**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No: 1**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Study of PCI and SCSI. |

**AIM: To Study and learn PCI and SCSI**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment : (Mention CO/CO’s attained here )**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Books/ Journals/ Websites referred:**

1. [**https://www.techopedia.com/definition/8815/peripheral-component-interconnect-bus-pci-bus**](https://www.techopedia.com/definition/8815/peripheral-component-interconnect-bus-pci-bus)
2. [**https://www.techopedia.com/definition/331/small-computer-system-interface-scsi**](https://www.techopedia.com/definition/331/small-computer-system-interface-scsi)
3. [**http://www.csun.edu/~edaasic/roosta/BUS\_Structures.pdf**](http://www.csun.edu/~edaasic/roosta/BUS_Structures.pdf)
4. W.Stallings William “Computer Organization and Architecture: Designing for Performance”, Pearson Prentice Hall Publication, 7thEdition. C.

**Pre Lab/ Prior Concepts:**

Microcomputer buses which communicate with a peripheral devices or a memory location through communication lines called buses.

The major parts of microcomputers are central processing unit (CPU), memory, and input and output unit. To connect these parts together through three sets of parallel lines, called buses.  These three buses are  Address bus, data bus, and Control bus.

**Address Bus:**

The address bus consists of 16, 20, 24, or more parallel signal lines, through which the CPU sends out the address of the memory location. This memory location is used for to written to or read from. The number of memory location is depends on 2 to the power N address lines.  Example, a CPU with 16 address lines can address 216 or 65,536 memory locations. When the CPU reads data from or writes data to a port. The port address is also sent out on the address bus. This is unidirectional. This means that the CPU can send data to a memory location or I/O ports.

**Data Bus:**

The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bidirectional. This means that the CPU can read data from memory or from a I/O port as well as send data to a memory location or to a I/O port. In a system, many output devices are connected to the data bus, but only one device at a time will be enabled to the output.

**Control Bus:**

The control bus consists of 4-10 parallel signal lines. The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices. Typically control bus signals are memory read, memory write, I/O read and I/O write. To read a data from a memory location, the CPU sends out the address of the desired data on the address bus and then sends out a memory read signal on the control bus. The memory read signal enables the addressed memory device to output the data onto the data bus where it is read by the CPU.

**PCI Bus**

PCI stands for Peripheral Component Interconnect. It could be a standard information transport that was common in computers from 1993 to 2007 or so. It was for a long time the standard transport for extension cards in computers, like sound cards, network cards, etc. It was a parallel transport, that, in its most common shape, had a clock speed of 66 MHz, and can either be 32 or 64 bits wide. It has since been replaced by PCI Express, which could be a serial transport as contradicted to PCI. A PCI port, or, more precisely, PCI opening, is essentially the connector that’s utilized to put through the card to the transport. When purge, it basically sits there and does nothing.

Types of PCI :

These are various types of PCI:

* PCI 32 bits have a transport speed of 33 MHz and work at 132 MBps.
* PCI 64 bits have a transport speed of 33 MHz and work at 264 MBps.
* PCI 64 bits have a transport speed of 66 MHz and work at 512 MBps.
* PCI 64 bits have a transport speed of 66 MHz and work at 1 GBps.

**Function of PCI :**

PCI slots are utilized to install sound cards, Ethernet and remote cards and presently strong state drives utilizing NVMe innovation to supply SSD drive speeds that are numerous times speedier than SATA SSD speeds. PCI openings too permit discrete design cards to be included to a computer as well.

PCI openings (and their variations) permit you to include expansion cards to a motherboard. The extension cards increment the machines capabilities past what the motherboard may create alone, such as: upgraded illustrations, extended sound, expanded USB and difficult drive controller, and extra arrange interface options, to title a couple of.

**Advantage of PCI :**

* You’ll interface a greatest of five components to the PCI and you’ll be able moreover supplant each of them by settled gadgets on the motherboard.
* You have different PCI buses on the same computer.
* The PCI transport will improve the speed of the exchanges from 33MHz to 133 MHz with a transfer rate of 1 gigabyte per second.
* The PCI can handle gadgets employing a greatest of 5 volts and the pins utilized can exchange more that one flag through one stick.

**Disadvantage of PCI :**

* PCI Graphics Card cannot get to to system memory.
* PCI does not support pipeline.

**SCSI bus:**

The basic interface for connecting peripheral devices to a PC is a small computer system interface. Based on the specification, it can typically respond up to 16 external devices using a single route, along with a host adapter. Small Computer System Interface is used to boost performance, deliver fast data transfer delivery and provide wider expansion for machines like CD-ROM drivers, scanners, DVD> drives and CD writers. Small Computer System Interface is most commonly used for RAID, servers, highly efficient desktop computers, and storage area networks. The Small Computer System Interface has control, which is responsible for transmitting data across the Small Computer System Interface bus and the computers. It can be fixed on a motherboard, or one client adapter is installed through an extension on the computer's motherboard. The controller also incorporates a simple SCSI input/output system, which is a small chip that provides access and control equipment with the necessary software. The SCSI ID is his number. Using serial storage architecture initiators, new serial SCSI IDs such as serial attached SCSI use an automatic process which assigns a 7-bit number.

**Post Lab Descriptive Questions**

**Q1 . Differentiate between PCI and SCSI Bus**

|  |  |
| --- | --- |
| **PCI** | **SCSI** |
| Peripheral Component Interconnect (PCI), as its name implies is a standard that describes how to connect the peripheral components of a system together in a structured and controlled way. | SCSI is standard electronic interfaces that allow personal computers to communicate with peripheral hardware such as disk drives, tape drives etc. |
| PCI bus was created by Intel in 1993. PCI bus can transfer 32 or 64 bits at one time. PCI bus can run at 33 Mhz. | It is a high performance bus which is used for fast disks, scanners, and for devices which require high bandwidth. It has a data rate of 160 MB/s. |
| Typical bandwidth is 80 m/s | Typical bandwidth is 1.5 to 40 m/s |
| Bus type is Backplane | Bus type is I/O |

**Q2. List two applications each of PCI and SCSI Bus**

**Applications for PCI are:**

* Designed for multiprocessor system and high performing peripheral. This includes audio, video system, network adapters, graphics and accelerator board, data storage collectors
* Because each PCI design is unique, programmable logic devices provide an ideal solution for PCI design
* PCI has replaced ISA as the bus of choice in new desktop and industrial PCs. With the PCI bus, high-performance, low-cost, and convenient PCI based image processing and data acquisition solutions are at hand.

**Applications for SCSI are:**

* SCSI is a multi-task interface with bus arbitration function. Multiple peripherals hung on one SCSI bus can work simultaneously
* SCSI devices have equal possession of the bus.
* SCSI interface can transmit data synchronously and asynchronously. The synchronous transmission rate reaches 10MB/s, and the asynchronously transmission rate reaches 1.5MB/s.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No: 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** To study and implement Booth’s Multiplication Algorithm. |

**AIM:** Booth’s Algorithm for Multiplication

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

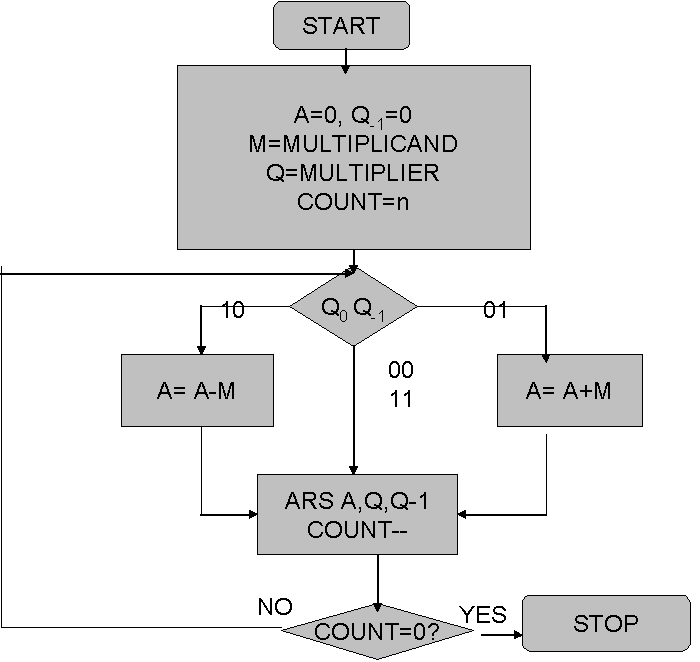
3. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

It is a powerful algorithm for signed number multiplication which generates a 2n bit product and treats both positive and negative numbers uniformly. Also the efficiency of the algorithm is good due to the fact that, block of 1’s and 0’s are skipped over and subtraction/addition is only done if pair contains 10 or 01

**Flowchart:**

****

**Design Steps**:

1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q-1 =0
4. Convert M and Q into binar
5. Compare Q0 andQ-1 and perform the respective operation.

|  |  |
| --- | --- |
| **Q0 Q-1** | **Operation** |
| 00/11 | Arithmetic right shift |
| 01 | A+M and Arithmetic right shift |
| 10 | A-M and Arithmetic right shift |

6. Repeat steps 5 till all bits are compared

7. Convert the result to decimal form and display

8. End

Example: (Handwritten solved problem needs to be uploaded) Table

Description automatically generated

Graphical user interface, table

Description automatically generated

**Code:**

#include <bits/stdc++.h>

using namespace std;

int findbit(int m,int q){

    m=max(abs(m),abs(q));

    for(q=0;pow(2,q)<m;q++);

    return (max((q+1),4));

}

int\* binary(int a,int num){

    int\* ptr=(int\*)malloc(num\*sizeof(int));

    int acopy=abs(a),check=1;

    for (int i = 0;i<num; i++){

        ptr[i] = acopy % 2;

        acopy = acopy/2;

    }

    if (a < 0){

        for (int i = 0; i <num; i++){

            if (ptr[i] == 1 && check==1)

                check=0;

            else if(ptr[i] == 1 && check==0)

                ptr[i]=0;

            else if(ptr[i] == 0 && check==0)

                ptr[i]=1;

        }

    }

    return ptr;

}

void printbinary(int\* ans,string s,int num){

    for(int i=2\*num;i>num;i--)

        cout<<ans[i]<<" ";

    cout<<"\t";

    for(int i=num;i>0;i--)

        cout<<ans[i]<<" ";

    cout<<"\t"<<ans[0]<<"\t"<<s<<endl;

}

void binaryadd(int\* ans,int\* n,int num){

    int carry=0;

    for(int i=num+1;i<=2\*num;i++){

        if(ans[i]+n[i-num-1]+carry==1){

            ans[i]=1;

            carry=0;

        }

        else if(ans[i]+n[i-num-1]+carry==2){

            ans[i]=0;

            carry=1;

        }

        else if(ans[i]+n[i-num-1]+carry==3){

            ans[i]=1;

            carry=1;

        }

    }

}

int main()

