Batch: B2	**Roll No.: 16010121110**
Experiment / assignment / tutorial No._____	
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
2. https://www.techopedia.com/definition/331/small-computer-system-interface-scsi
3. 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.

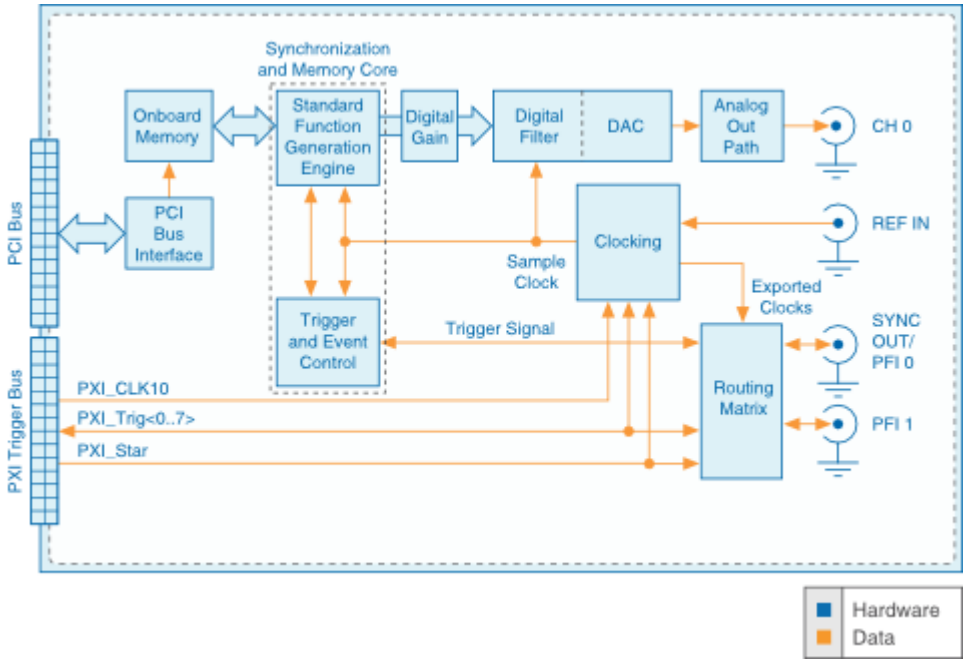
PCI Bus

PCI Bus is a bi-directional bus which transfers data to and from CPU to peripheral devices.

Definition: Peripheral Component Interconnect (PCI) is a term used to describe a common connection interface for attaching computer peripherals to a PC's motherboard.

Architecture:

PCI bus are basically wires. Below is diagram of motherboard architecture, yellow lines represent PCI busses. PCI slots are given on right.



Features:

• coupling of the processor and expansion bus by means of a bridge,

• 32-bit standard bus width with a maximum transfer rate of 133 Mbytes/s,

• supporting of multi-processor systems,

• supporting of 5 V and 3.3 V power supplies

• Multitasking capabilities,

• multiplexing of address and data bus reducing the number of pins

• Configuration through software and registers,

• Processor independent specification

SCSI bus:

SCSI is old version of USB.

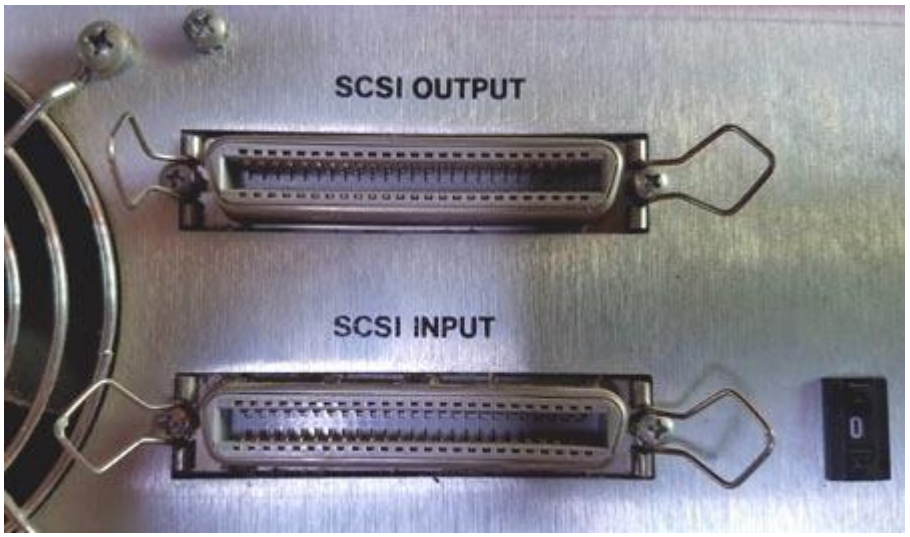
Defination:

The Small Computer System Interface (SCSI) is a set of parallel interface standards developed by the American National Standards Institute (ANSI) for attaching printers, disk drives, scanners and other peripherals to computers. SCSI is supported by all major operating systems.

Architecture:

The SCSI standard specifies eight distinct phases for the SCSI bus:

- **Bus free**. This phase means that no SCSI devices are using the bus, and that the bus is available for another SCSI operation.
- **Arbitration**. This phase is preceded by the bus free phase and permits a SCSI device to gain control of the SCSI bus. During this phase, all devices wishing to use the bus assert the /BSY signal and put their SCSI ID onto the bus (using the data signals). The device with highest SCSI ID wins the arbitration.
- **Selection**. This phase follows the arbitration phase. The device that won arbitration uses this phase to select another device to communicate with.
- **Reselection**. This optional phase is used by systems that allow peripheral devices to disconnect and reconnect from the bus during lengthy operations. This phase is not supported by the original Macintosh SCSI Manager, but is by SCSI Manager 4.3.
- **Command**. During this phase, the target requests a command from the initiator.
- **Data**. The data phase occurs when the target requests a transfer of data to or from the initiator.
- **Status**. This phase occurs when the target requests that status information be sent to the initiator.
- **Message**. The message phase occurs when the target requests the transfer of a message. Messages are small blocks of data that carry information or requests between the initiator and a target. Multiple messages can be sent during this phase.



