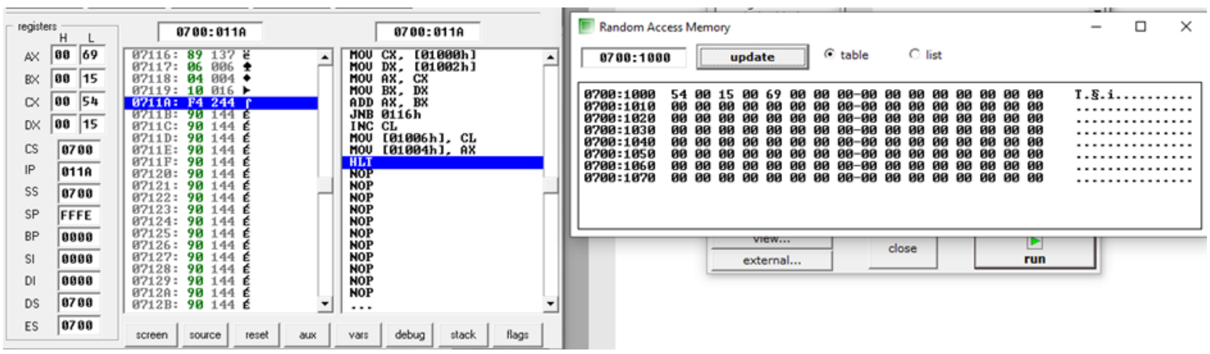
(ii) Subtraction-

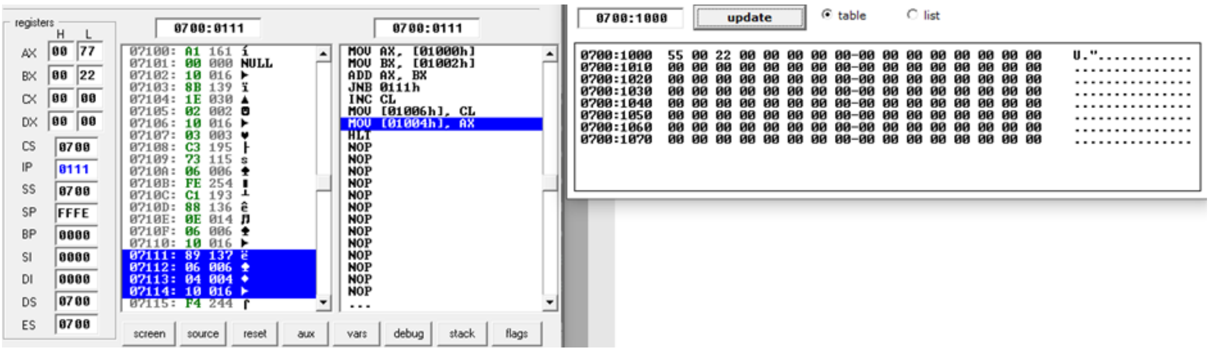
```
data segment
a dw 9A88h
b dw 8765h
c dw ?
data ends
 code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov ax,a
mov bx,b
sub ax,bx
mov c,ax
int 3
code ends
end start
```

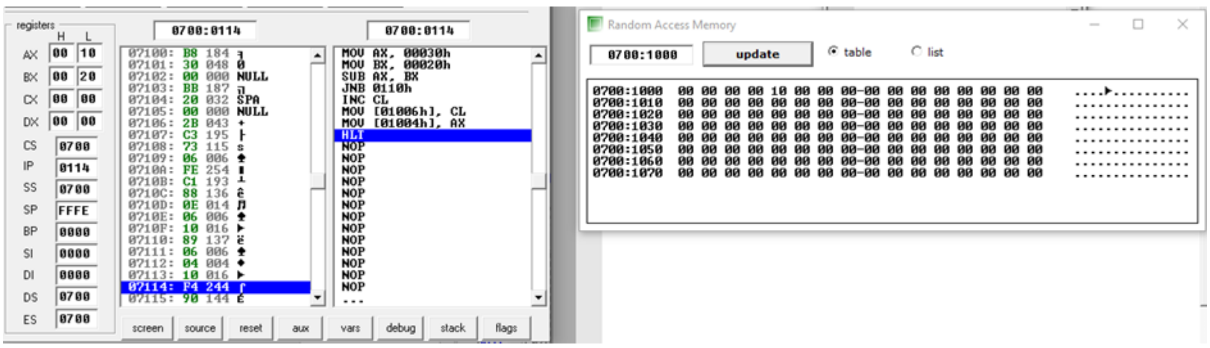
Output:


registers
0700:0115
MOV AX, [01000h]
MOV BX, [01002h]
SUB AX, BX
JNB 0111h
INC CL
MOV [01006h], CL
MOV [01004h], AX
HLT
Random Access Memory
0700:1000
update
table
list
screen
source
reset
aux
vars
debug
stack
flags


registers
0700:0100
MOV CX, [01000h]
MOV DX, [01002h]
MOV AX, CX
MOV BX, DX
SUB AX, BX
JNB 0116h
INC CL
MOV [01006h], CL
MOV [01004h], AX
HLT
Random Access Memory
0700:1000
update
table
list
screen
source
reset
aux
vars
debug
stack
flags


registers
0700:011A
MOV CX, [01000h]
MOV DX, [01002h]
MOV AX, CX
MOV BX, DX
ADD AX, BX
JNB 0116h
INC CL
MOV [01006h], CL
MOV [01004h], AX
HLT
Random Access Memory
0700:1000
update
table
list
external...
close
run
screen
source
reset
aux
vars
debug
stack
flags