{

    int m,q;

    cout<<"Enter M and Q: ";

    cin>>m>>q;

    int num=findbit(m,q);

    int ans[2\*num+1]={0};

    int \*arr=binary(q,num);

    for(int i=num;i>0;i--)

        ans[i]=arr[i-1];

    cout<<endl<<"A\t\tQ\t\tQ-1\t\tOperation"<<endl<<endl;

    printbinary(ans,"Initial Value",num);

    for(int i=0;i<num;i++){

        if(ans[1]==0 && ans[0]==1){

            binaryadd(ans,binary(m,num),num);

            printbinary(ans,"A <- A + M",num);

        }

        else if(ans[1]==1 && ans[0]==0){

            binaryadd(ans,binary(-m,num),num);

            printbinary(ans,"A <- A - M",num);

        }

        for(int i=0;i<2\*num;i++)

            ans[i]=ans[i+1]; // Right Shifting

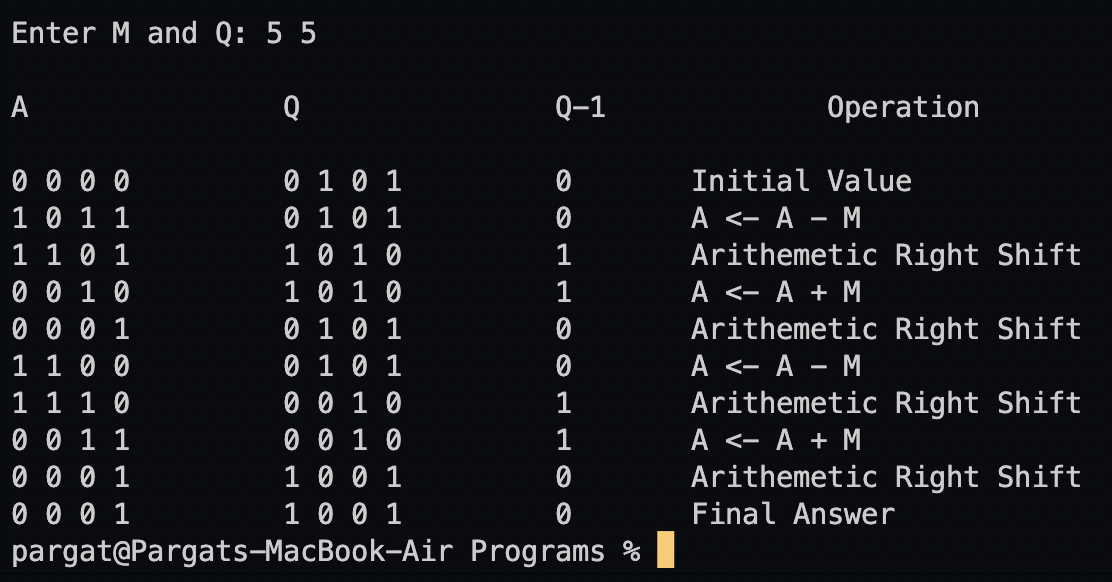
        printbinary(ans,"Arithemetic Right Shift",num);

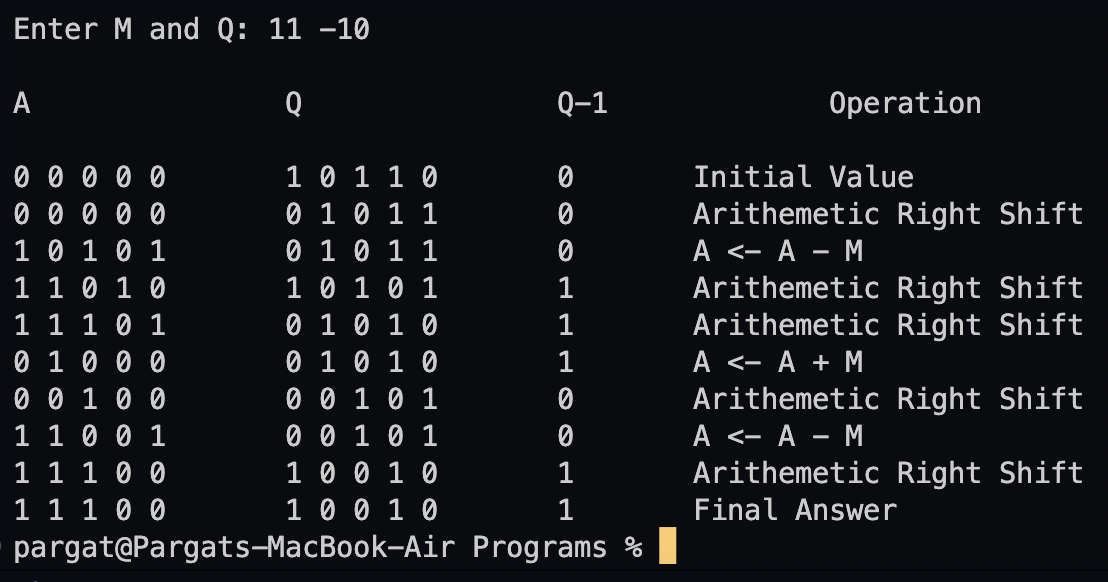
    }

    printbinary(ans,"Final Answer",num);

}

**Output:**





**Conclusion:**

Learnt and implemented booths algorithm along with the understanding of computer bits and operations like arithmetic shift right.

**Post Lab Descriptive Questions**

**1. Explain advantages and disadvantages of Booth’s algorithm.**

Advantages of booth's multiplication:

* Easy calculation of multiplication problem.
* Consecutive additions will be replaced.
* Less complex and ease scaling.

Disadvantages of booth's multiplication:

* This algorithm will not work for isolated 1's.
* It is time consuming.
* If digital gates are more, chip area would be large.

1. **Is Booth’s recoding better than Booth’s algorithm? Justify**

Advantage of Booth’s recoding is that it reduces the number of 1’s and increases the number of 0’s in a binary number. Having more number of 0’s is advantageous for easier calculation.

For Example: (01111)2 is equivalent to (+1 0 0 0 -1) in Booth Recoding. Hence it is more efficient and less time consuming in comparison to Booth’s algorithm.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No: 3**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :** To study and implement Restoring method of division |

**AIM :** The basis of algorithm is based on paper and pencil approach and the operation involves repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO /CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

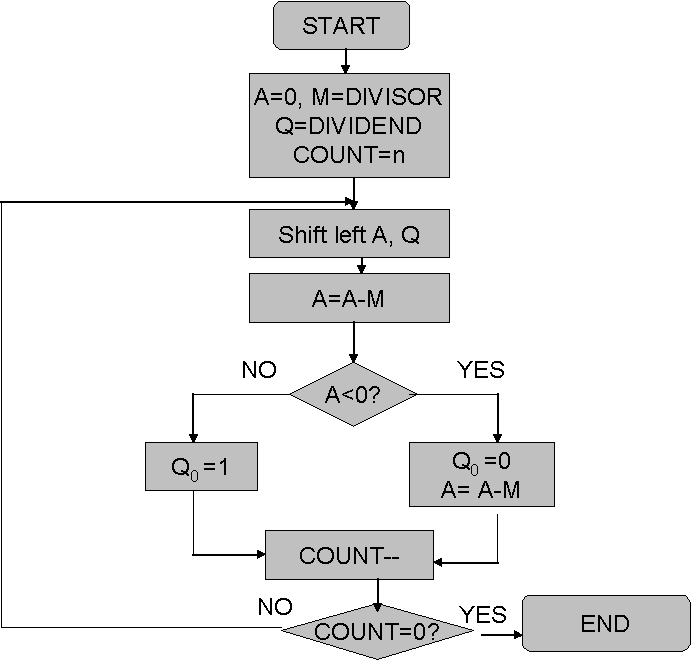
**3**. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

The Restoring algorithm works with any combination of positive and negative numbers.

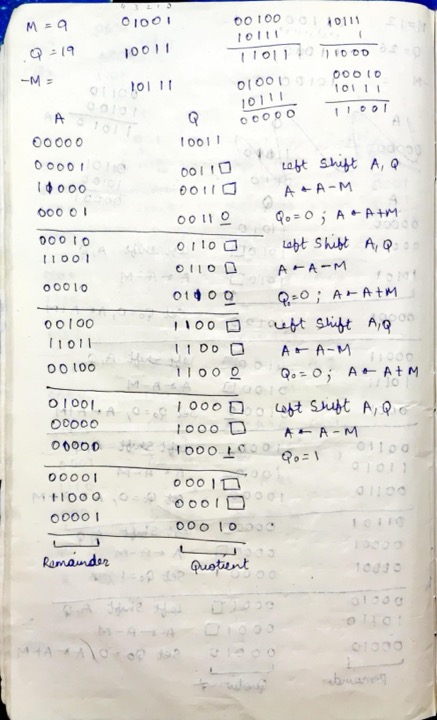
**Flowchart for Restoring of Division:**



**Design Steps**:

1. Start
2. Initialize A=0, M=Divisor, Q=Dividend and count=n (no of bits)
3. Left shift A, Q
4. If MSB of A and M are same
5. Then A=A-M
6. Else A=A+M
7. If MSB of previous A and present A are same
8. Q0=0 & store present A
9. Else Q0=0 & restore previous A
10. Decrement count.
11. If count=0 go to 11
12. Else go to 3
13. STOP

**Example:- (Handwritten solved problems needs to be uploaded)**



Text, letter

Description automatically generated

**Code:**

*#include* <stdio.h>

*#include* <stdlib.h>

int dec\_bin(int, int []);

int twos(int [], int []);

int left(int [], int []);

int add(int [], int []);

int main()

{

    int a, b, m[4]={0,0,0,0}, q[4]={0,0,0,0}, acc[4]={0,0,0,0}, m2[4], i, n=4;

    printf("Enter the Dividend: ");

    scanf("%d", &a);

    printf("Enter the Divisor: ");

    scanf("%d", &b);

    dec\_bin(a, q);

    dec\_bin(b, m);

    twos(m, m2);

    printf("\nA\tQ\tComments\n");

*for*(i=3; i>=0; i--)

    {

        printf("%d", acc[i]);

    }

    printf("\t");

*for*(i=3; i>=0; i--)

    {

        printf("%d", q[i]);

    }

    printf("\tStart\n");

*while*(n>0)

    {

        left(acc, q);

*for*(i=3; i>=0; i--)

        {

            printf("%d", acc[i]);

        }

        printf("\t");

*for*(i=3; i>=1; i--)

        {

            printf("%d", q[i]);

        }

        printf("\_\tLeft Shift A,Q\n");

        add(acc, m2);

*for*(i=3; i>=0; i--)

        {

            printf("%d", acc[i]);

        }

        printf("\t");

*for*(i=3; i>=1; i--)

        {

            printf("%d", q[i]);

        }

        printf("\_\tA=A-M\n");

*if*(acc[3]==0)

        {

            q[0]=1;

*for*(i=3; i>=0; i--)

            {

                printf("%d", acc[i]);

            }

            printf("\t");

*for*(i=3; i>=0; i--)

            {

                printf("%d", q[i]);

            }

            printf("\tQo=1\n");

        }

*else*

        {

            q[0]=0;

            add(acc, m);

*for*(i=3; i>=0; i--)

            {

                printf("%d", acc[i]);

