



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)
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

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

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

count = maxlen
m = binNum1
minusb = 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(minusb,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]
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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:
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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]
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
    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)
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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.



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

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
minusb = binCompNum2
q = binNum1
a = "0"
```




Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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))
```



SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



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
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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
minusb = binCompNum2
q = binNum1
a = "0"
a = a.zfill(maxlen+1)
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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:]
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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";
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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");
    }
}
```




Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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;
    }
}
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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)
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
        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)
{

```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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;
}
```




Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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;
}
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
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
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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++;
            }
        }
    }
}
```



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

```
    }  
    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



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

Output:

registers H L

AX	FF	F6
BX	00	42
CX	00	01
DX	00	00
CS	0700	
IP	0115	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0700:0115

```
07100: 01 161 i MOV AX, [01000h]
07101: 00 000 NULL MOV BX, [01002h]
07102: 10 016 SUB AX, BX
07103: 8B 139 i JNB 0111h
07104: 1E 030 INC CL
07105: 02 002 MOV [01006h], CL
07106: 10 016 MOV [01004h], AX
07107: 2B 043 + HLT
07108: C3 195 r NOP
07109: 73 115 c NOP
0710A: 06 006 + NOP
0710B: FE 254 i NOP
0710C: C1 193 r NOP
0710D: 88 136 e NOP
0710E: 0E 014 j NOP
0710F: 06 006 + NOP
07110: 10 016 + NOP
07111: 89 137 e NOP
07112: 06 006 + NOP
07113: 04 004 + NOP
07114: 10 016 + NOP
07115: F4 244 f ...
```

Random Access Memory

0700:1000	38 00 42 00 F6 FF 01 00-00 00 00 00 00 00 00 00 00	S.B.+..0.....
0700:1010	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1020	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1030	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1040	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1050	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1060	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1070	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

registers H L

AX	00	00
BX <td>00</td> <td>00</td>	00	00
CX <td>00</td> <td>1B</td>	00	1B
DX <td>00</td> <td>00</td>	00	00
CS <td>0700</td> <td></td>	0700	
IP <td>0100</td> <td></td>	0100	
SS <td>0700</td> <td></td>	0700	
SP <td>FFFE</td> <td></td>	FFFE	
BP <td>0000</td> <td></td>	0000	
SI <td>0000</td> <td></td>	0000	
DI <td>0000</td> <td></td>	0000	
DS <td>0700</td> <td></td>	0700	
ES <td>0700</td> <td></td>	0700	

0700:0100

```
07100: 8B 139 i MOV CX, [01000h]
07101: 0E 014 j MOV DX, [01002h]
07102: 00 000 NULL MOV AX, CX
07103: 10 016 + MOV BX, DX
07104: 8B 139 i SUB AX, BX
07105: 16 022 JNB 0116h
07106: 02 002 INC CL
07107: 10 016 + MOV [01006h], CL
07108: 8B 139 i MOV [01004h], AX
07109: C1 193 r HLT
0710A: 8B 139 i NOP
0710B: DA 218 r NOP
0710C: 2B 043 + NOP
0710D: C3 195 r NOP
0710E: 73 115 c NOP
0710F: 06 006 + NOP
07110: FE 254 i NOP
07111: C1 193 r NOP
07112: 88 136 e NOP
07113: 0E 014 j NOP
07114: 06 006 + NOP
07115: 10 016 + ...
```

Random Access Memory

0700:1000	58 00 20 00 38 00 00-00 00 00 00 00 00 00 00 00	X..8.....
0700:1010	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1020	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1030	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1040	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1050	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1060	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1070	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

registers H L

AX	00	69
BX <td>00</td> <td>15</td>	00	15
CX <td>00</td> <td>54</td>	00	54
DX <td>00</td> <td>15</td>	00	15
CS <td>0700</td> <td></td>	0700	
IP <td>011A</td> <td></td>	011A	
SS <td>0700</td> <td></td>	0700	
SP <td>FFFE</td> <td></td>	FFFE	
BP <td>0000</td> <td></td>	0000	
SI <td>0000</td> <td></td>	0000	
DI <td>0000</td> <td></td>	0000	
DS <td>0700</td> <td></td>	0700	
ES <td>0700</td> <td></td>	0700	

0700:011A