TYPES OF SCSI BUSES:

- **SCSI-1**
- SCSI-1 is the oldest version. Its asynchronous transmission frequency is 3MB/s, and the synchronous transmission frequency is 5MB/s. Although SCSI-1 is nearly eliminated, it is still used in some scanners and internal ZIP drives, adopting 25-pin interface.

- **SCSI-2**
- Early SCSI-2 is called Fast SCSI. The data transmission rate is promoted from original 5MB/s to 10MB/s by increasing the synchronous transmission frequency. SCSI supports 8-bit parallel data transmission and 7 peripherals can be connected to it.
- **SCSI-3**
- SCSI-3 was born in 1995. It is also called UltraSCSI, which had a faster synchronous transmission rate of 20MB/s. If the 16-bit transmission Wide mode is adopted, the data transmission rate can be promoted to 40MB/s. This SCSI version uses 68-pin interface and it is mainly used in keyboard. The typical characteristic of SCSI-3 is largely increasing the bus frequency and decreasing the signal interference to enhance the stability.

Features:

- New types of peripheral devices can be added to a system without hardware changes; only a new I/O device driver is needed
- SCSI satisfies the high-performance requirements of medium and large systems
- Intelligence can be moved from the host to a peripheral device, thereby off-loading the system or controller's processor
- SCSI's ability to logically disconnect and reconnect devices from the bus means that slow operations can be performed offline, thus allowing several operations in a system to run concurrently

Post Lab Descriptive Questions

Q1 . Differentiate between PCI and SCSI Bus

PCI bus is used to connect components in motherboard and contains slot for extra components. SCSI bus is a slot to which SCSI bus is attached which is used to connect I/O devices.

Q2. List two applications each of PCI and SCSI Bus

PCI bus is used to
1) Connect RAM and CPU.
2) Connect extra components to motherboard.

SCSI bus is used to
1) Connect peripherals like scanners to motherboard
2) Used in old keyboards.

Date: 18 Aug 22_

Signature of faculty in-charge

Batch:	**B2**	**Roll No.:**	**16010121110**

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)

CO1- Describe and define the structure of a computer with buses structure and detail working of the arithmetic logic unit and its sub modules

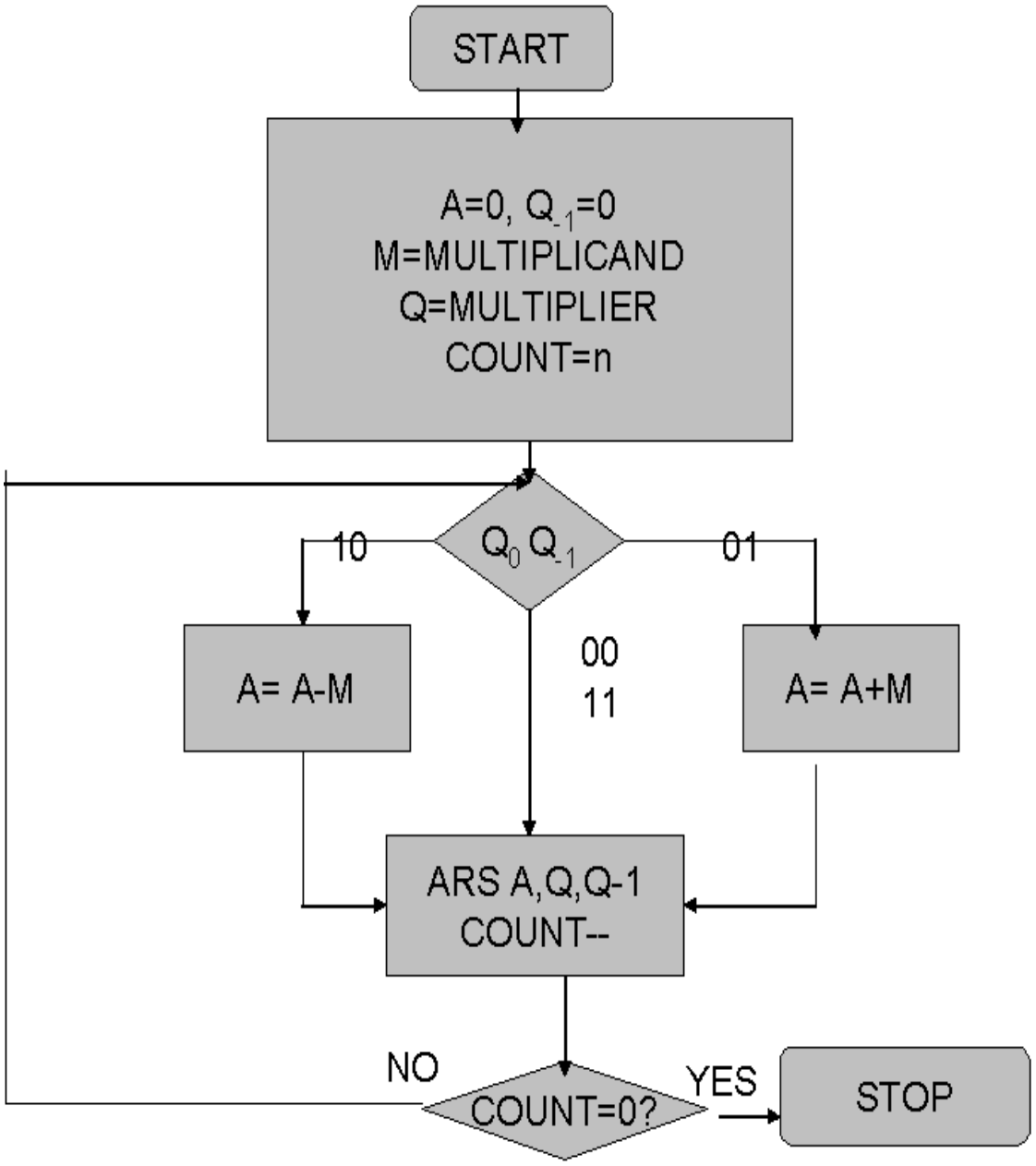
Books/ Journals/ Websites referred:

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:


START
A=0, Q-1=0
M=MULTIPLICAND
Q=MULTIPLIER
COUNT=n
Q0 Q-1
10
01
00
11
A= A-M
A= A+M
ARS A,Q,Q-1
COUNT--
NO
COUNT=0?
YES
STOP

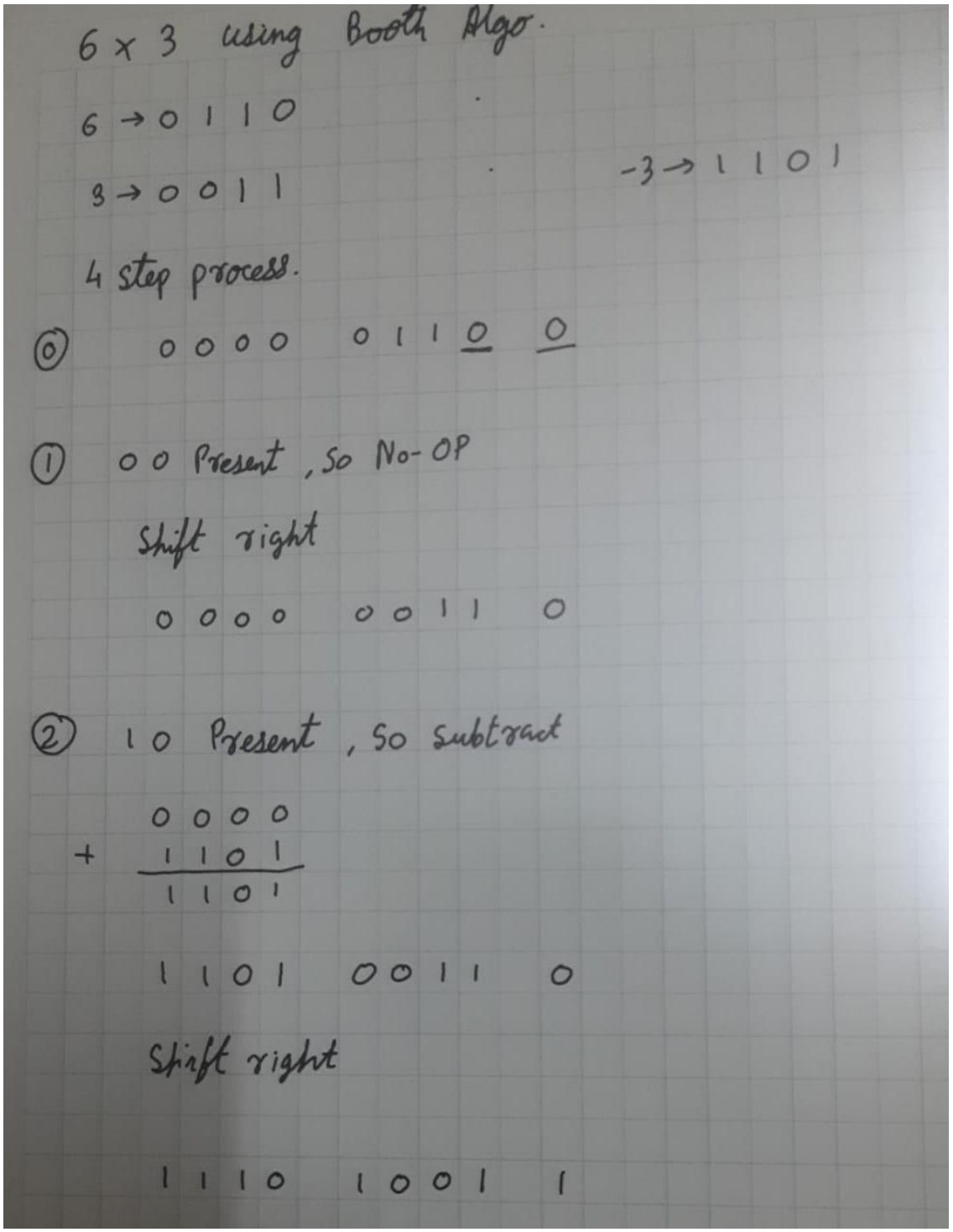
Design Steps:

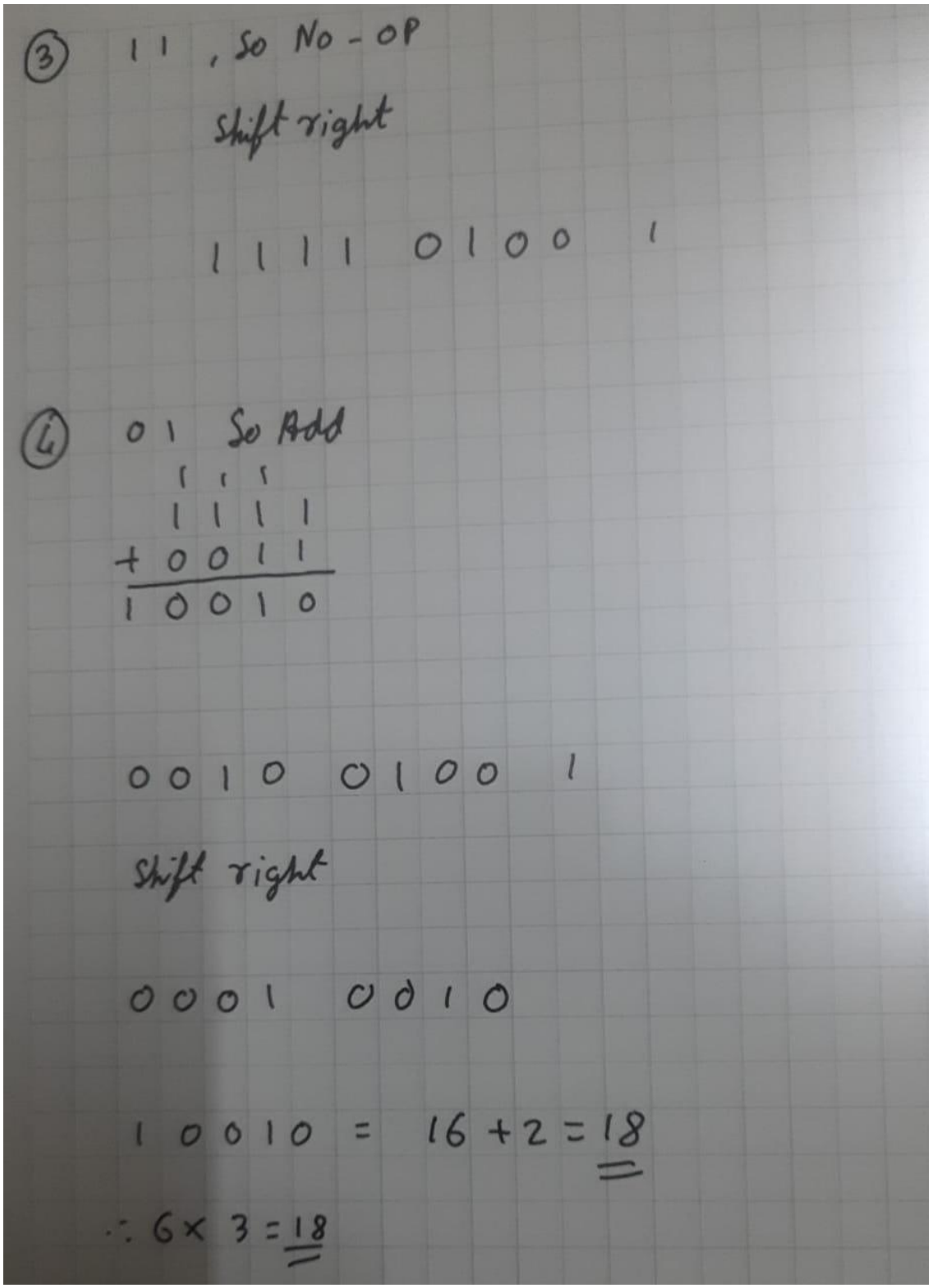
1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q_{-1} =0
4. Convert M and Q into binar
5. Compare Q_0 and Q_{-1} and perform the respective operation.

Q_0 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)




(3) 11, So No-OP
Shift right
1111 0100 1
(4) 01 So Add
1 1 1
1 1 1 1
+ 0 0 1 1
1 0 0 1 0
0010 0100 1
Shift right
0001 0010
10010 = 16 + 2 = 18
∴ 6 × 3 = 18

```
def convert_to_binary(num):
    arr=[]
    while (num!=1):
        arr.append(round(num%2))
        num=num-num%2
        num/=2
    arr.append(1)
    arr=arr[::-1]
    if(len(arr)<5):
        while(len(arr)!=5):
         arr.insert(0,0)
    return arr


def complement(x):
    num=[]
    one=[]#array  eg 0001
    for i in range(0,len(x)):
        one.append(0)
        if(x[i]==0):
            num.append(1)
        if(x[i]==1):
            num.append(0)
```

```
    one.pop()
    one.append(1)
    num=add(num,one)
    return num

def shift_right(c):
    c.pop()
    c.insert(0,c[0])
    return c
def add(x,y):
    carry=0
    z=[]
    x=x[::-1]
    y=y[::-1]
    for i in range(0,len(x)):
        temp=x[i]+y[i]+carry
        carry=0
        if(temp==2):
            carry=1
            temp=0
        if(temp==3):
            carry=1
            temp=1
        z.append(temp)
```

```
    z=z[::-1]
    return z


a_=convert_to_binary(6) #-6
print(a_)
a=complement(a_)#-6


b_=convert_to_binary(4) #4
b=complement(b_) #-4


c=[0,0,0,0,0]+b+[0]
print(c)
"""print(c)
c=shift_right(c)
print(add(a,b))"""
for i in range(0,5):
    if(c[-1]>c[-2]):
        c=add(c[0:5],a)+c[5:]
    if(c[-1]<c[-2]):
       c=add(c[0:5],a_)+c[5:]
    c=shift_right(c)
    print(c)
c=c[5:-1]
print(c)
```

Conclusion:

Thus we have successfully understood the implementation of Booth's multiplication algorithm. It is an algorithm which is used to perform binary multiplication by addition. We implemented it on paper and using code.

Post Lab Descriptive Questions

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

One advantage of the Booth multiplier is, it reduce the number of partial product, thus make it extensively used in multiplier with long operands (>16 bits). The main disadvantage of Booth multiplier is the complexity of the circuit to generate a partial product bit in the Booth encoding

2. **Is Booth's recoding better than Booth's algorithm? Justify**

In some cases. While Booth's algorithm reduces the number of additions, the encoding that it performs may be counterproductive for multipliers such as 010101... The modified Booth's algorithm attempts to address this problem.

Signature of

Batch:B2 Roll No.: 16010121110

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)

Describe and define the structure of a computer with buses structure and detail working of the arithmetic logic unit and its sub modules