            }

            printf("\t");

*for*(i=3; i>=0; i--)

            {

                printf("%d", q[i]);

            }

            printf("\tQo=0; A=A+M\n");

        }

        n--;

    }

    printf("\nQuotient = ");

*for*(i=3; i>=0; i--)

    {

            printf("%d", q[i]);

    }

    printf("\tRemainder = ");

*for*(i=3; i>=0; i--)

    {

            printf("%d", acc[i]);

    }

    printf("\n");

*return* 0;

}

int dec\_bin(int *d*, int *m*[])

{

    int b=0, i=0;

*for*(i=0; i<4; i++)

    {

        m[i]=d%2;

        d=d/2;

    }

*return* 0;

}

int twos(int *m*[], int *m2*[])

{

    int i, m1[4];

*for*(i=0; i<4; i++)

    {

*if*(m[i]==0)

        {

            m1[i]=1;

        }

*else*

        {

            m1[i]=0;

        }

    }

*for*(i=0; i<4; i++)

    {

        m2[i]=m1[i];

    }

*if*(m2[0]==0)

    {

        m2[0]=1;

    }

*else*

    {

        m2[0]=0;

*if*(m2[1]==0)

        {

            m2[1]=1;

        }

*else*

        {

            m2[1]=0;

*if*(m2[2]==0)

            {

                m2[2]=1;

            }

*else*

            {

                m2[2]=0;

*if*(m2[3]==0)

                {

                  m2[3]=1;

                }

*else*

                {

                  m2[3]=0;

                }

            }

        }

    }

*return* 0;

}

int left(int *acc*[], int *q*[])

{

    int i;

*for*(i=3; i>0; i--)

    {

        acc[i]=acc[i-1];

    }

    acc[0]=q[3];

*for*(i=3; i>0; i--)

    {

        q[i]=q[i-1];

    }

}

int add(int *acc*[], int *m*[])

{

  int i, carry=0;

*for*(i=0; i<4; i++)

  {

*if*(acc[i]+m[i]+carry==0)

    {

      acc[i]=0;

      carry=0;

    }

*else* *if*(acc[i]+m[i]+carry==1)

    {

      acc[i]=1;

      carry=0;

    }

*else* *if*(acc[i]+m[i]+carry==2)

    {

      acc[i]=0;

      carry=1;

    }

*else* *if*(acc[i]+m[i]+carry==3)

    {

      acc[i]=1;

      carry=1;

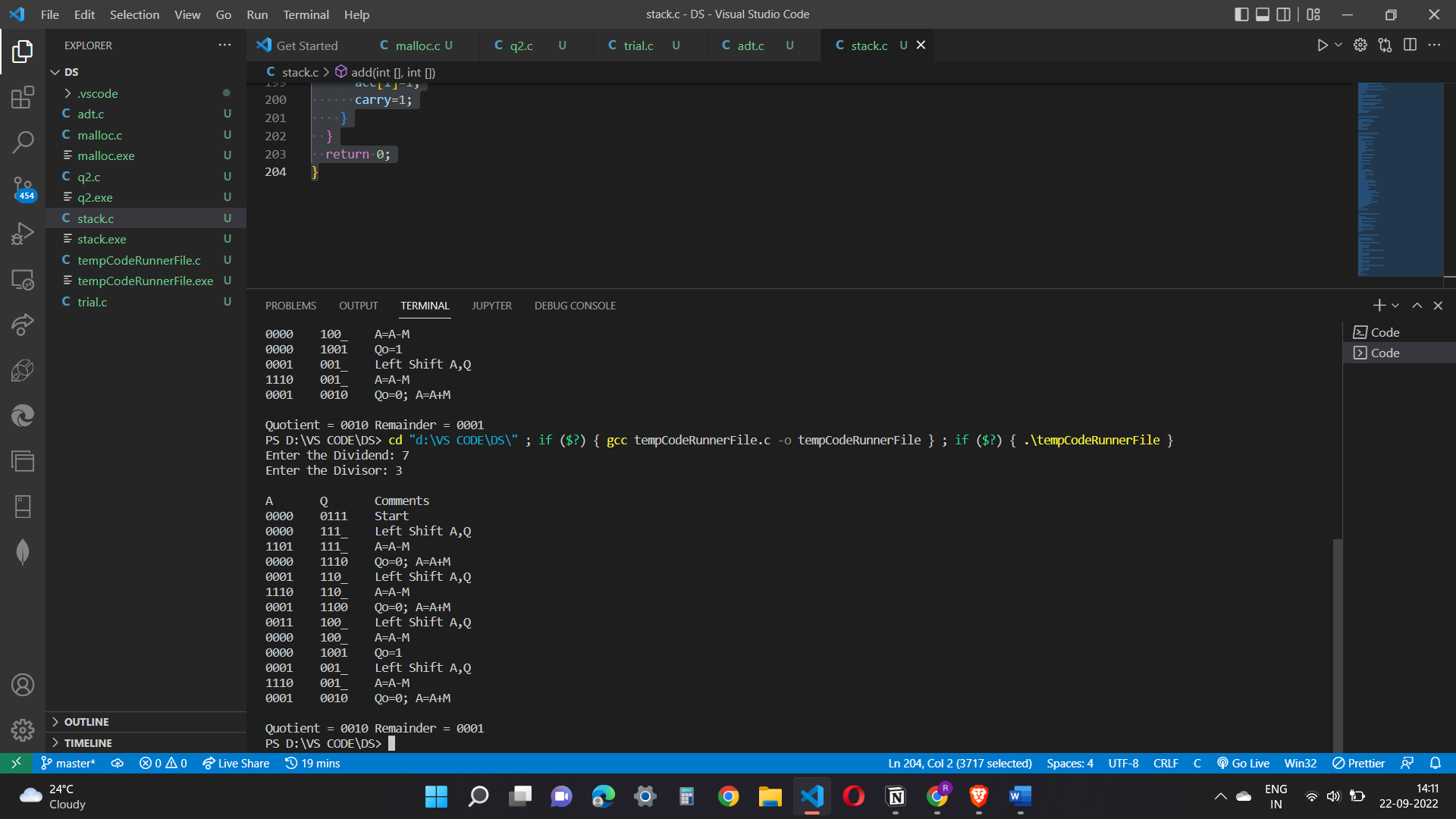
    }

  }

*return* 0;

}

**Output**



**Conclusion**

The Restoring method of division has been studied and its implementation has been

conducted successfully.

**Post Lab Descriptive Questions**

1. **What are the advantages of restoring division over non restoring division?**

In each step of your division calculation the result of the step is either 1 or 0,

depending if the dividend is less than or larger than the divisor.

You generally do a test subtraction for each digit step; if the result is positive or zero,

you note down a 1 as next digit of your quotient.

If the result is negative, you proceed with one of two strategies:

• restoring method: you add the divisor back, and put 0 as your next

quotient digit

• non-restoring method: you don’t do that - you keep negative

remainder and a digit 1, and basically correct things by a

supplementary addition afterwards.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No. 4**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :** To study and implement Non Restoring method of division |

**AIM :** The basis of algorithm is based on paper and pencil approach and the operation involve repetitive shifting with addition and subtraction. So the main aim is to depict the usual process in the form of an algorithm.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

To better understand the non-restoring algorithm and executing it using a programming

language. To find the advantage of non-restoring over restoring division.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

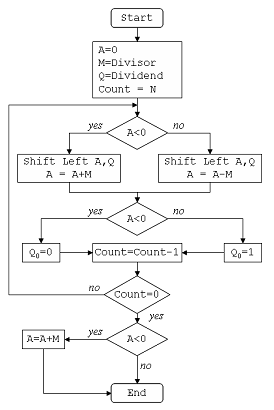
**3**. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

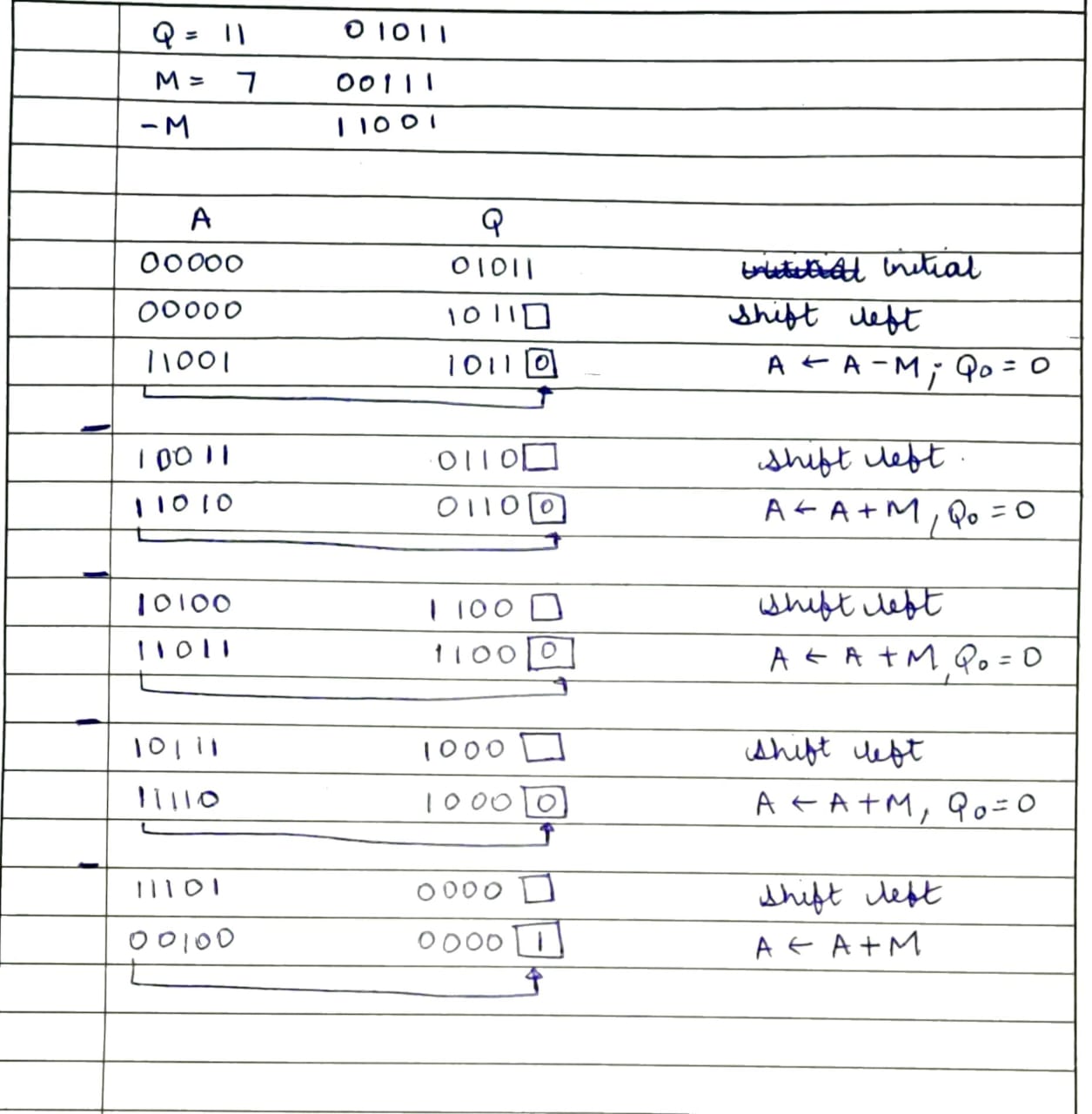
**Pre Lab/ Prior Concepts:**

The Non Restoring algorithm works with any combination of positive and negative numbers.