```
07116: 89 137 e MOV CX, [01000h]
07117: 06 006 + MOV DX, [01002h]
07118: 04 004 + MOV AX, CX
07119: 10 016 + MOV BX, DX
0711A: F4 244 f RDD AX, BX
0711B: 90 144 e JNB 0116h
0711C: 90 144 e INC CL
0711D: 90 144 e MOV [01006h], CL
0711E: 90 144 e MOV [01004h], AX
0711F: 90 144 e NOP
07120: 90 144 e NOP
07121: 90 144 e NOP
07122: 90 144 e NOP
07123: 90 144 e NOP
07124: 90 144 e NOP
07125: 90 144 e NOP
07126: 90 144 e NOP
07127: 90 144 e NOP
07128: 90 144 e NOP
07129: 90 144 e NOP
0712A: 90 144 e NOP
0712B: 90 144 e ...
```

Random Access Memory

0700:1000	54 00 15 00 69 00 00-00 00 00 00 00 00 00 00 00	T.S.i.....
0700:1010	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1020	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1030	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1040	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1050	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1060	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:1070	00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

VIEW... close run

external...



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

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



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

Output:

The screenshot displays an x86-64 emulator interface. On the left, the 'original source code' window shows assembly instructions. The main window, titled 'emulator: ACSENDING.exe', shows the execution state. The registers window on the right lists registers AX through DS, with their high (H) and low (L) bytes. The instruction window shows the current instruction being executed, which is 'HLT'. The flags window on the right shows the status of various flags. The variables window at the bottom shows the memory address 00000000 and the value 15h, 23h, 43h, 55h, 86h.

```
01 DATA SEGMENT
02 STRING1 DB 23H, 43H, 15H, 55H, 86H
03 DATA ENDS
04
05 CODE SEGMENT
06 ASSUME CS:CODE, DS:DATA
07
08 START:
09 MOV AX,DATA
10 MOV DS,AX
11 MOV CH,04H
12
13 UP2: MOV CL,04H
14 LEA SI,STRING1
15
16 UP1: MOV AL,[SI]
17 MOV BL,[SI+1]
18 CMP AL,BL
19 JC DOWN
20 MOV DL,[SI+1]
21 XCHG [SI],DL
22 MOV [SI+1],DL
23
24 DOWN: INC SI
25 DEC CL
26 JNZ UP1
27 DEC CH
28 JNZ UP2
29
30 CODE ENDS
31 END START
32
33
```

emulator: ACSENDING.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	07	55
BX	00	86
CX	00	00
DX	00	23
CS	0711	
IP	003A	
SS	0710	
SP	0000	
BP	0000	
SI	0004	
DI	0000	
DS	0710	
ES	0700	

0711:003A 0711:003A

Address	Instruction
07148	90 144 E
07149	90 144 E
0714A	F4 244 F
0714B	00 000 NULL
0714C	00 000 NULL
0714D	00 000 NULL
0714E	00 000 NULL
0714F	00 000 NULL
07150	00 000 NULL
07151	00 000 NULL
07152	00 000 NULL
07153	00 000 NULL
07154	00 000 NULL
07155	00 000 NULL
07156	00 000 NULL
07157	00 000 NULL
07158	00 000 NULL
07159	00 000 NULL
0715A	00 000 NULL
0715B	00 000 NULL
0715C	00 000 NULL
0715D	00 000 NULL

screen source reset aux vars debug stack flags analyse

variables

size: byte elements: 5

edit show as: hex

STRING1 15h, 23h, 43h, 55h, 86h



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

(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:

The screenshot displays an x86 emulator interface. On the left, the 'original source code' window shows assembly instructions. The 'emulator: ACSENDING.exe' window on the right shows the state of registers, memory, and flags.

original source code