Conclusion: Thus, we have successfully implemented 16-bit addition and subtraction in assembly language.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**
Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Experiment No: 6

(i) Ascending Order

Code:

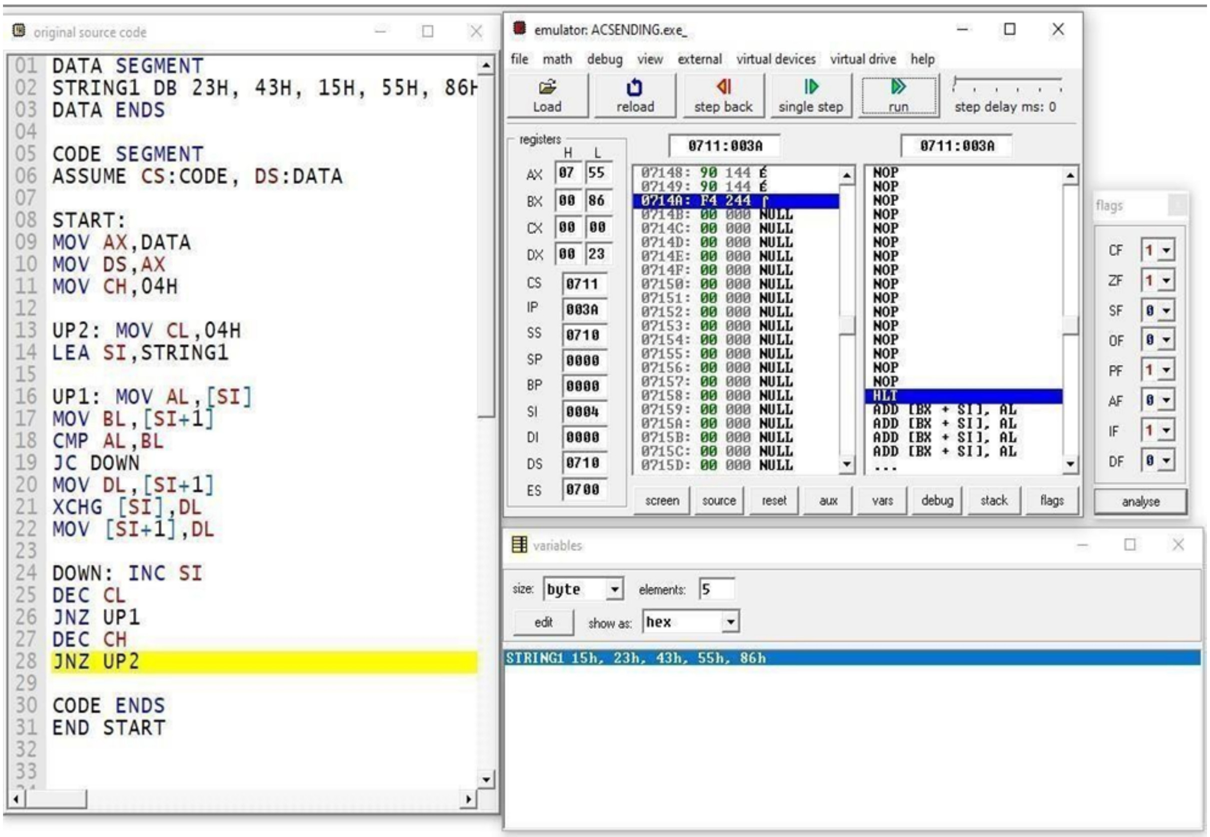
```
DATA SEGMENT
STRING1 DB 99H,12H,56H,45H,36H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H
LEA SI,STRING1

UP1: MOV AL,[SI]
MOV BL,[SI+1]
CMP AL,BL
JC DOWN
MOV DL,[SI+1]
XCHG [SI],DL
MOV [SI+1],DL

DOWN: INC SI
DEC CL
JNZ UP1
DEC CH
JNZ UP2

CODE ENDS
END START
```

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Output:



(ii) Descending Order

Code:

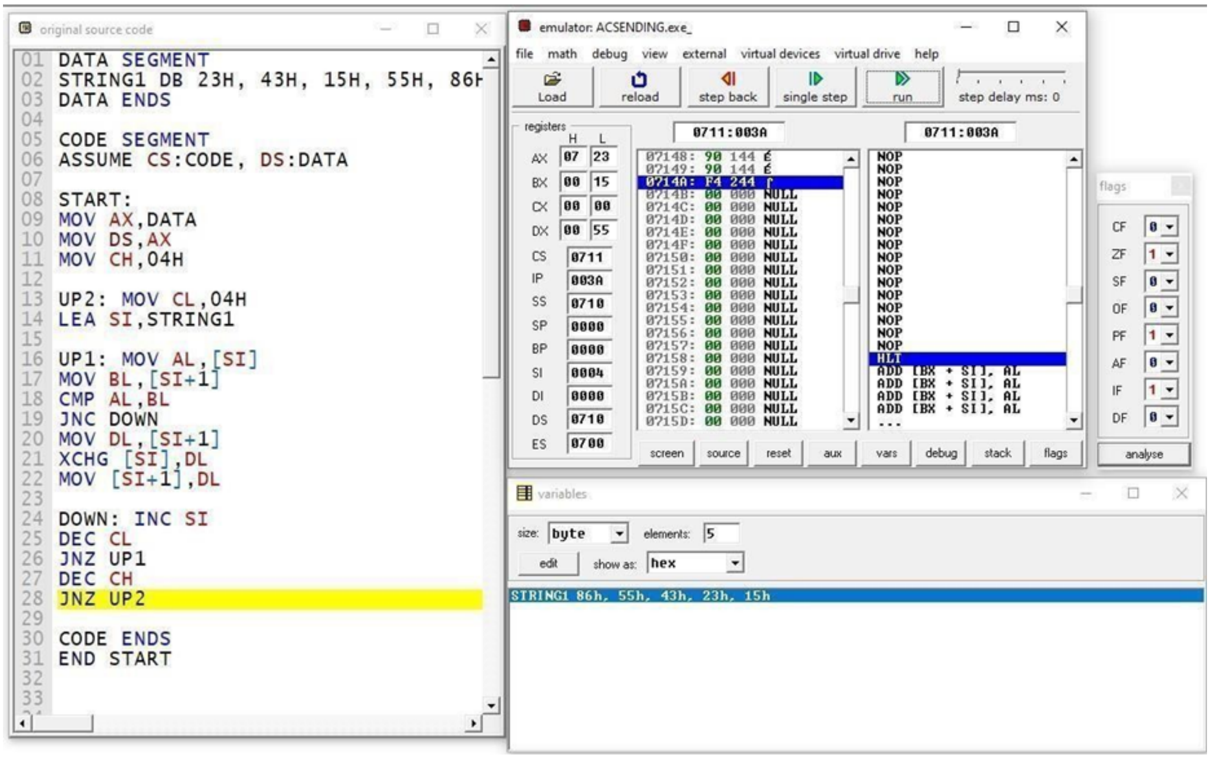
```
DATA SEGMENT
STRING1 DB 99H,12H,56H,45H,36H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H
LEA SI,STRING1


UP1:MOV AL,[SI]
MOV BL,[SI+1]
CMP AL,BL
JNC DOWN
MOV DL,[SI+1]
XCHG [SI],DL
MOV [SI+1],DL

DOWN: INC SI
DEC CL
JNZ UP1
DEC CH
JNZ UP2

CODE ENDS
END START
```

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Output:



<u>Conclusion:</u> We successfully program to sort numbers in ascending/ descending order.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**
Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Experiment No: 7

Code:

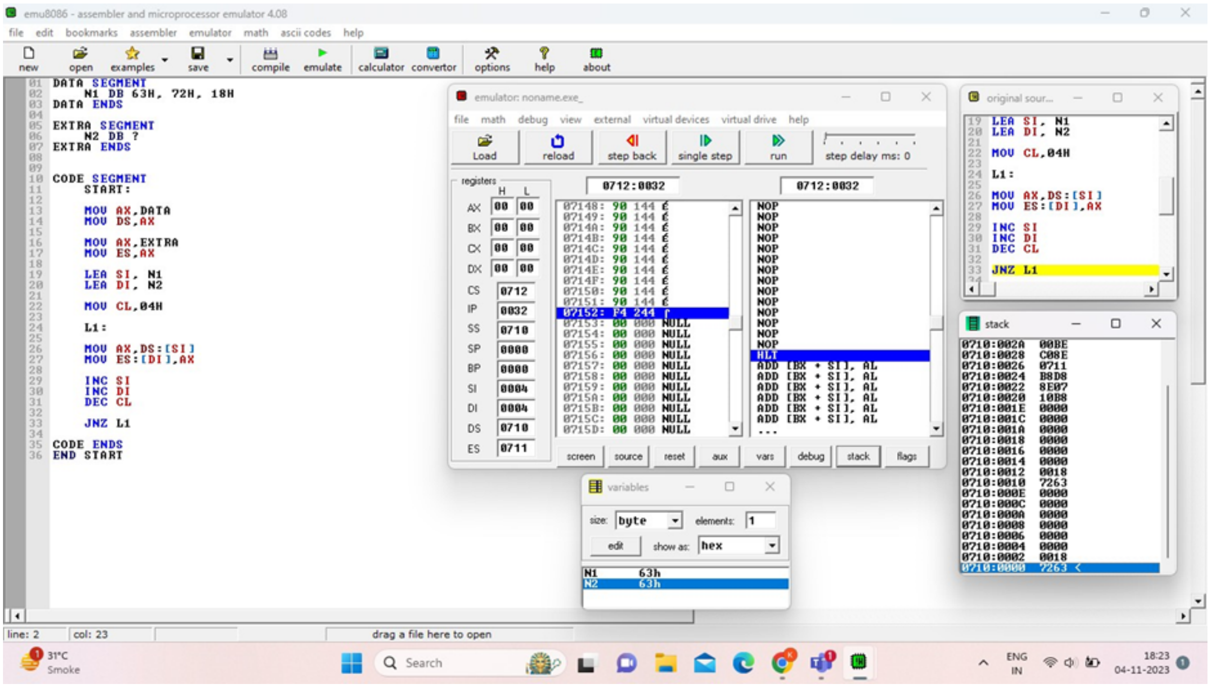
```
; multi-segment executable file template
data segment
seg1 db 1h ,2h ,3h
ends
extra segment
seg2 db ?
ends
code segment
start:
; set segment registers:
mov ax, data
mov ds, ax
mov ax, extra
mov es, ax
; add your code here
lea si , seg1
lea di , seg2
mov cx, 03h
x: mov ah,ds:[si]
mov es:[di],ah

inc si
inc di
dec cx
jnz x
int 3

ends
end start ; set entry point and stop the assembler.
```

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

Output:



Conclusion: In this experiment, we successfully developed an assembly program to transfer n blocks of data from one memory segment to another using the emu8086 emulator. This program is a fundamental example of memory manipulation in low-level programming.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**
Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