**Flowchart for Non Restoring of Division:**



**Example: (Handwritten solved problem needs to uploaded)**



Table

Description automatically generated

**Code:**

#include <math.h>

#include <stdio.h>

// NON-RESTORING DIVISION

int main() {

    int a[50], a1[50], b[50], d = 0, i, j;

    int n1, n2, c, k1, k2, n, k, quo = 0, rem = 0;

    // Input the number of bits

    printf("Enter the number of bits\n");

    scanf("%d", &n);

    // Input the divisor and dividend

    printf("Enter the divisor and dividend (n1 n2):\n");

    scanf("%d %d", &n1, &n2);

    // Convert the two numbers to binary representation

    for (c = n - 1; c >= 0; c--) {

        k1 = n1 >> c;

        if (k1 & 1)

            a[n - 1 - c] = 1; // M

        else

            a[n - 1 - c] = 0;

        k2 = n2 >> c;

        if (k2 & 1)

            b[2 \* n - 1 - c] = 1; // Q

        else

            b[2 \* n - 1 - c] = 0;

    }

    // Compute the two's complement of 'a' (negative M)

    for (i = 0; i < n; i++) {

        if (a[i] == 0)

            a1[i] = 1;

        else

            a1[i] = 0;

    }

    a1[n - 1] += 1; // Add 1 to get two's complement of 'a'

    // Handle carry propagation in two's complement

    if (a1[n - 1] == 2) {

        for (i = n - 1; i > 0; i--) {

            if (a1[i] == 2) {

                a1[i - 1] += 1;

                a1[i] = 0;

            }

        }

    }

    if (a1[0] == 2)

        a1[0] = 0;

    // Initialize 'A' with zeros

    for (i = 0; i < n; i++) {

        b[i] = 0;

    }

    printf("A\tQ\tPROCESS\n");

    for (i = 0; i < 2 \* n; i++) {

        if (i == n)

            printf("\t");

        printf("%d", b[i]);

    }

    printf("\n");

    for (k = 0; k < n; k++) { // n iterations

        for (j = 0; j < 2 \* n - 1; j++) // Left shift

        {

            b[j] = b[j + 1];

        }

        for (i = 0; i < 2 \* n - 1; i++) {

            if (i == n)

                printf("\t");

            printf("%d", b[i]);

        }

        printf("\_");

        printf("\tLEFT SHIFT\n");

        if (b[0] == 0) {

            for (i = n - 1; i >= 0; i--) // A = A - M

            {

                b[i] += a1[i];

                if (i != 0) {

                    if (b[i] == 2) {

                        b[i - 1] += 1;

                        b[i] = 0;

                    }

                    if (b[i] == 3) {

                        b[i - 1] += 1;

                        b[i] = 1;

                    }

                }

            }

            if (b[0] == 2)

                b[0] = 0;

            if (b[0] == 3)

                b[0] = 1;

            for (i = 0; i < 2 \* n - 1; i++) {

                if (i == n)

                    printf("\t");

                printf("%d", b[i]);

            }

            printf("\_");

            printf("\tA-M\n");

        } else {

            for (j = n - 1; j >= 0; j--) // A = A + M

            {

                b[j] += a[j];

                if (j != 0) {

                    if (b[j] == 2) {

                        b[j - 1] += 1;

                        b[j] = 0;

                    }

                    if (b[j] == 3) {

                        b[j - 1] += 1;

                        b[j] = 1;

                    }

                }

                if (b[0] == 2)

                    b[0] = 0;

                if (b[0] == 3)

                    b[0] = 1;

            }

            for (i = 0; i < 2 \* n - 1; i++) {

                if (i == n)

                    printf("\t");

                printf("%d", b[i]);

            }

            printf("\_");

            printf("\tA+M\n");

        }

        if (b[0] == 0) {

            b[2 \* n - 1] = 1;

            for (i = 0; i < 2 \* n; i++) {

                if (i == n)

                    printf("\t");

                printf("%d", b[i]);

            }

            printf("\tQ0=1\n");

        }

        if (b[0] == 1) {

            b[2 \* n - 1] = 0;

            for (i = 0; i < 2 \* n; i++) {

                if (i == n)

                    printf("\t");

                printf("%d", b[i]);

            }

            printf("\tQ0=0\n");

        }

    }

    // Handle the final iteration when b[0] == 1 (A = A + M)

    if (b[0] == 1) {

        for (j = n - 1; j >= 0; j--) // A = A + M

        {

            b[j] += a[j];

            if (j != 0) {

                if (b[j] == 2) {

                    b[j - 1] += 1;

                    b[j] = 0;

                }

                if (b[j] == 3) {

                    b[j - 1] += 1;

                    b[j] = 1;

                }

            }

            if (b[0] == 2)

                b[0] = 0;

            if (b[0] == 3)

                b[0] = 1;

        }

        for (i = 0; i < 2 \* n; i++) {

            if (i == n)

                printf("\t");

            printf("%d", b[i]);

        }

        printf("\tA+M\n");

    }

    printf("\n");

    // Calculate the quotient and remainder

    for (i = n; i < 2 \* n; i++) {

        quo += b[i] \* pow(2, 2 \* n - 1 - i);

    }

    for (i = 0; i < n; i++) {

        rem += b[i] \* pow(2, n - 1 - i);

    }

    // Output the results

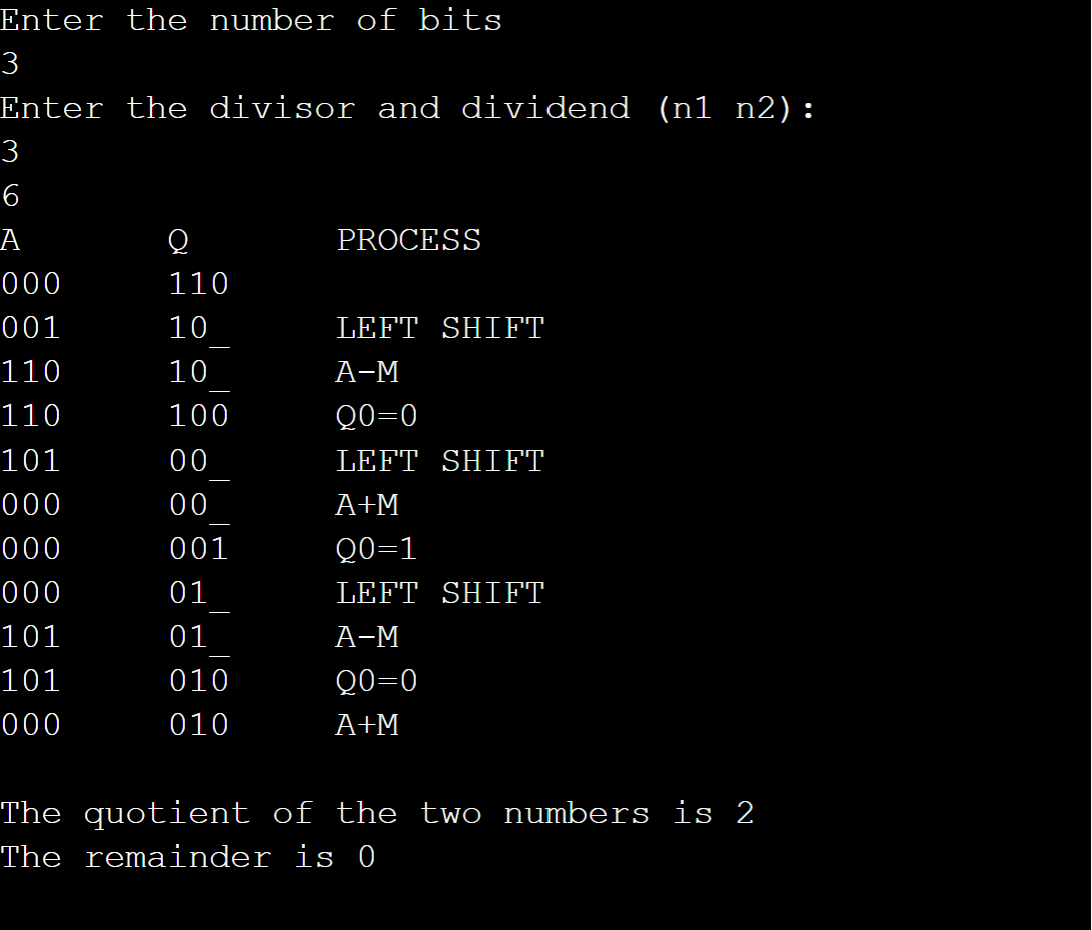
    printf("The quotient of the two numbers is %d\nThe remainder is %d", quo, rem);

    printf("\n");

    return 0;

}

**Output:**

****

**Conclusion**

Successfully executed and coded the algorithm for non-restoring division. In this experiment, Non-Restoring Division Algorithm is executed with the help of C++ programming.

The advantage of Non-Restoring Division over Restoring Division is better understood.

**Post Lab Descriptive Questions**

**1. What are the advantages of non-restoring division over restoring division?**

Non-restoring division uses the digit set {−1, 1} for the quotient digits instead

of {0, 1}. Non-Restoring Division when implemented in hardware, there is only one

decision and addition/subtraction per quotient bit; there is no restoring step after the

subtraction, which potentially cuts down the numbers of operations by up to half and

lets it be executed faster.