```
01 DATA SEGMENT
02 STRING1 DB 23H, 43H, 15H, 55H, 86H
03 DATA ENDS
04
05 CODE SEGMENT
06 ASSUME CS:CODE, DS:DATA
07
08 START:
09 MOV AX,DATA
10 MOV DS,AX
11 MOV CH,04H
12
13 UP2: MOV CL,04H
14 LEA SI,STRING1
15
16 UP1: MOV AL,[SI]
17 MOV BL,[SI+1]
18 CMP AL,BL
19 JNC DOWN
20 MOV DL,[SI+1]
21 XCHG [SI],DL
22 MOV [SI+1],DL
23
24 DOWN: INC SI
25 DEC CL
26 JNZ UP1
27 DEC CH
28 JNZ UP2
29
30 CODE ENDS
31 END START
32
33
```

emulator: ACSENDING.exe

Registers:

Register	H	L	Value
AX	07	23	07148: 90 144 6
BX	00	15	07148: 90 144 6
CX	00	00	07148: 00 000 NULL
DX	00	55	07148: 00 000 NULL
CS	0711		07148: 00 000 NULL
IP	003A		07148: 00 000 NULL
SS	0710		07148: 00 000 NULL
SP	0000		07148: 00 000 NULL
BP	0000		07148: 00 000 NULL
SI	0004		07148: 00 000 NULL
DI	0000		07148: 00 000 NULL
DS	0710		07148: 00 000 NULL
ES	0700		07148: 00 000 NULL

Flags:

Flag	Value
CF	0
ZF	1
SF	0
OF	0
PF	1
AF	0
IF	1
DF	0

Variables:

size: byte elements: 5

edit show as: hex

STRING1 86h, 55h, 43h, 23h, 15h

Conclusion: 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.
```




**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

Output:

The screenshot displays the emu8086 emulator interface. The main window shows the assembly code being executed, with the following code visible:

```
01 DATA SEGMENT
02 N1 DB 63H, 72H, 10H
03 DATA ENDS
04
05 EXTRA SEGMENT
06 N2 DB ?
07 EXTRA ENDS
08
09 CODE SEGMENT
10 START:
11
12 MOV AX, DATA
13 MOV DS, AX
14
15 MOV AX, EXTRA
16 MOV ES, AX
17
18 LEA SI, N1
19 LEA DI, N2
20
21 MOV CL, 04H
22
23 L1:
24
25 MOV AX, DS:[SI]
26 MOV ES:[DI], AX
27
28 INC SI
29 INC DI
30 DEC CL
31 DEC CL
32
33 JNZ L1
34
35 CODE ENDS
36 END START
```

The registers window shows the following values:

Register	Value
AX	00 00
BX	00 00
CX	00 00
DX	00 00
SI	0004
DI	0004
DS	0710
ES	0711

The stack window shows the following values:

Address	Value
0710:002A	00BE
0710:0028	C0BE
0710:0026	0711
0710:0024	08D8
0710:0022	8E97
0710:0020	10B8
0710:001E	0000
0710:001C	0000
0710:001A	0000
0710:0018	0000
0710:0016	0000
0710:0014	0000
0710:0012	0018
0710:0010	7263
0710:000E	0000
0710:000C	0000
0710:000A	0000
0710:0008	0000
0710:0006	0000
0710:0004	0000
0710:0002	0018
0710:0000	7263

The variables window shows the following values:

Variable	Value
N1	63h
N2	63h

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:



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

INC SI

LOOP REPEAT

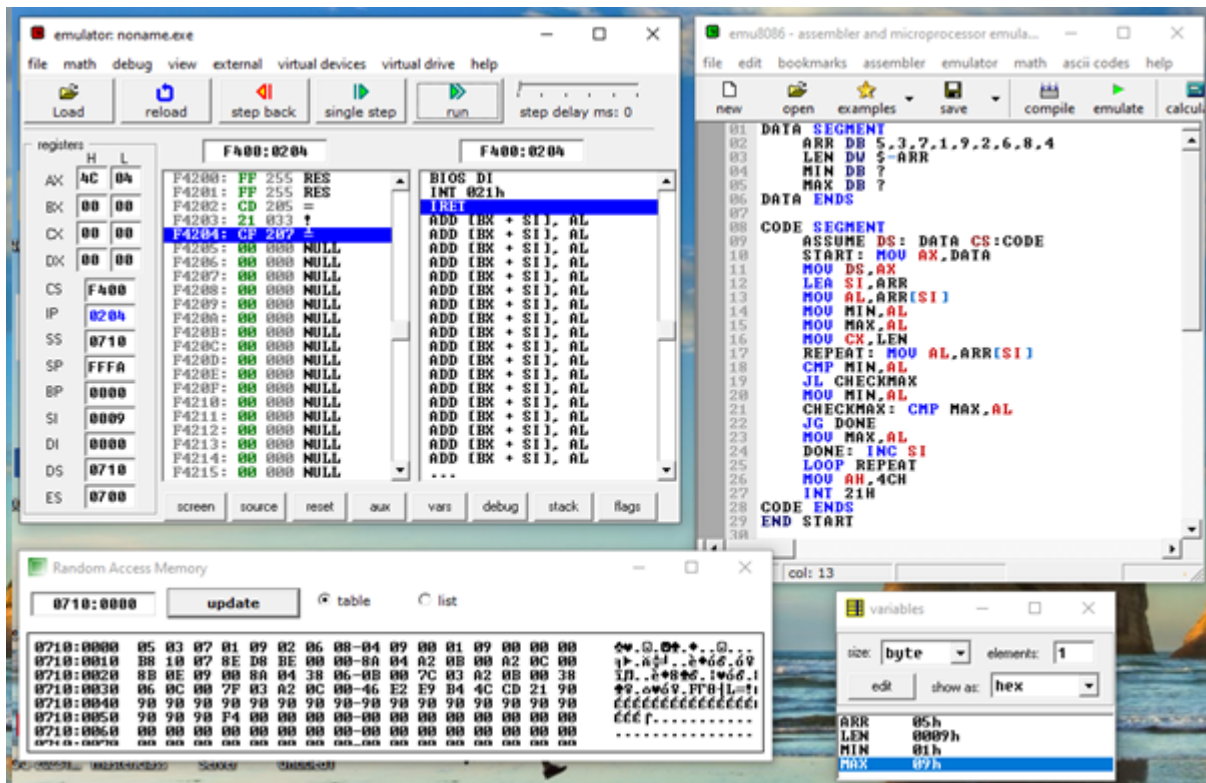
MOV AH,4CH

INT 21H

CODE ENDS

END START

Output:





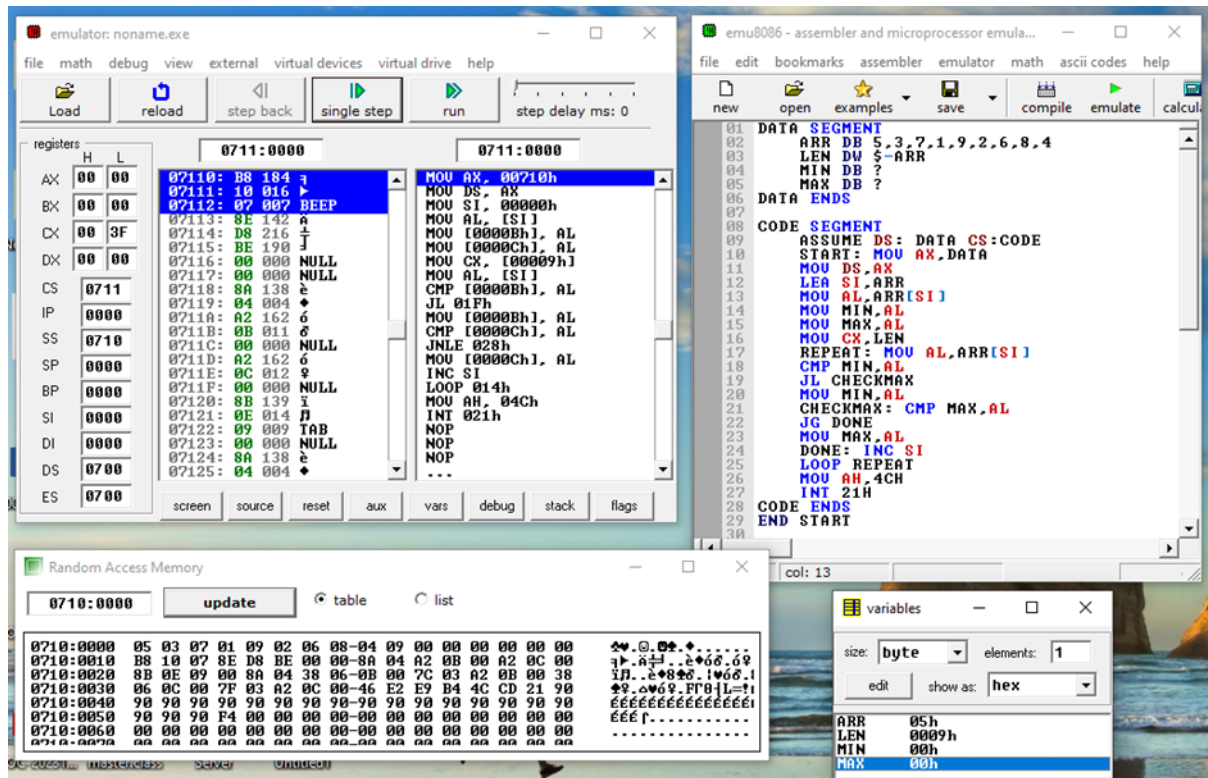
Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture