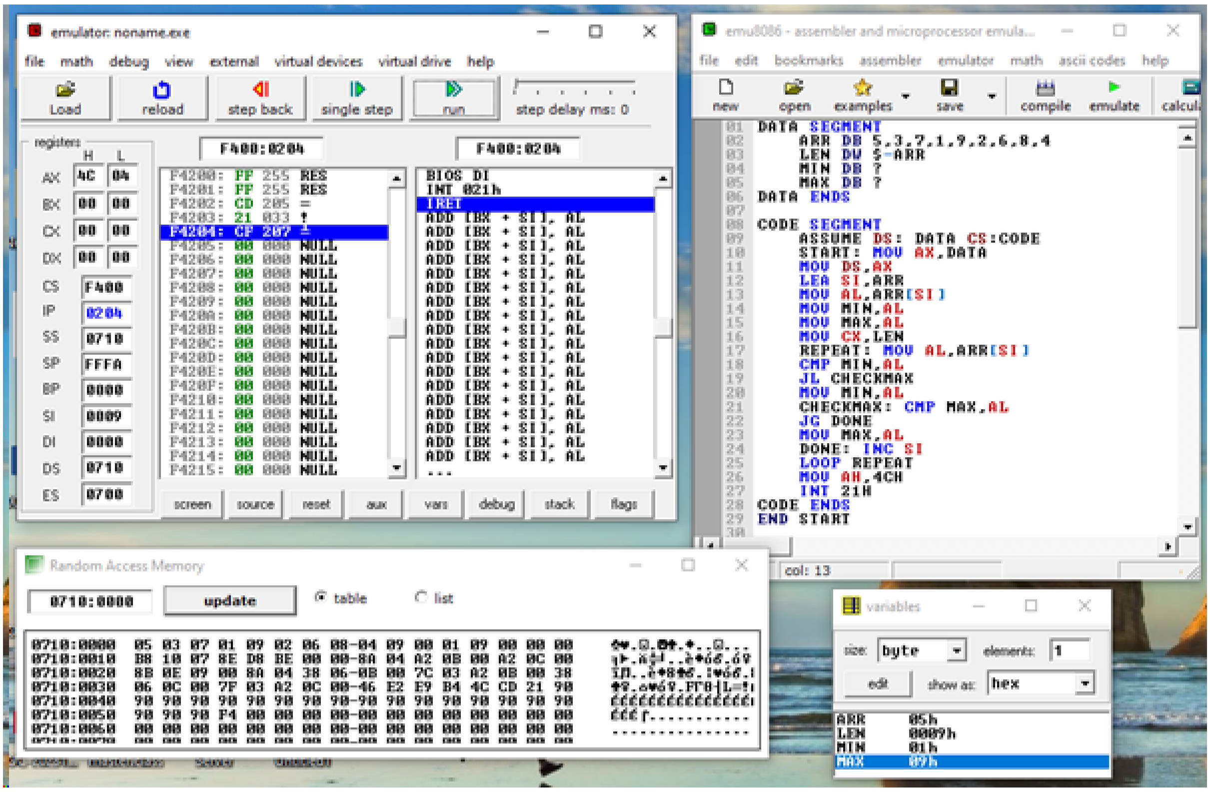
Experiment No: 8

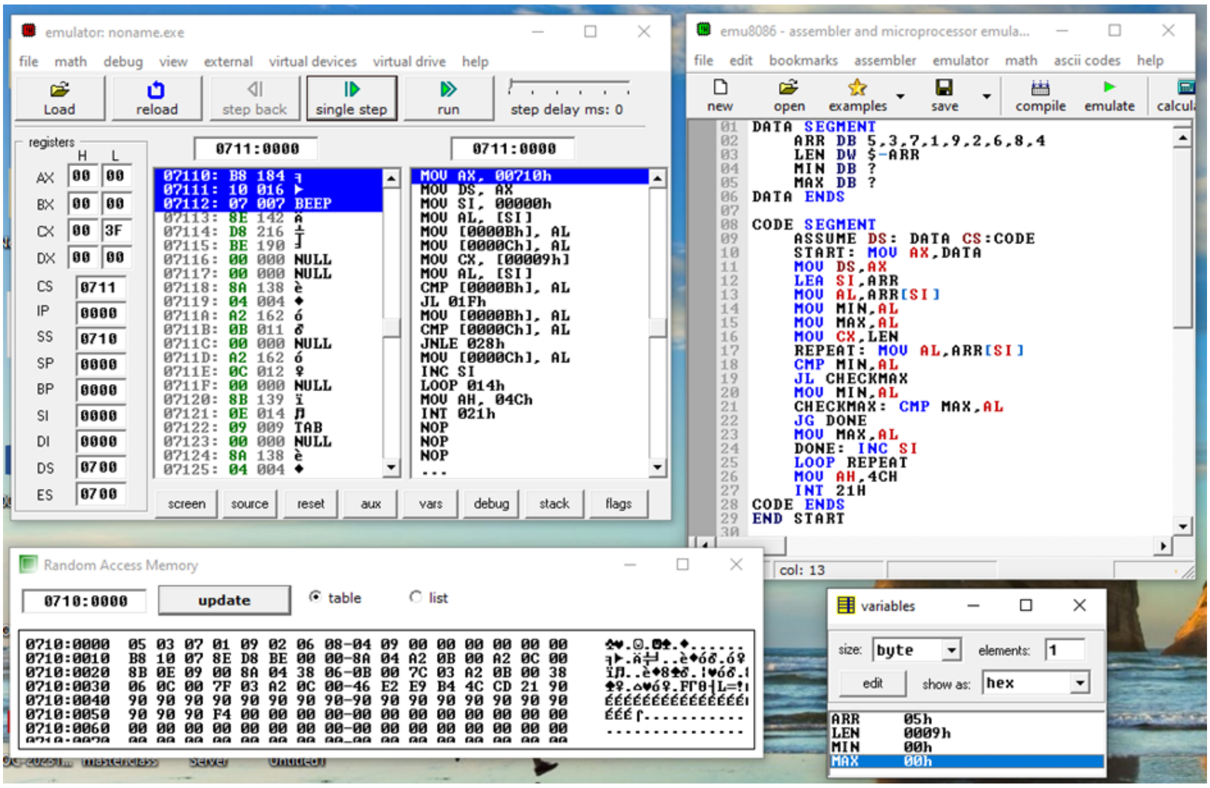
Code:

```
DATA SEGMENT
ARR DB 5,3,7,1,9,2,6,8,4
LEN DW $-ARR
MIN DB ?
MAX DB ?
DATA ENDS

CODE SEGMENT
ASSUME DS:DATA CS:CODE
START:
MOV AX,DATA
MOV DS,AX

LEA SI,ARR
MOV AL,ARR[SI]
MOV MIN,AL
MOV MAX,AL

MOV CX,LEN
REPEAT:
MOV AL,ARR[SI]
CMP MIN,AL
JL CHECKMAX

MOV MIN,AL
CHECKMAX:
CMP MAX,AL
JG DONE

MOV MAX,AL