Restoring method: you add the divisor back, and put 0 as your next quotient digit

Non-restoring method: you don’t do that - you keep negative remainder and a digit

1, and basically correct things by a supplementary addition afterwards.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Implementation of IEEE-754 floating point representation** |

**AIM:** To demonstrate the single and double precision formats to represent floating point numbers.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

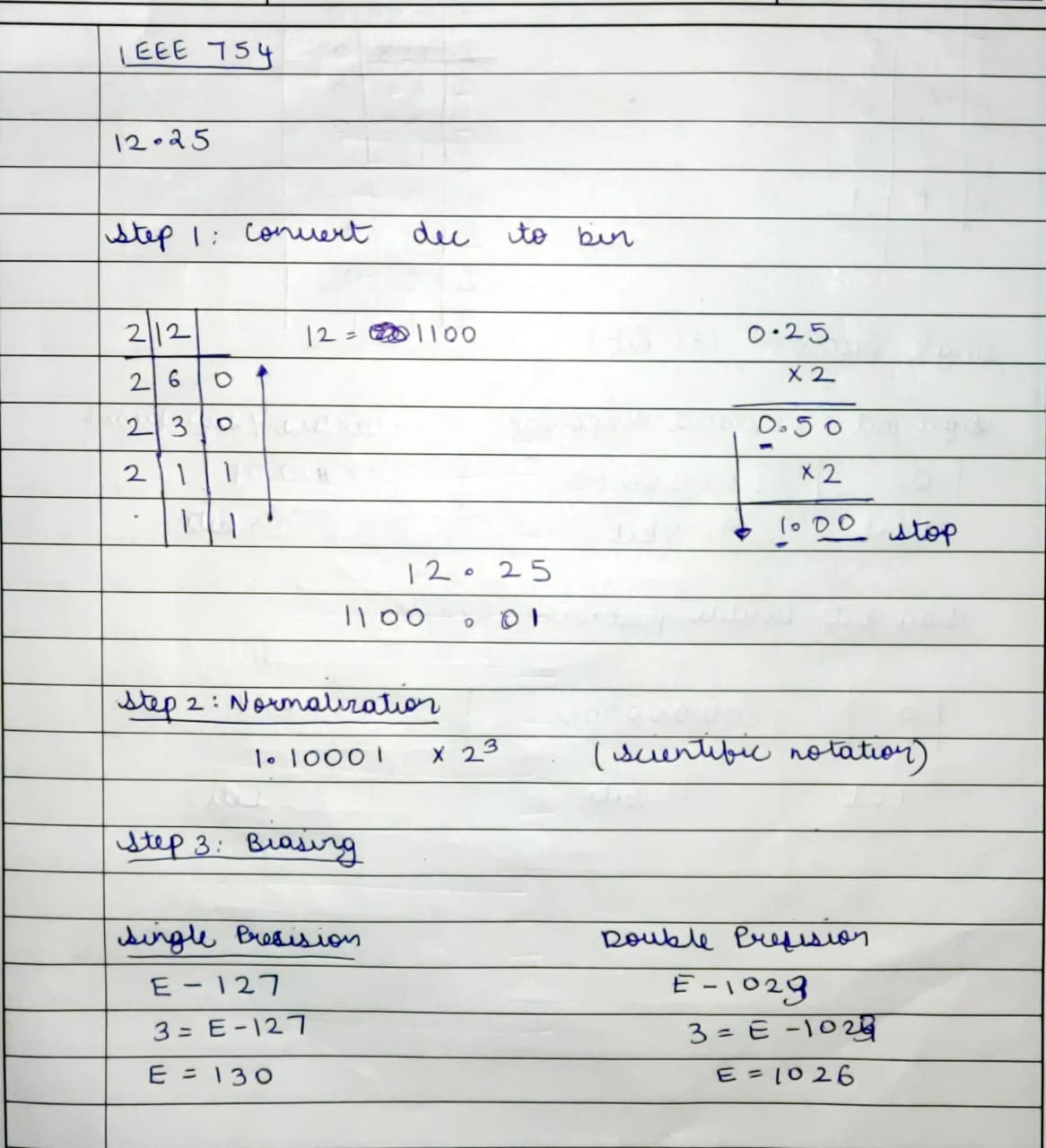
The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a [technical standard](https://en.wikipedia.org/wiki/Technical_standard) for [floating-point](https://en.wikipedia.org/wiki/Floating_point) computation established in 1985 by the [Institute of Electrical and Electronics Engineers](https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers) (IEEE). The standard [addressed many problems](https://en.wikipedia.org/wiki/Floating_point#IEEE_754_design_rationale) found in the diverse floating point implementations that made them difficult to use reliably and [portably](https://en.wikipedia.org/wiki/Software_portability). Many hardware [floating point units](https://en.wikipedia.org/wiki/Floating_point_unit) now use the IEEE 754 standard.

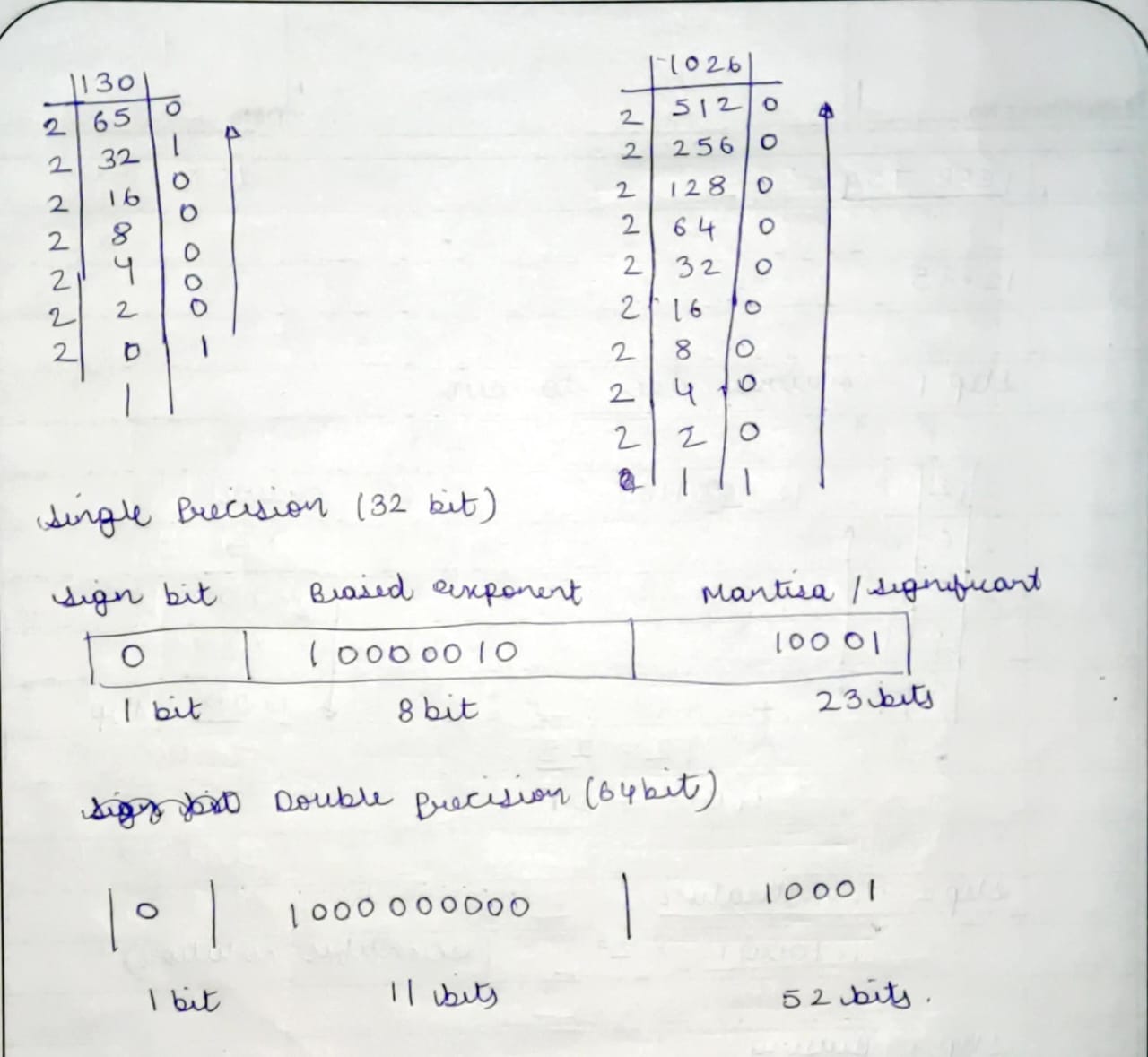
The standard defines:

* *arithmetic formats:* sets of [binary](https://en.wikipedia.org/wiki/Binary_code) and [decimal](https://en.wikipedia.org/wiki/Decimal) floating-point data, which consist of finite numbers (including [signed zeros](https://en.wikipedia.org/wiki/Signed_zero) and [subnormal numbers](https://en.wikipedia.org/wiki/Subnormal_number)), [infinities](https://en.wikipedia.org/wiki/Infinity), and special "not a number" values ([NaNs](https://en.wikipedia.org/wiki/NaN))
* *interchange formats:* encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
* *rounding rules:* properties to be satisfied when rounding numbers during arithmetic and conversions
* *operations:* arithmetic and other operations (such as [trigonometric functions](https://en.wikipedia.org/wiki/Trigonometric_functions)) on arithmetic formats
* *exception handling:* indications of exceptional conditions (such as [division by zero](https://en.wikipedia.org/wiki/Division_by_zero), overflow, *etc*

**Example (Single Precision- 32 bit representation )**

**Example (Double Precision- 64 bit representation )**





**Implementation:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int bi[11], f[23], sign[1], expo[8], frac[23];

int expo1[11], fract[52];

int m = 0, fl = 0, i;

// to convert decimal to binary

void binary(int n)

{

    while (n > 0)

    {

        bi[m] = n % 2;

        n = n / 2;

        m++;

    }

}

// to convert floating decimal to binary

void floating(float x)

{

    for (i = 0; i < 23; i++)

    {

        x = x \* 2;

        f[i] = (int)x;

        x = x - f[i];

    }

}

// for finding single and double precision

void precision(int num)

{

    int e, ee, ee1, k = 0, j = 0, l, r = 0;

    while (m != 0)

    {

        if (bi[m] == 1)

        {

            e = m;

            ee = m + 127;

            ee1 = m + 1023;

            printf("\nSingle precision:\nBiased exponent:%d\n", ee);

            printf("\nDouble precision:\nBiased exponent:%d\n", ee1);

            while (ee1 > 0)

            {

                expo1[r] = ee1 % 2;

                ee1 = ee1 / 2;

                r++;

            }

            printf("\n");

            printf("%d.", bi[m]);

            m--;

            for (i = m; i >= 0; i--)

            {

                frac[k] = bi[i];

                fract[k] = bi[i];

                printf("%d", frac[k]);

                k++;

            }

            for (i = 0; i < 10; i++)

            {

                frac[k] = f[i];

                fract[k] = f[i];

                printf("%d", frac[k]);

                k++;

            }

            printf(" x 2^%d", e);

            printf("\n");

            if (num > 0)

                sign[0] = 0;

            else

                sign[0] = 1;

            while (ee > 0)

            {

                expo[j] = ee % 2;

                ee = ee / 2;

                j++;

            }

            // Display

            printf("\nSingle bit precision:\n");

            printf("\nSign bit     Exponent\t \t \t        Mantissa\n");

            printf("%d", sign[0]);

            printf("\t\t\t");

            for (i = j; i >= 0; i--)

                printf("%d", expo[i]);

            printf("\t\t\t");

            for (i = 0; i < 23; i++)

                printf("%d", frac[i]);

            printf("\n");

            // Display

            printf("\nDouble bit precision:\n");

            printf("\nSign bit     Exponent\t \t \t        Mantissa\n");

            printf("%d", sign[0]);

            printf("\t\t\t");

            for (i = r; i >= 0; i--)

                printf("%d", expo1[i]);

            printf("\t\t\t");

            for (i = 0; i < 52; i++)

                printf("%d", fract[i]);

            break;

        }

        else

            m--;

    }

}

int main(void)

{

    float num, x;

    int n;

    printf("Enter the no.: ");

    scanf("%f", &num);

    n = (int)fabs(num);

    x = fabs(num) - n;

    binary(n);

    floating(x);

    printf("\nIEEE Representation:\n");

    precision(num);

    return 0;

}

**Output:**



**Post Lab Descriptive Questions**

1. **Give the importance of IEEE-754 representation for floating point numbers?**

* The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**.
* The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC’s, Macs, and most Unix platforms.
* There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases.