Conclusion: Therefore we have successfully implemented the code to 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

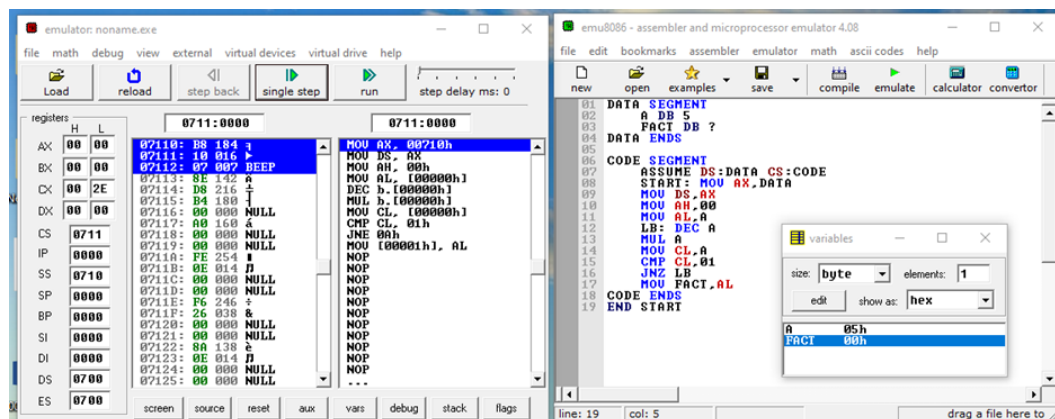
JNZ L1

MOV fact, AL

CODE ENDS

END START

Output:





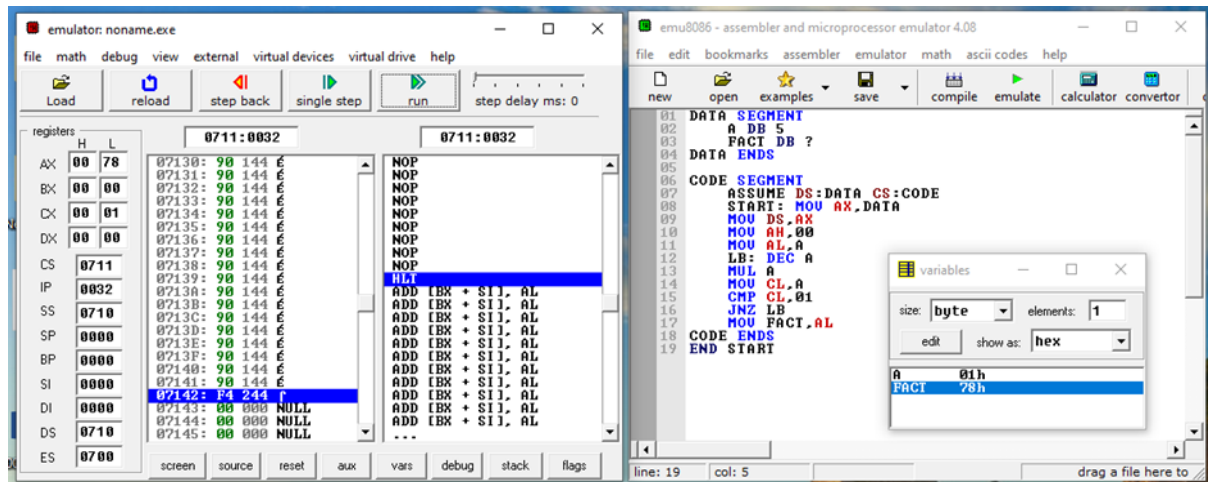
Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture



(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



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

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

Output:

emulator: noname.exe

em8086 - assembler and microprocessor emulator 4.08

```
01 FACT MACRO F
02 UP: MUL F
03 DEC F
04 JNZ UP
05 ENDM
06
07 DATA SEGMENT
08 NUM DW 05H
09 RESULT DW ?
10 ENDS
11
12 CODE SEGMENT
13 START: MOV AX, DATA
14 MOV DS, AX
15 MOV CX, NUM
16 MOV AX, 0001H
17 FACT NUM
18 MOV RESULT, AX
19 ENDS
20 END START
```

emulator: noname.exe

em8086 - assembler and microprocessor emulator 4.08

```
01 FACT MACRO F
02 UP: MUL F
03 DEC F
04 JNZ UP
05 ENDM
06
07 DATA SEGMENT
08 NUM DW 05H
09 RESULT DW ?
10 ENDS
11
12 CODE SEGMENT
13 START: MOV AX, DATA
14 MOV DS, AX
15 MOV CX, NUM
16 MOV AX, 0001H
17 FACT NUM
18 MOV RESULT, AX
19 ENDS
20 END START
```

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



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



Department of Computer Engineering

Class: T.Y. B.Tech.

Semester: V

Course Code: DJ19CEL502

Course Name: Processor Organization & Architecture

Output:

The screenshot shows the emu8086 emulator interface. The main window displays assembly code with the following segments:

```
01 DATA SEGMENT
02 MSG DB "Enter a character: $"
03 DATA ENDS
04
05 CODE SEGMENT
06 ASSUME CS:CODE, DS: DATA
07 START: MOV AX, DATA
08 MOV DS, AX
09 LEA DX, MSG
10 MOV AH, 09H
11 INT 21H
12 MOV AH, 01H
13 INT 21H
14 MOV DL, AL
15 MOV AH, 02H
16 INT 21H
17 MOV AH, 4CH
18 INT 21H
19 CODE ENDS
20 END START
```

The registers window shows the following values:

Register	Value
AX	01 41
BX	00 00
CX	00 3A
DX	00 00
SI	0000
DI	0000

The emulator screen shows the prompt "Enter a character: \$" and the user has entered 'a'.

This screenshot is identical to the one above, showing the same assembly code and register values in the emu8086 emulator.



**SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING**
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)



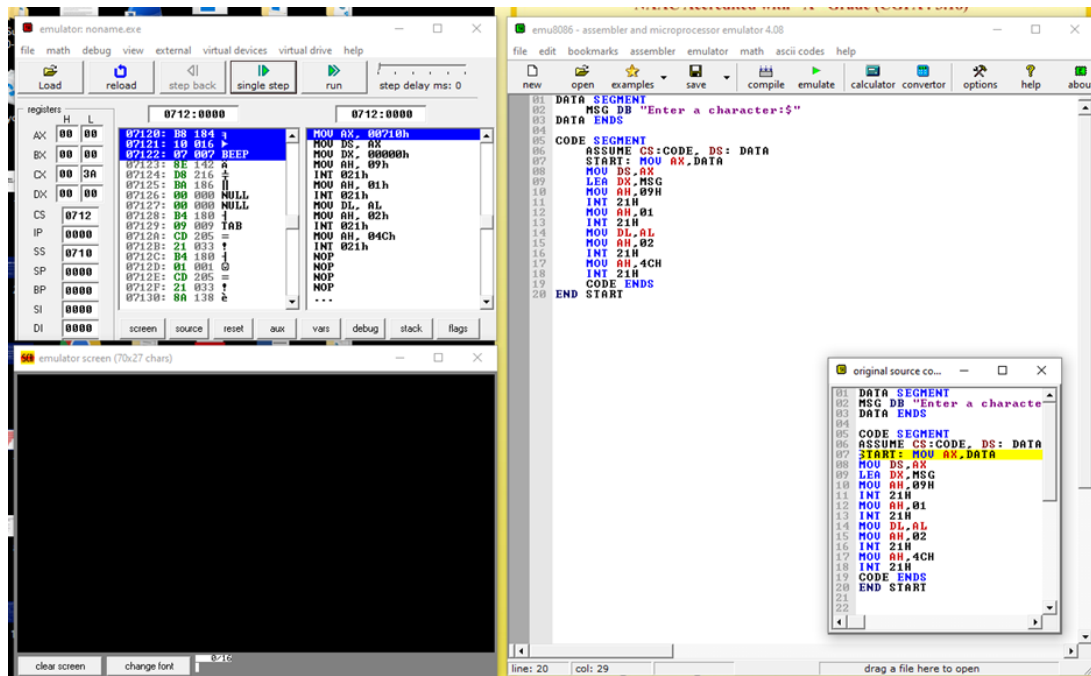
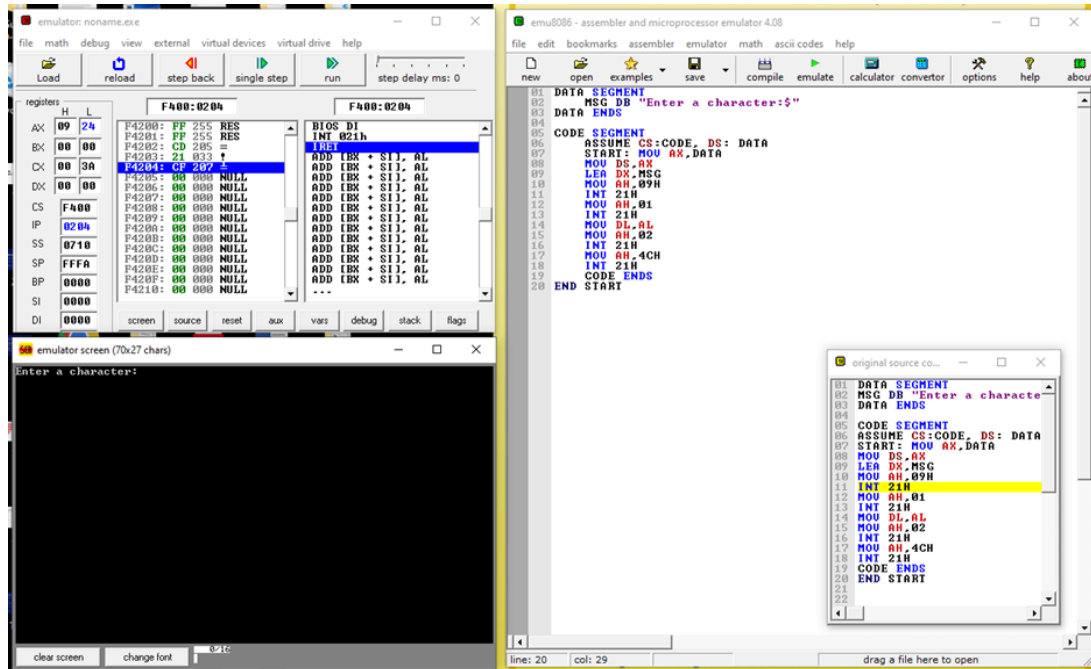
Department of Computer Engineering

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.