DONE:
```

```
INC SI
LOOP REPEAT

MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Output:





Conclusion: Therefore we have successfully implemented the code to transfer a block of given size n using assembly language.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

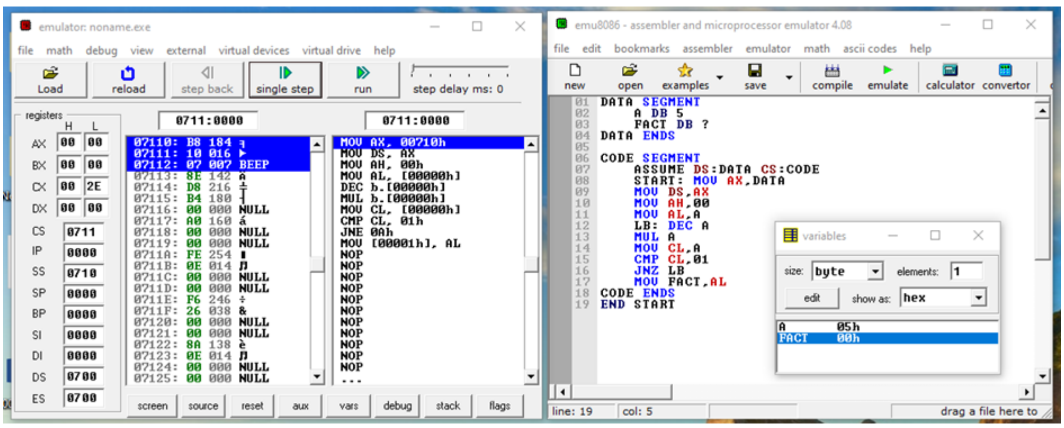
Experiment No: 9

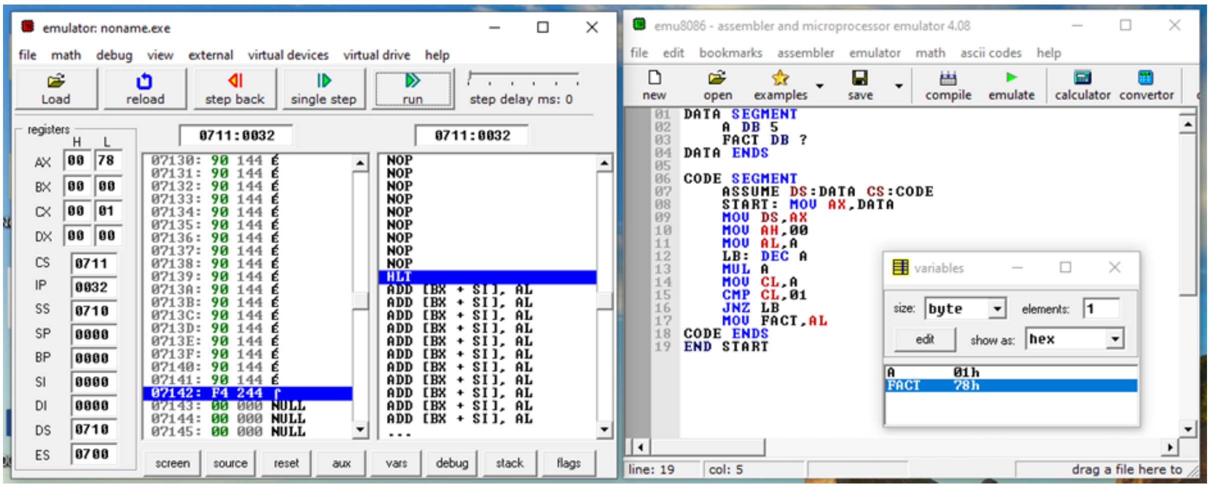
(i) Factorial without macros

Code:

```
DATA SEGMENT
A DB 5
fact DB ?
DATA ENDS
CODE SEGMENT
        ASSUME DS:DATA,CS:CODE
START:
      MOV AX,DATA
      MOV DS,AX
      MOV AH,00
      MOV AL,A
 L1:  DEC A
      MUL A
      MOV CL,A
      CMP CL,01
      JNZ L1
      MOV fact, AL
CODE ENDS
END START
```

Output:





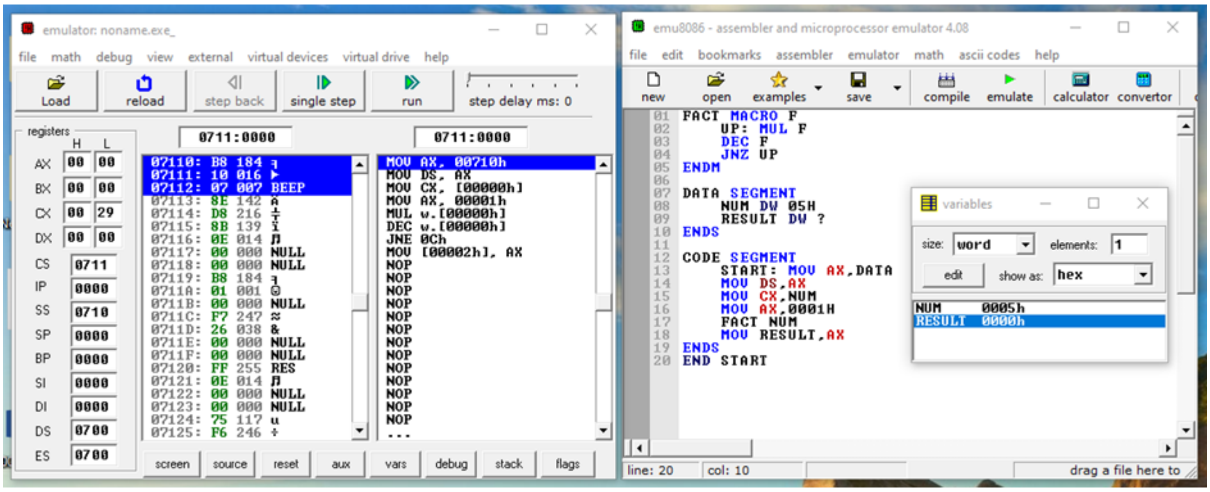
(ii) Factorial with macros

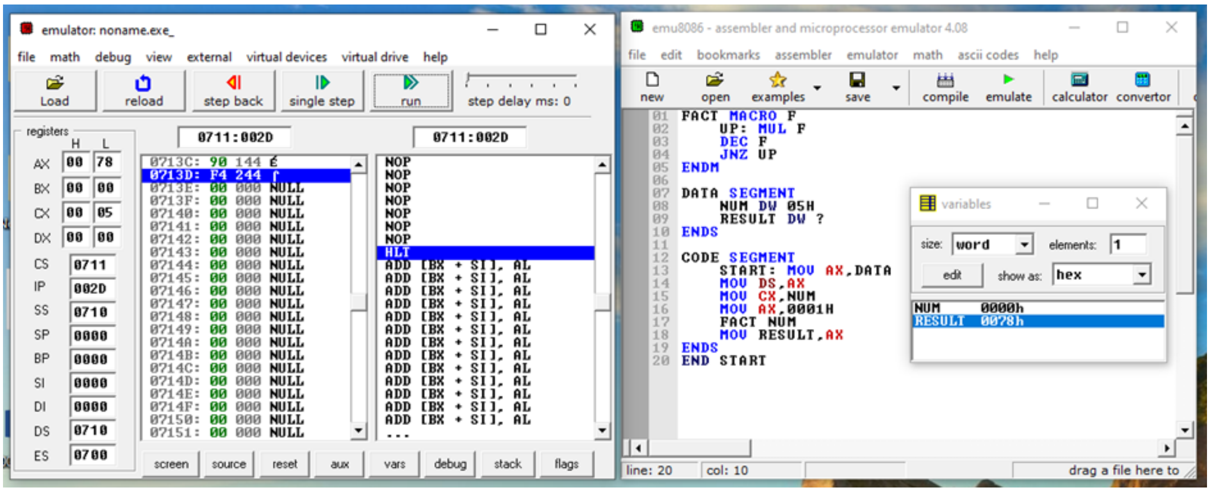
Code:

```
fact macro f
    up:
    mul f
    dec f
    jnz up
endm

data segment
    num dw 05h
    result dw ?
ends

stack segment
    dw 128 dup(0)
ends

code segment
    start:
    mov ax,data
    mov ds,ax
    mov cx,num
```

```
    mov ax, 0001h
    fact num
    mov result,ax
ends
```

Output:





Conclusion: Therefore we have successfully implemented the code to find the factorial of a given number both with and without macros using Assembly Language.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

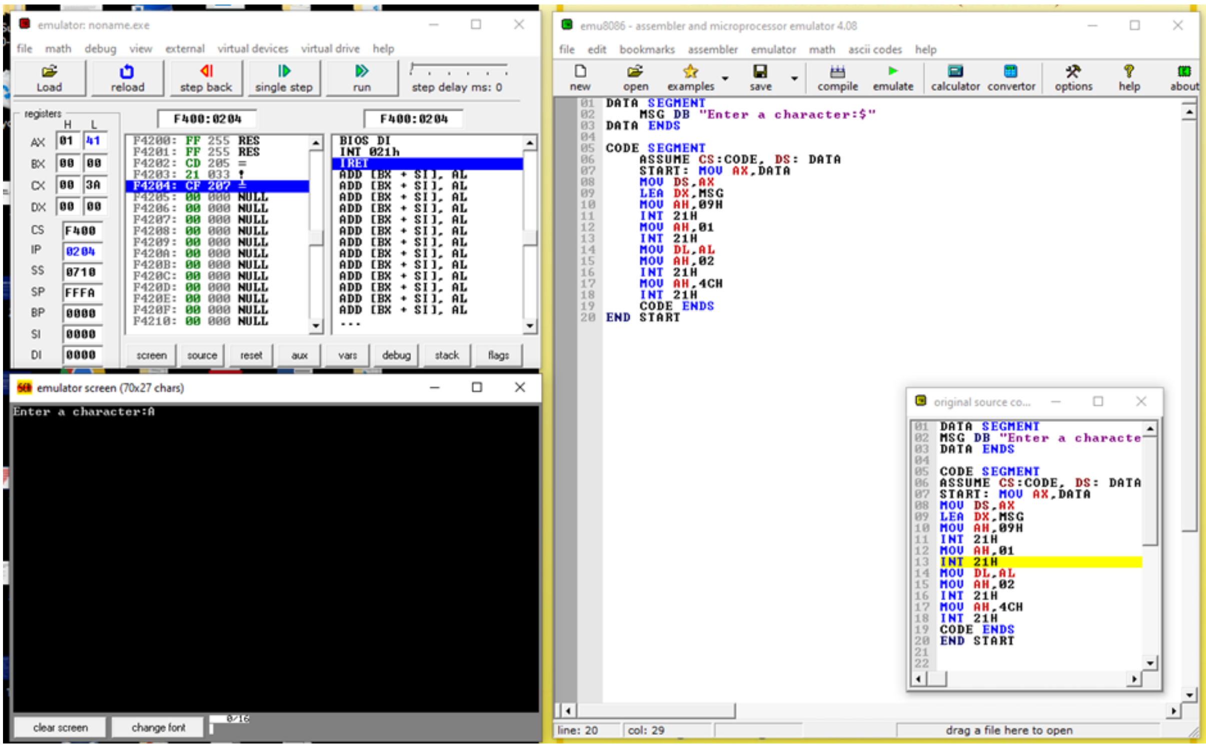
Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**

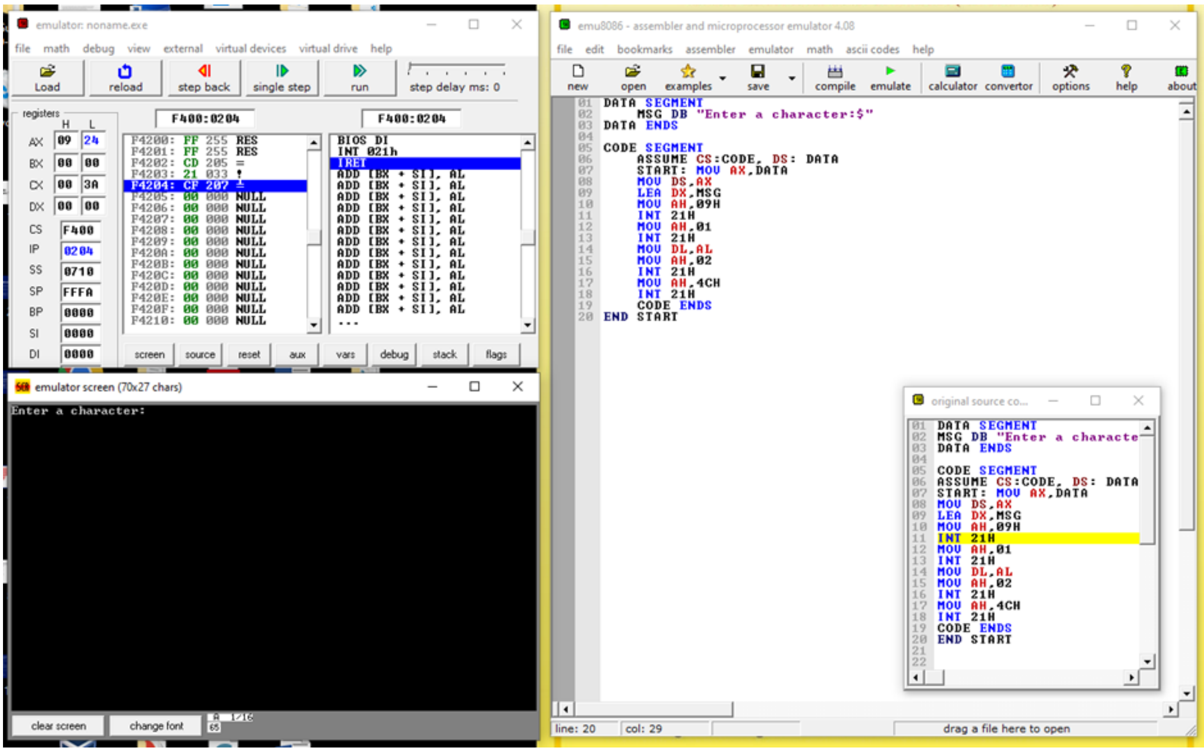
Experiment No: 10

Code:

```
data segment
   MSG DB "Enter a character:$"
   data ends

code segment
   assume cs:code, ds:data
   start:

   mov ax,data
   mov ds,ax
   lea DX,MSG
   MOV AH,09h
   INT 21H

   mov ah,01
   int 21h

   mov dl,al
   mov ah,02
   int 21h

   mov ah,4ch
   int 21h
   code ends
end start
```

Output:

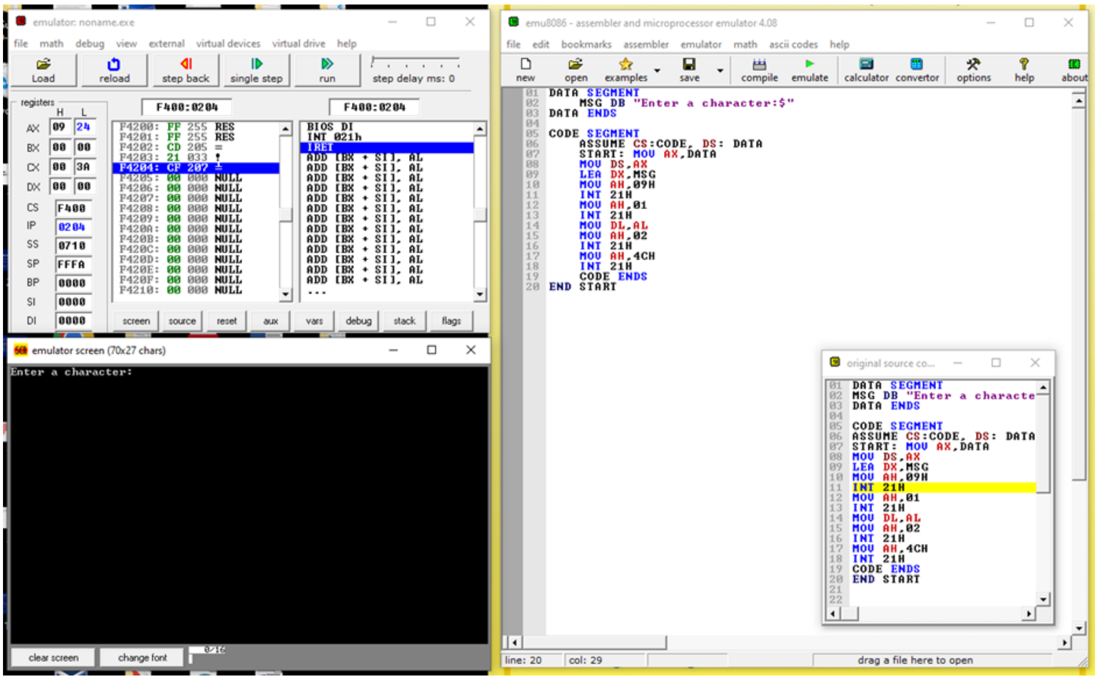

emulator: noname.exe
Enter a character:A
emu8086 - assembler and microprocessor emulator 4.08
DATA SEGMENT
MSG DB "Enter a character:$"
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS: DATA
START: MOV AX,DATA
MOV DS,AX
LEA DX,MSG
MOV AH,09H
INT 21H
MOV AH,01
INT 21H
MOV DL,AL
MOV AH,02
INT 21H
MOV AH,4CH
INT 21H
CODE ENDS
END START

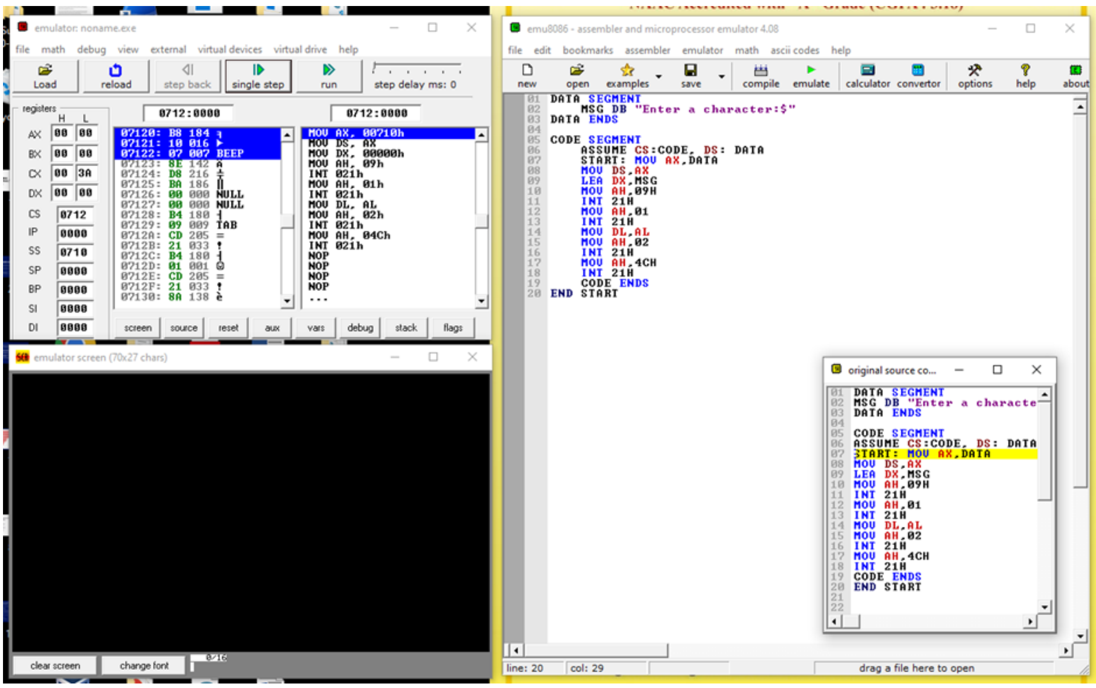

emulator: noname.exe
Enter a character:
emu8086 - assembler and microprocessor emulator 4.08

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJ19CEL502 **Course Name: Processor Organization & Architecture**





Conclusion: Therefore we have successfully implemented the code to show Hardware Interrupt when the user enters a character using a hardware device such as keyboard using Assembly Language.