**Conclusion :** The code for single and double precision formats to represent floating point numbers was executed successfully.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A3 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Implementation of LRU Page Replacement Algorithm.** |

**AIM:** The LRU algorithm replaces the least recently used that is the last accessed memory block from user.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

It follows a simple logic, while replacing it will replace that page which has least recently used out of all.

a) A hit is said to be occurred when a memory location requested is already in the cache.

b) When cache is not full, the number of blocks is added.

c) When cache is full, the block is replaced which is recently used

**Algorithm:**

1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing least recently used element
8. Repeat step 3 to 7 for remaining elements
9. Display the cache at very instance of step 8
10. Print hit ratio
11. End

**Example:**

*#include* <stdio.h>

int main()

{

int q[20], p[50], c = 0, c1, d, f, i, j, k = 0, n, r, t, b[20], c2[20], hit;

float fh;

printf("Enter number of frames:- ");

scanf("%d", &f);

printf("Enter number of pages:- ");

scanf("%d", &n);

printf("Enter the reference string:- ");

*for* (i = 0; i < n; i++)

scanf("%d", &p[i]);

q[k] = p[k];

printf("\n\t%d\n", q[k]);

c++;

k++;

*for* (i = 1; i < n; i++)

{

c1 = 0;

*for* (j = 0; j < f; j++)

{

*if* (p[i] != q[j])

c1++;

}

*if* (c1 == f)

{

c++;

*if* (k < f)

{

q[k] = p[i];

k++;

*for* (j = 0; j < k; j++)

printf("\t%d", q[j]);

printf("\n");

}

*else*

{

*for* (r = 0; r < f; r++)

{

c2[r] = 0;

*for* (j = i - 1; j < n; j--)

{

*if* (q[r] != p[j])

c2[r]++;

*else*

*break*;

}

}

*for* (r = 0; r < f; r++)

b[r] = c2[r];

*for* (r = 0; r < f; r++)

{

*for* (j = r; j < f; j++)

{

*if* (b[r] < b[j])

{

t = b[r];

b[r] = b[j];

b[j] = t;

}

}

}

*for* (r = 0; r < f; r++)

{

*if* (c2[r] == b[0])

q[r] = p[i];

printf("\t%d", q[r]);

}

printf("\n");

}

}

}

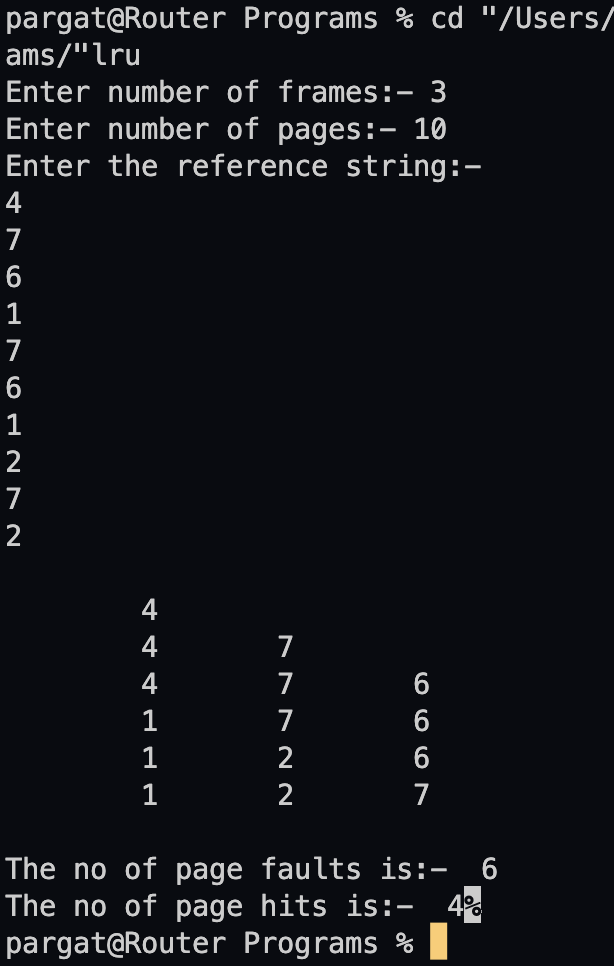
printf("\nThe no of page faults is:- %d", c);

hit = n - c;

printf("\nThe no of page hits is:- %d", hit);

*return* 0;

}



**Post Lab Descriptive Questions**

**1. Define hit rate and miss ratio?**

A hit ratio is a calculation of cache hits, and comparing them with how many total content requests were received.

A miss ratio is the flip side of this where the cache misses are calculated and compared with the total number of content requests that were received.

**2. What is the need for virtual memory**?

Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

**Conclusion : Successfully executed the given program.**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A3 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :**Implementation ofFIFO Page Replacement Algorithm |

**AIM:** The FIFO algorithm uses the principle that the block in the set which has been in for the longest time will be replaced

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

**3**. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**---------------------------------------------------------------------------------------------------------**

**Pre Lab/ Prior Concepts:**

T he FIFO algorithm uses the principle that the block in the set which has been in the block for the longest time is replaced. FIFO is easily implemented as a round robin or criteria buffer technique. The data structure used for implementation is a queue. Assume that the number of cache pages is three. Let the request to this cache is shown alongside.

**Algorithm:**

1. A hit is said to be occurred when a memory location requested is already in the cache.

2. When cache is not full, the number of blocks is added.

3. When cache is full, the block is replaced which was added first

**Design Steps:**

1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing first element (which is in first).
8. Repeat step 3 to 7 for remaining elements
9. Display the cache at very instance of step 8
10. Print hit ratio
11. End.

**Example:**

*#include* <bits/stdc++.h>

using namespace std;

int c;

void fifo(string *st*)

{

int arr[c], hit = 0, k = 0;

bool p = 0;

*for* (int i = 0; i < c; i++)

{

arr[i] = -1;

}

*for* (int i = 0; i < *st*.length(); i++)

{

*if* (k < c)

{

int ele = int(*st*[i] - '0');

*for* (int j = 0; j < c; j++)

{

*if* (arr[j] == ele)

{

p = 1;

*break*;

}

}

*if* (p)

{

hit++;

p = 0;

*continue*;

}

*else*

{

arr[k] = ele;

k = ++k % c;

}

}

*for* (int j = 0; j < c; j++)

{

*if* (arr[j] != -1)

cout << arr[j] << " ";

*else*

cout << "- ";

}

cout << endl;

}

cout << "The hit ratio:" << hit << "/" << *st*.length() << endl;

}

int main()

{

string str;

cout << "Enter the number of page frames: ";

cin >> c;

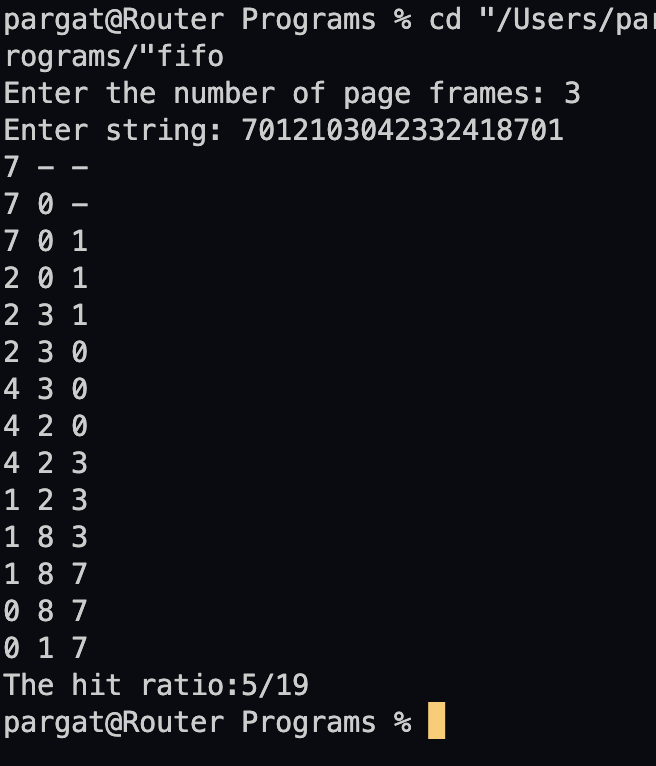
cout << "Enter string: ";

cin >> str;

fifo(str);

}

**Output:**



**Post Lab Descriptive Questions**

**1. What is meant by memory interleaving?**

Memory Interleaving is an abstraction technique which divides memory into a number

of modules such that successive words in the address space are placed in the different module.

**2. Explain Paging Concept?**

Paging is a storage mechanism that allows OS to retrieve processes from the secondary

storage into the main memory in the form of pages. In the Paging method, the main

memory is divided into small fixed-size blocks of physical memory, which is called

frames. The size of a frame should be kept the same as that of a page to have maximum

utilization of the main memory and to avoid external fragmentation. Paging is used for

faster access to data, and it is a logical concept.

**Conclusion : We have successfully implemented FIFO Page Replacement Algorithm.**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A3 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE :** Implementation of Cache Mapping Techniques. |

**AIM:** To study and implement concept of various mapping techniques designed for cache memory.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Cache memory: The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.

2. Hit Ratio: You want to increase as much as possible the likelihood of the cache containing the memory addresses that the processor wants.

**Hit Ratio= No. of hits/ (No. of hits + No. of misses)**

There are only fewer cache lines than the main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines. Further a means is needed for determining which main memory block currently occupies in a cache line. The choice of cache function dictates how the cache is organized. Three techniques can be used.

1. Direct mapping.
2. Associative mapping.
3. Set Associative mapping.

**Direct Mapped Cache**: The direct mapped cache is the simplest form of cache and the easiest to check for a hit. Since there is only one possible place that any memory location can be cached, there is nothing to search; the line either contains the memory information we are looking for, or it doesn't.  
Unfortunately, the direct mapped cache also has the worst performance, because again there is only one place that any address can be stored. Let's look again at our 512 KB level 2 cache and 64 MB of system memory. As you recall this cache has 16,384 lines (assuming 32-byte cache lines) and so each one is shared by 4,096 memory addresses. In the absolute worst case, imagine that the processor needs 2 different addresses (call them X and Y) that both map to the same cache line, in alternating sequence (X, Y, X, Y). This could happen in a small loop if you were unlucky. The processor will load X from memory and store it in cache. Then it will look in the cache for Y, but Y uses the same cache line as X, so it won't be there. So Y is loaded from memory, and stored in the cache for future use. But then the processor requests X, and looks in the cache only to find Y. This conflict repeats over and over. The net result is that the hit ratio here is 0%. This is a worst case scenario, but in general the performance is worst for this type of mapping.

**Fully Associative Cache:** The fully associative cache has the best hit ratio because any line in the cache can hold any address that needs to be cached. This means the problem seen in the direct mapped cache disappears, because there is no dedicated single line that an address must use.However (you knew it was coming), this cache suffers from problems involving searching the cache. If a given address can be stored in any of 16,384 lines, how do you know where it is? Even with specialized hardware to do the searching, a performance penalty is incurred. And this penalty occurs for all accesses to memory, whether a cache hit occurs or not, because it is part of searching the cache to determine a hit. In addition, more logic must be added to determine which of the various lines to use when a new entry must be added (usually some form of a "least recently used" algorithm is employed to decide which cache line to use next). All this overhead adds cost, complexity and execution time.

**Set Associative Cache (To be filled in by students)**

After CPU generates a memory request,

* The set number field of the address is used to access the set of the cache.
* The tag field of the CPU address is then compared with the tags of all k lines
* within that set.
* If the CPU tag matches to the tag of any cache line, a cache hit occurs.
* If the CPU tag does not match to the tag of any cache line, a cache miss occurs.
* In case of a cache miss, the required word must be brought from the main
* memory.
* If the cache is full, a replacement is made in accordance with the employed
* replacement policy.

**Direct Mapping Implementation:**

The mapping is expressed as

**i=j modulo m**

i=cache line number

j= main memory block number

m= number of lines in the cache

* Address length = (s+w) bits
* Number of addressable units = 2s+w words or bytes
* Block size = line size = 2w words or bytes
* Number of blocks in main memory = 2s+w / 2w = 2s
* Number of lines in cache = m = 2r
* Size of tag = (s-r) tags

**Associative Mapping Implementation**: **(To be filled in by students)**

* Address length =(s+w) bits
* Number of addressable units= 2 s+wwords or bytes
* Block size=line size = 2 wwords or bytes
* Number of blocks in main memory = 2 s
* Number of lines in cache = undefined
* size of tags = s bits

**Set** **Associative Mapping Implementation**:

m=v\*k

i=j modulo n, where:

* i=cache line number
* j=main memory block number
* m=number of lines in the cache
* v=number of sets
* k=number of lines in each set
* Addressable length=(s+w) bits
* Number of addressable units= 2 𝑠+𝑤words or bytes
* Block size=line size=2 𝑤words or bytes
* Number of blocks in main memory=2 𝑠
* Number of lines in set=k
* Number of sets = v = 2 𝑑
* Number of lines in cache = m = kv = k\*2 𝑑
* Size of cache = k \* 2 𝑑+𝑤words or bytes
* Size of tag = (s-d) bits

**Code:**

*#include* <bits/stdc++.h>

using namespace std;

int main()

{

int memory\_lines, blocks;

cout << "Enter number of main memory lines:";

cin >> memory\_lines;

cout << "Enter number of blocks in the main memory:";

cin >> blocks;

int bmemory[blocks][4];

int mmemory[memory\_lines];

cout << "\nEnter the main memory data:" << endl;

*for* (int i = 0; i < memory\_lines; i++)

{

cout << "Line no. " << i + 1 << ": ";

cin >> mmemory[i];

}

int k = 0;

*for* (int i = 0; i < blocks; i++)

*for* (int j = 0; j < 4; j++)

bmemory[i][j] = mmemory[k++];

cout << "\nDirect mapped cache\n";

*for* (int i = 0; i < blocks; i++)

{

cout << endl

<< "Block " << i << ": ";

*for* (int j = 0; j < 4; j++)

cout << bmemory[i][j] << " ";

}

cout << "\n\nSample cache:\n";

*for* (int i = 0; i < blocks; i++)

{

int random = rand() % 5;

cout << bmemory[i][random] << " ";

}

}

**Text

Description automatically generated**

**Post Lab Descriptive Questions**

**1. For a direct mapped cache, a main memory is viewed as consisting of 3 fields. List and define 3 fields.**

* One field on the direct-mapped cache memory identifies a unique word or byte within a block of main memory
* The remaining two fields specify one of the blocks of main memory
* These two fields are a line field, which identifies one of the lines of the cache, and a tag field, which identifies one of the blocks that can fit into that line

**2. What is the general relationship among access time, memory cost, and capacity?**

* Faster access time is directly proportional to cost per bit, it means that as the access time speed increases the cost per bit also increases
* Memory capacity is inversely proportional to cost per bit, it means that as memory capacity increases, the cost per bit decreases
* Memory capacity is inversely proportional to access time, it means that as memory capacity increases the access time speed decreases

**Conclusion :** Therefore, with the help of the experiment the various mapping techniques are understood. The given task was implemented by writing programs to demonstrate them.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A3 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 9**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implement simple addition, subtraction, multiplication and division instructions using TASM. |

**AIM:** Implement simple addition, subtraction, multiplication and division instructions using TASM.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mentions the CO/CO’s attained)**

Understand the Central processing unit with addressing modes and working of control unit in depth.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1) Microprocessor architecture and applications with 8085: By Ramesh Gaonkar (Penram International Publication).**

**2) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).**

**Pre Lab/ Prior Concepts:**

**Assembler directives: These are statements that direct the assembler to do something**

**Definition:**

**Types of Assembler Directives:**

**ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

**Example:**

**ASUME CS:CODE** ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

**ASUME DS:DATA** ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA

**Start:**

It is entry point of the program. without this program won’t run.

**END** - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

**ENDS** - This ENDS directive is used with name ofthe segment to indicate the end of that logic segment.

**Example:**

**CODE SEGMENT** ;

Hear it Start the logic

;segment containing code

; Some instructions statements to perform the logical

;operation

**CODE ENDS** ;End of segment named as;CODE

**Arithmetic instruction set:**

**ADD instruction:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flags** |
|  |  |  |  | **Affected** |
|  |  |  |  |  |
| ADD | Addition | ADD D, S | (S) + (D)🡪(D) | All |
|  |  |  |  |  |
|  |  |  | Carry🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |
| ADC | Add with | ADC D, S | (S) + (D) +(CF) | All |
|  | Carry |  | 🡪 (D) |  |
|  |  |  | Carry🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Syntax: ADD destination,source**

**SUB instruction:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flags Affected** |
|  |  |  |  |  |
| SUB | Subtract | SUB D, S | (D) - (S)🡪(D) | All |
|  |  |  |  |  |
|  |  |  | Borrow🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |
| SBB | Subtract with | SBB D, S | (D) - (S) –(CF)🡪(D) | All |
|  | Borrow |  |  |  |
|  |  |  |  |  |

**MUL instruction:**

**Syntax: MUL source**

|  |  |  |  |
| --- | --- | --- | --- |
| **Multiplication** | **Multiplicand** | **Operand** | **Result** |
| **(MUL or IMUL)** |  | **(Multiplier)** |  |
|  |  |  |  |
| Byte \* Byte | AL | Register or | AX |
|  |  | Memory |  |
|  |  |  |  |
| Word \* Word | AX | Register or memory | DX :AX |

**DIV instruction:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Division** | **Dividend** | **Operand** | **Quotient : Remainder** |
| **(DIV or IDIV)** |  | **(Divisor)** |  |
|  |  |  |  |
| Word / Byte | AX | Register or memory | AL : AH |
|  |  |  |  |
| Dword / Word | DX:AX | Register or memory | AX : DX |
|  |  |  |  |

**The steps to execute a program in TASM are**

**ASSEMBLING AND EXECUTING THE ROGRAM**

1. **Writing an Assembly Language Program**

Assembly level programs generally abbreviated as ALP are written in text editor EDIT.

Type *EDIT* in front of the command prompt **(C:\TASM\BIN)** to open an untitled text file.

*EDIT<file name>*

After typing the program save the file with appropriate file name with an extension *.ASM*

Ex:Add.ASM

1. **Assembling an Assembly Language Program**

To assumble an ALP we needed executable file called MASM.EXE. Only if this file is in current working directory we can assemble the program. The command is

*TASM<filename.ASM>*

If the program is free from all syntactical errors, this command will give the **OBJEC**T file.In case of errors it list out the number of errors, warnings and kind of error.

**Note: No object file is created until all errors are rectified.**

1. **Linking**

After successful assembling of the program we have to link it to get **Executable file.**

The command is

*TLINK<File name.OBJ>*

This command results in <*Filename.exe>*which can be executed in front of the command prompt.

1. **Executing the Program**

Open the program in debugger by the command(note only exe files can be open)by the command.

*<Filename.exe>*

This will open the program in debugger screen where in you can view the assemble code with the CS and IP values at the left most side and the machine code. Register content,memory content also be viewed using ***TD***option of the debugger & to execute the program in single steps(F7)

**Algorithm for adding the two 8-bit numbers:**

* 1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can’t take input while running the code in assembly language)
  2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
  3. Then move the contents of data to AL
  4. Move the contents of AL to DS
  5. Move the first value(a) to AL
  6. Move the second value(b) to BL
  7. Then add both of them using ADD AL, BL wherein the memory gets stored in AL
  8. Then move the value of the modified AL to c to store the answer
  9. Then perform MOV ah,4ch and then int 21h to interrupt the code
  10. Type “code ends” to end the execution of the code.

**Algorithm for subtracting the two 8 bit numbers:**

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can’t take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then subtract both of them using SUB AL, BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type “code ends” to end the execution of the code.

**Algorithm for multiplying the two 8 bit numbers:**

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can’t take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then multiply both of them using MUL BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type “code ends” to end the execution of the code.

**Algorithm for dividing the two 8-bit numbers:**

1. Define a data segment and then define the two numbers on which the operation is to be performed in two memory locations(a, b)(as we can’t take input while running the code in assembly language)
2. Also define another memory location(c) to store the final answer of the two values on which the operation is to be performed
3. Then move the contents of data to AL
4. Move the contents of AL to DS
5. Move the first value(a) to AL
6. Move the second value(b) to BL
7. Then divide both of them using DIV BL wherein the memory gets stored in AL
8. Then move the value of the modified AL to c to store the answer
9. Then perform MOV ah,4ch and then int 21h to interrupt the code
10. Type “code ends” to end the execution of the code.

**CODE:**

**ADDITION:**

DATA SEGMENT

NUM1 DW 1234H

NUM2 DW 1234H

RES DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:

MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

ADD AX,BX

MOV RES,AX

MOV AH,4CH

INT 21H

CODE ENDS

END START

**SUBTRACTION:**

DATA SEGMENT

NUM1 DW 1255

NUM2 DW 28

RES DW ?

DATA ENDS

CODE SEGMENT

START:

ASSUME CS:CODE,DS:DATA

MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

SUB AX,BX

MOV RES,AX

MOV AH,4CH

INT 21H

CODE ENDS

END START

**MULTIPLICATION:**

DATA SEGMENT

NUM1 DW 1234H

NUM2 DW 1234H

RES DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

MUL BX

MOV RES,AX

MOV AH,4CH

INT 21H

CODE ENDS

END START

**DIVISION:**

DATA SEGMENT

NUM1 DW 1234H

NUM2 DW 1234H

RES DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

DIV BX

MOV RES,AX

MOV AH,4CH

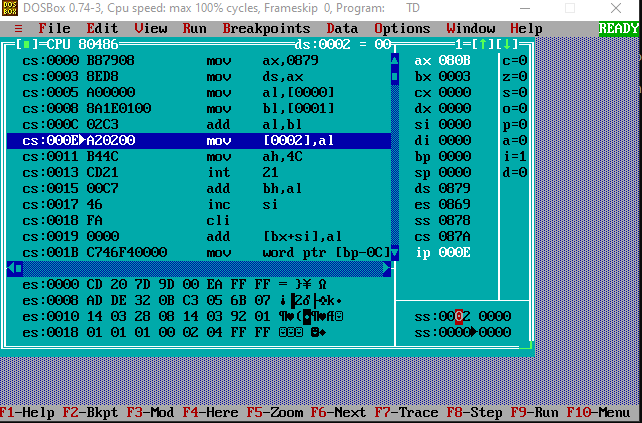
INT 21H

CODE ENDS

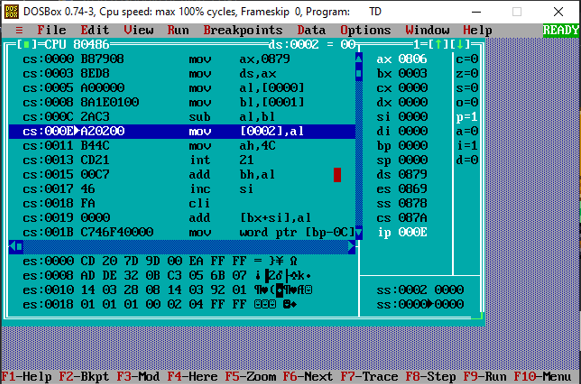
END START

**OUTPUT:**

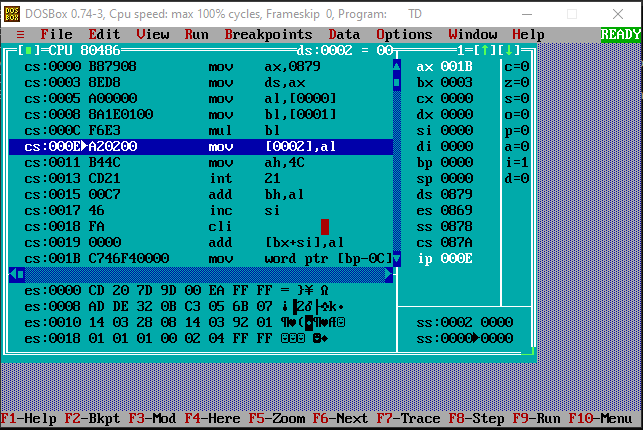
**ADDITION:**



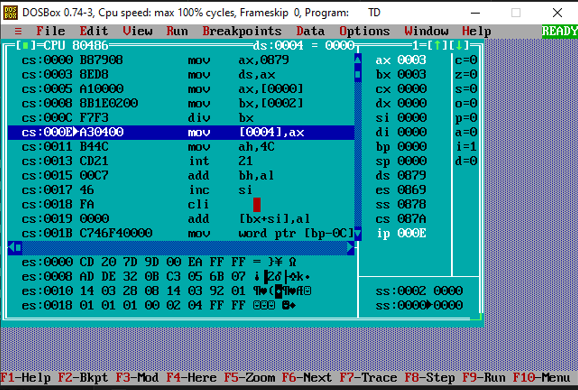
**SUBTRACTION:**

****

**MULTIPLICATION:**



**DIVISION:**



**Conclusion: Successfully implemented the given experiment.**

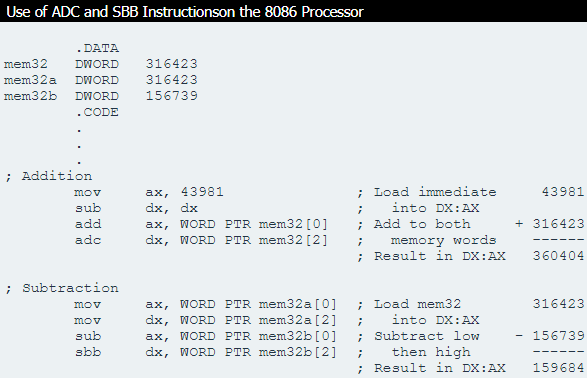
**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain instructions ADC and SBB with example**

Numbers larger than the register size on your processor can be added and subtracted with the ADC (Add with Carry) and SBB (Subtract with Borrow) instructions.

These instructions work as follows:

ADC Dest, Source ; Dest = Dest + Source + Carry Flag SBB Dest, Source ; Dest = Dest - Source - Carry Flag

If the operations prior to an ADC or SBB instruction do not set the carry flag, these instructions are identical to ADD and SUB. While operating on large values in more than one register, ADD and SUB are used for the least significant part of the number and ADC or SBB for the most significant part.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A3 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 10**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Study of multiprocessor configuration concepts through Virtual lab |

**AIM:** Understanding Virtual Lab concepts

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

The objective of this experiment is to learn the fundamentals of Floating Point Representation of Numbers.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

<http://vlabs.iitb.ac.in/vlab/labscse.html>

[http://vlabs.iitb.ac.in/vlab/#](http://vlabs.iitb.ac.in/vlab/)

<http://www.vlab.co.in/>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

The main aim of this experiment is to provide remote-access to Labs in various disciplines of Science and Engineering. These Virtual Labs would cater to students at the undergraduate level, post graduate level as well as to research scholars. Also, to enthuse students to conduct experiments by arousing their curiosity. This would help them in learning basic and advanced concepts through remote experimentation. It also provides a complete Learning Management System around the Virtual Labs where the students can avail the various tools for learning, including additional web-resources, video-lectures, animated demonstrations and self-evaluation. We can share costly equipment and resources, which are otherwise available to limited number of users due to constraints on time and geographical distances

**Salient Features:**

. 1. Virtual Labs will provide to the students the result of an experiment by one of the following methods (or possibly a combination)

* Modeling the physical phenomenon by a set of equations and carrying out simulations to yield the result of the particular experiment. This can, at-the-best, provide an approximate version of the ‘real-world’ experiment.
* Providing measured data for virtual lab experiments corresponding to the data previously obtained by measurements on an actual system.
* Remotely triggering an experiment in an actual lab and providing the student the result of the experiment through the computer interface. This would entail carrying out the actual lab experiment remotely.

2. Virtual Labs will be made more effective and realistic by providing additional inputs to the students like accompanying audio and video streaming of an actual lab experiment and equipment.

**Observations**

**Title of Study Experiment:**

Floating Point Numbers Representation

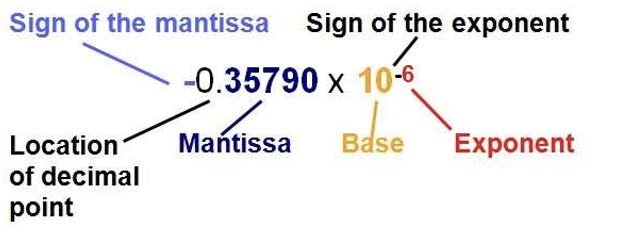
**Brief description of experiment under study:**

Computers use computations with integers and real numbers. In computers we cannot precisely represent all the real numbers. Hence, there is a way to effectively represent them with only a little loss in precision.

Floating point numbers are numbers that contain floating decimal points. Computers recognize real numbers that contain fractions as floating point numbers. When a calculation includes a floating point number, it is called a "floating point calculation." Older computers used to have a separate floating point unit (FPU) that handled these calculations, but now the FPU is typically built into the computer's CPU.

This experiment is to understand the concept of Floating Point Numbers and how they are converted to and from decimal form.

**Learning’s recorded:**



**Exponent Field**

8 - Bits Long

Determines The Range Of Numbers That Can be Represented

Increasing The Bits Will Increase The Range , Not Precision -> To Cover For -ve Numbers , exp = 127 + real exp

# Sign Bit

1- Bit Long

Dtermines The +ve or -ve number -> 1 = -ve Number 0 = +ve Number

# Mantissa Field

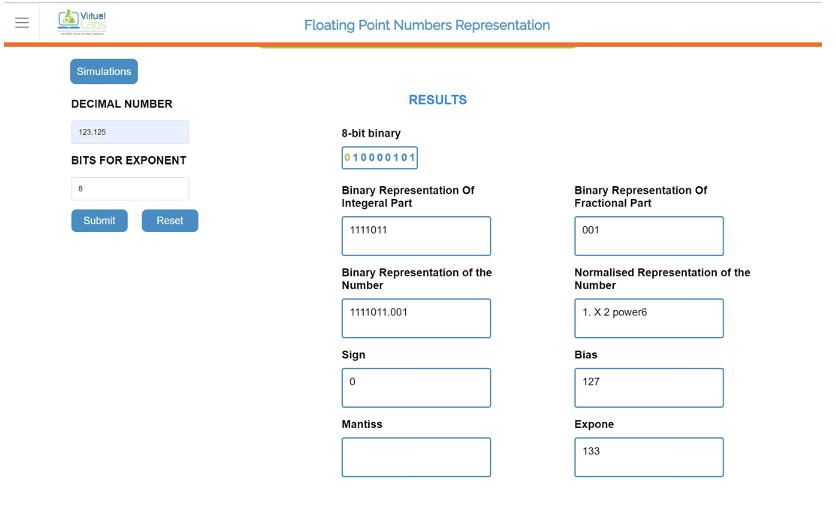
23 - Bits Long

Determines the precision of Numbers

Increasing bits Will Increase precision, not range.

# Procedure for conversion:

* Convert the absolute value of the number to binary, perhaps with a fractional part after the binary point. This can be done by converting the integral and fractional parts seperately.
* Normalize the number. Move the binary point so that it is one bit from the left. Adjust the exponent of two so that the value does not change. Place the mantissa into the mantissa field of the number. Omit the leading one and fill with zeros on the right.
* Add the bias to the exponent of two, and place it in the exponent field. The bias is 2k-1-1, where k is the number of bits in the exponent field. 6.
* For the eight-bit format, k-3, so the bias is 23-1-1-3. For IEEE 32-bit, k-8, so the bias is 28-1-1-127.
* Set the sign bit, 1 for negative, 0 for positive, according to the sign of the original number.



**Knowledge gained / Inference Obtained :**

The importance of Floating Point Representation was understood: Floating point representation makes numerical computation much easier.

The procedure to convert a decimal to floating point representation was also studied with the help of an example and verifying it manually as well.

**Post Lab Descriptive Questions**

**1. What are the applications of the virtual lab case study / tool reviewed by you?**

# Tensor Processing Units (TPUs)

Besides the 64-bit float we explored at length, there are also 32-bit floats (single precision) and 16-bit floats (half-precision) commonly available.

Google’s Tensor Processing Units instead use a modified 16-bit format for multiplication as part of their many optimizations for deep-learning tasks.

# HDR Images

HDR image uses floating point numbers to represent the pixels! This allows a high “dynamic” range (the exponent can be high or low) while still maintaining relative precision across all brightness scales. Perfect for keeping the data from scenes with high contrast.

**Conclusion :**

**Successfully implemented the given experiment.**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**