---

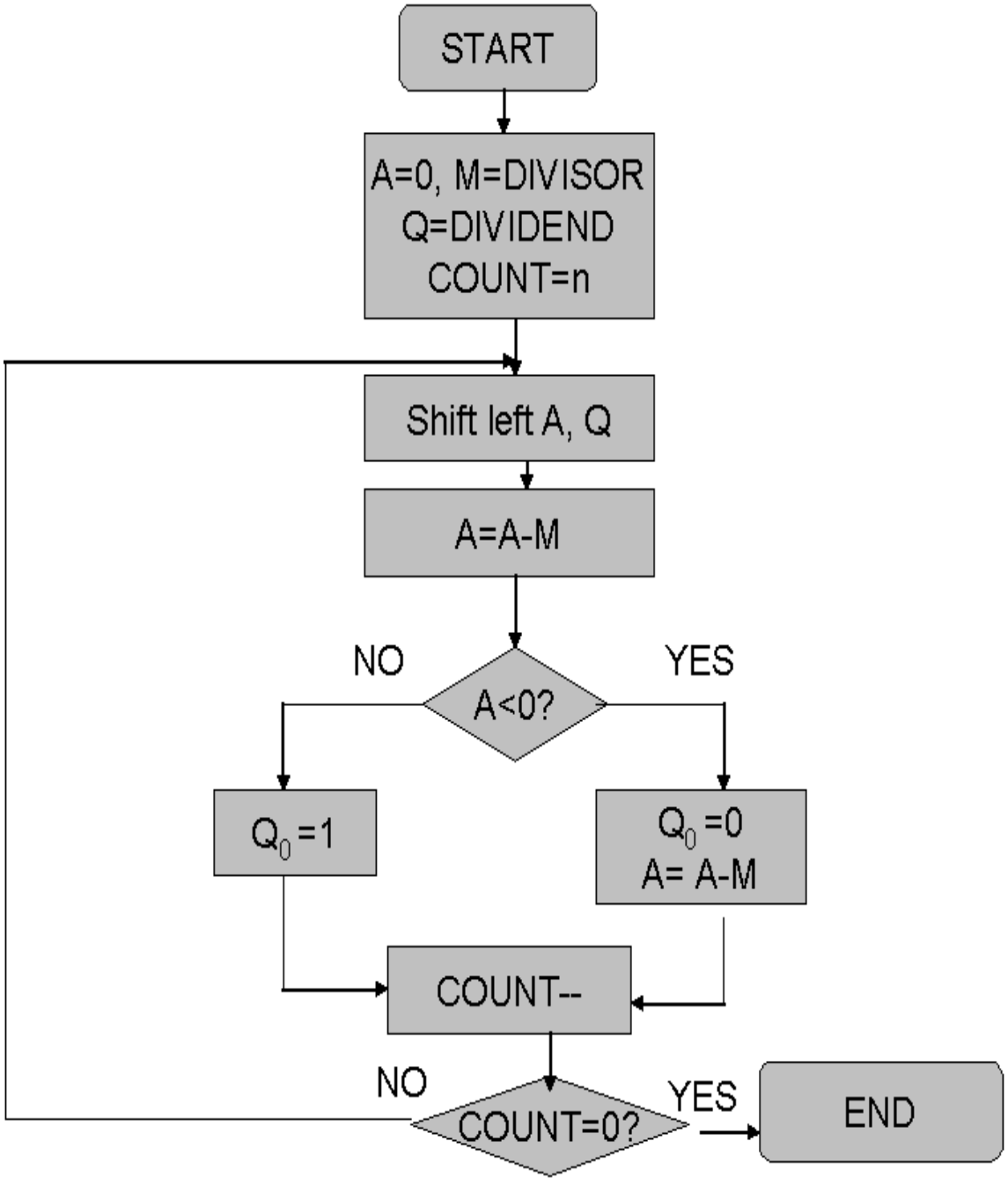
Books/ Journals/ Websites referred:

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.

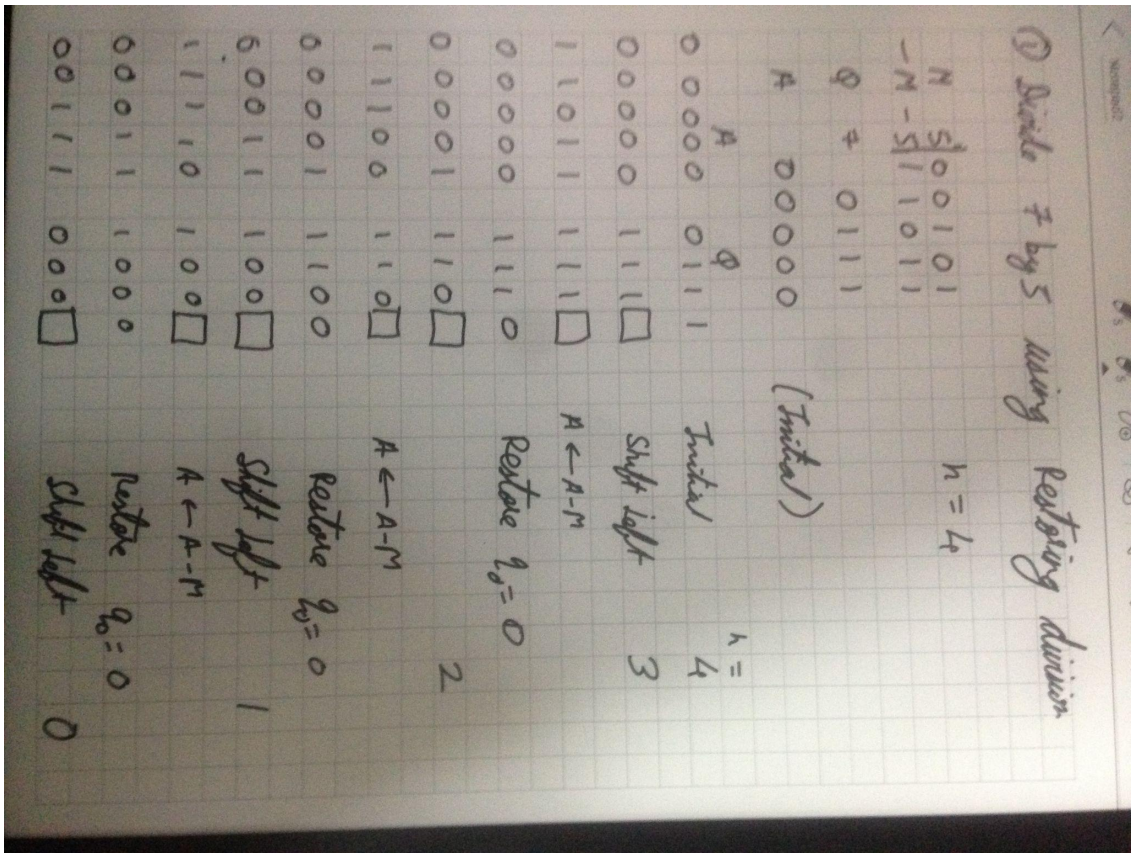
Flowchart for Restoring of Division:

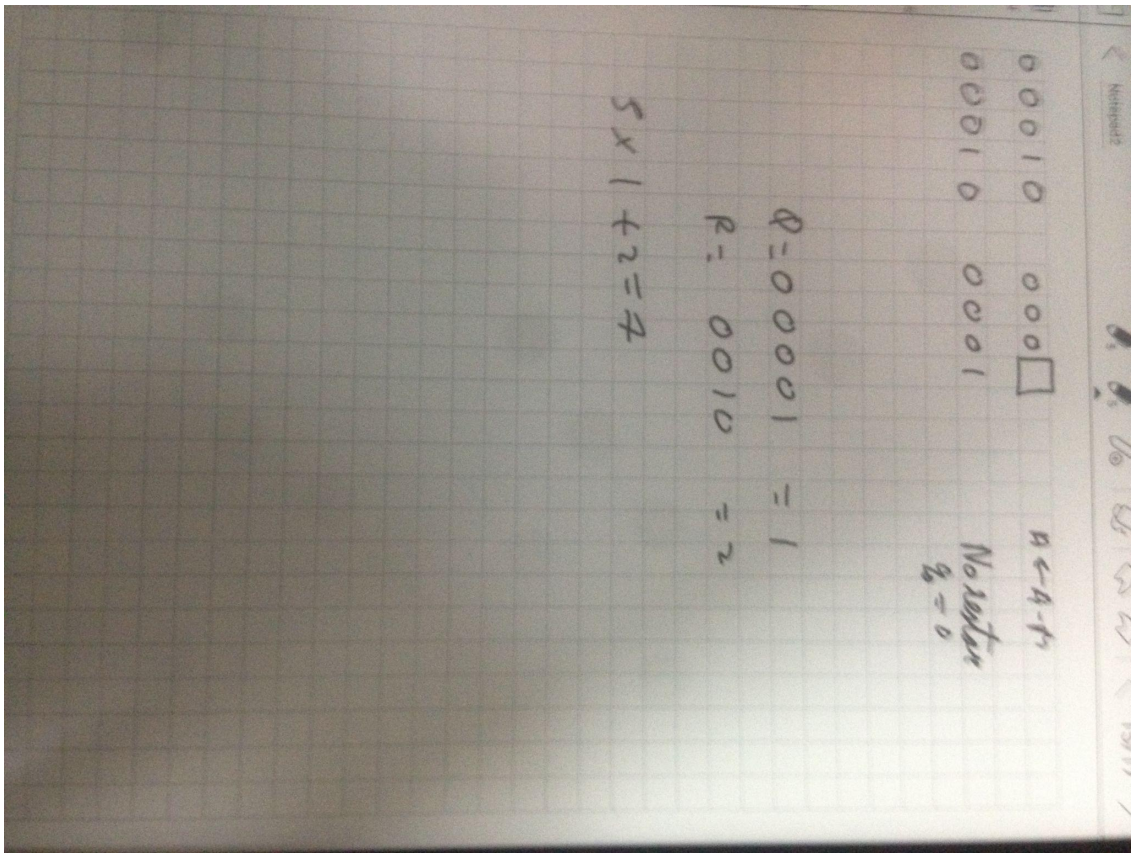


Design Steps:

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. Q_0=0 & store present A
9. Else Q_0=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)


① Divide 7 by 5 using Restoring division
h = 4
N 5 00101
-M -5 11011
Q = 0111
A 00000 (Initial)
A Q h =
0000 0111 Initial 4
00000 1110 Shift left 3
11011 1110 A ← A-M
00000 1110 Restore q0 = 0 2
00001 1100 A ← A-M
11100 1100 Restore q0 = 0
00001 1000 Shift left 1
11110 1000 A ← A-M
00011 0001 Restore q0 = 0
00111 0000 Shift left 0



Conclusion:

Thus we have understood how restoring division works. Restoring division works using addition and subtraction using addition of complement. Using this method, we can divide two numbers within a given bit range. Example for 4 bits, we can divide two 3 bit numbers. (From 1-7)

Post Lab Descriptive Questions

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

One extra bit must be maintained in the partial remainder to keep track of the sign in case of non restoring division. This overhead is not present in restoring division.

Date: 29 sept 22 **Signature of faculty in-charge**

Batch: B2 Roll No.: 16010121110

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)
Describe and define the structure of a computer with buses structure and detail working of the arithmetic logic unit and its sub modules

Books/ Journals/ Websites referred:

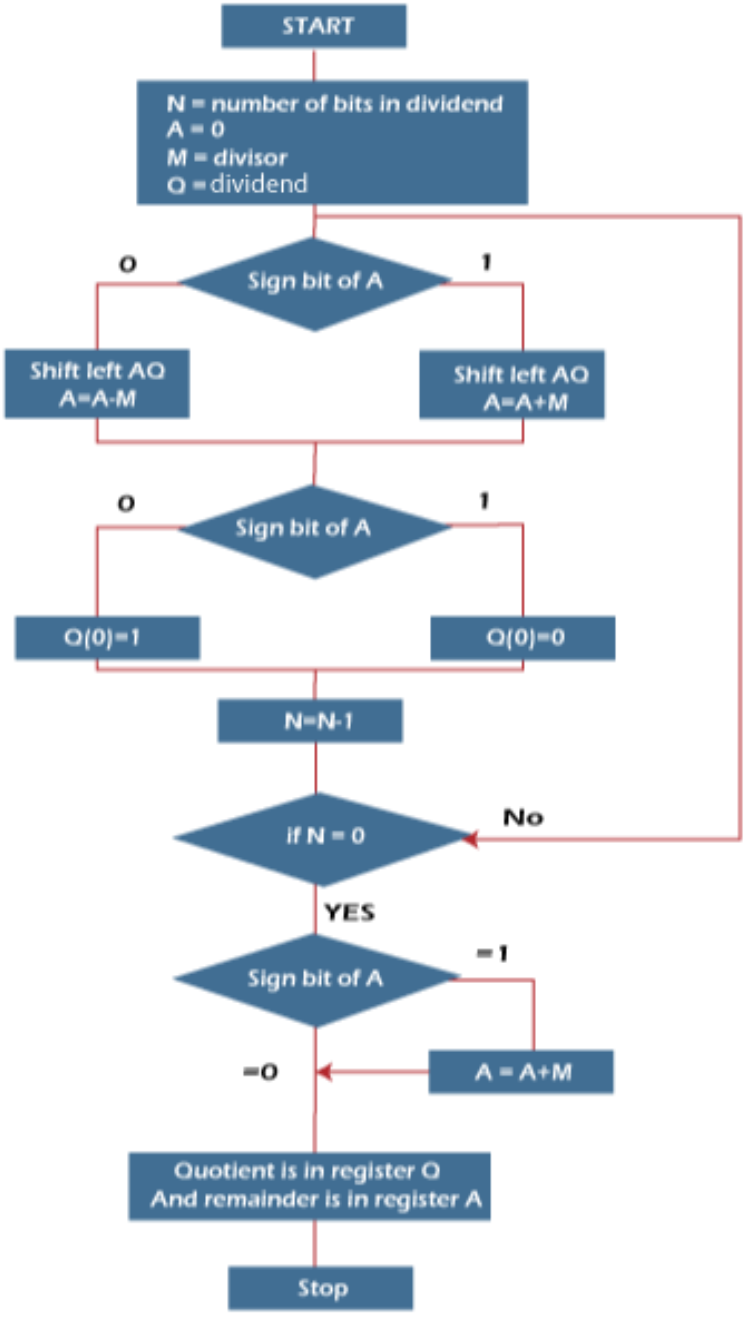
3. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.
4. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.

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

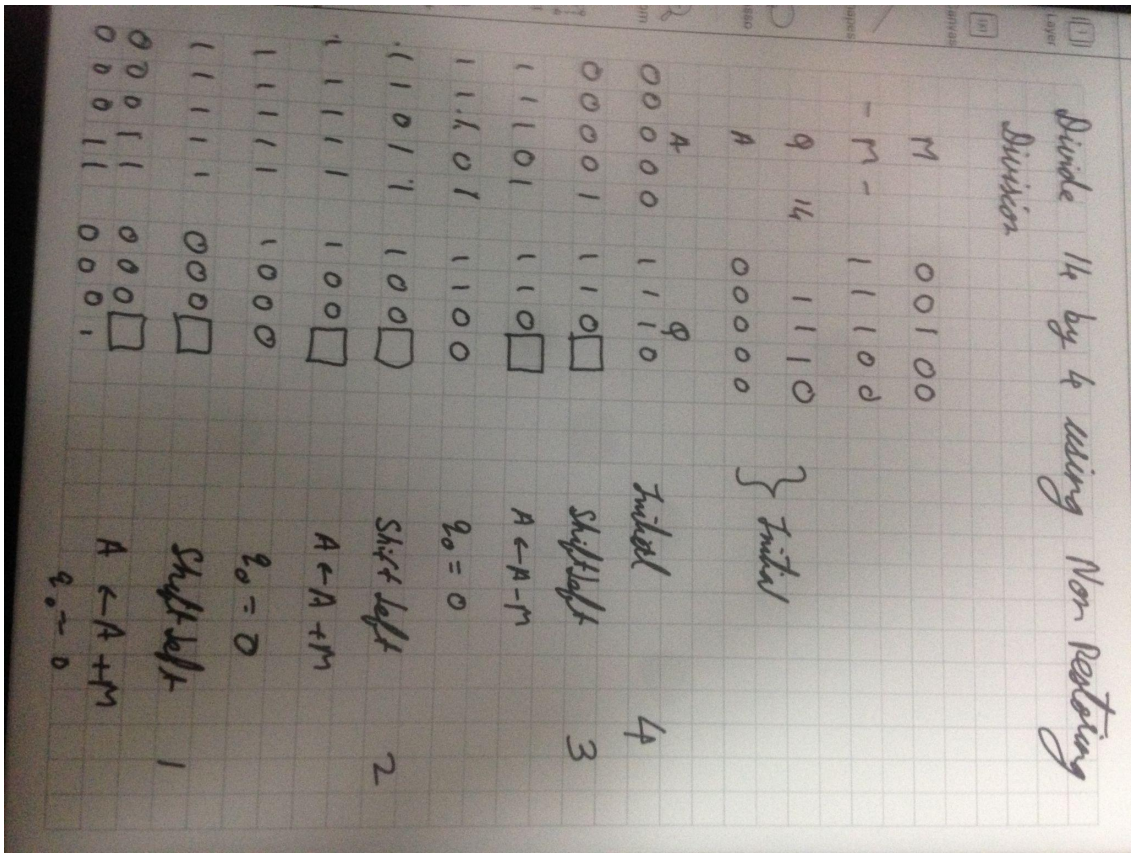
Pre Lab/ Prior Concepts:

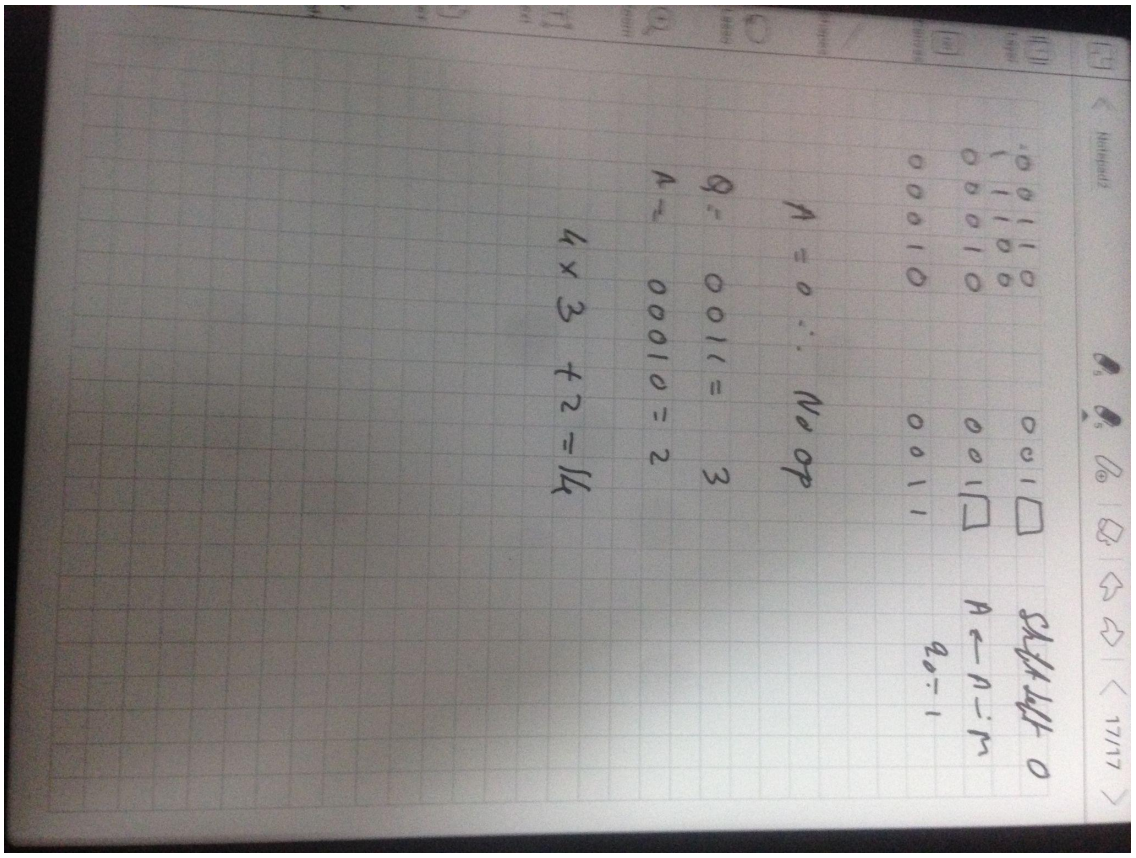
The Non Restoring algorithm works with any combination of positive and negative numbers.

Flowchart for Non Restoring of Division:



Example: (Handwritten solved problem needs to uploaded)


Divide 14 by 4 using Non Restoring Division
M 00100
-M- 11100
Q 14 1110
A 00000
A Q
00000 1110 Initial 4
00001 110□ Shift left 3
11101 110□ A←A-M
11101 1100 Q0 = 0
11101 100□ Shift left 2
11111 100□ A←A+M
11111 1000 Q0 = 0
11111 000□ Shift left 1
00001 000□ A←A+M
00001 0000 Q0 = 0



Conclusion

Thus we have understood the working behind the non-restoring algorithm.We implemented the code and handwritten working of the algoprithm.

In the non restoring algorithm we do not require to restore the bits. Instead we use the sign bit to determine if addition or subtraction is required. This requires one less addition.

Post Lab Descriptive Questions

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

The advantage of using non-restoring arithmetic over the standard restoring division is that a test subtraction is not required; the sign bit determines whether an addition or subtraction is used.

Date: 29 sept 22 **Signature of faculty in-charge**

Batch: B2 Roll No.: 16010121110

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-7

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 for floating-point computation 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 portably. Many hardware floating point units now use the IEEE 754 standard.

The standard defines:

- *arithmetic formats:* sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs)

- *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) on arithmetic formats

- *exception handling:* indications of exceptional conditions (such as division by zero, overflow, *etc*

```
def convert_to_binary_integer(num):
    arr = []
    while (num != 1):
        arr.append(round(num % 2))
        num = num - num % 2
        num /= 2
    arr.append(1)
    arr = arr[::-1]
    return arr
import math
def IEEE(num):
      signbit=[0]
      if(num<0):
            signbit=[1]
            num=num*-1
      intpart = convert_to_binary_integer(math.floor(num)) #numbers
before decimal
      intpart.pop(0) #remove first 1
           floatpart=  convert_to_binary_float(num-math.floor(num))
#numbers after decimal
      mantissa=intpart+floatpart
      exponent= len(intpart)
      exponent=exponent+127 #bais
      exponent=convert_to_binary_integer(exponent)
      return signbit+exponent+mantissa
def convert_to_binary_float(num):
    arr=[]
```

```
    for i in range(0,23):
        num=num*2
        if(num>=1):
            arr.append(1)
            num=num-1
        else:
            arr.append(0)
    return arr

print(IEEE(20.57))
```

Example (Single Precision- 32 bit representation)

Decimal Number 23.34
32 bit representation
signbit [0]
exponent [1, 0, 0, 0, 0, 0, 1, 1]
mantissa [0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]

Decimal Number 25.657
32 bit representation
signbit [0]
exponent [1, 0, 0, 0, 0, 0, 1, 1]
mantissa [1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]

Example (Double Precision- 64 bit representation)

Decimal Number 23.34
64 bit representation
signbit [0]
exponent [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
mantissa [0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0]

Decimal Number 25.657
64 bit representation
signbit [0]
exponent [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
mantissa [1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]

Post Lab Descriptive Questions

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

IEEE developed the IEEE 754 floating-point standard. This standard defines set formats and operation modes. All computers conforming to this standard would always calculate the same result for the same computation. This standard does not specify arithmetic procedures and hardware to be used to perform computations. It is important

because it is a standard way of representing numbers.

Conclusion:

Thus we have converted binary numbers to IEEE format for floating point representation. The IEEE-754 standard describes floating-point formats, a way to represent real numbers in hardware. 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.

Date: 20 oct 22 **Signature of faculty in-charge**

Batch: B2 Roll No.: 110
Experiment / assignment / tutorial No.____6___
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)

CO3- Learn and evaluate memory organization and cache structure

---

Books/ Journals/ Websites referred:

3. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, TataMcGraw-Hill.

4. 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:

```
[1]
[2, 1]
[3, 2, 1]
[5, 3, 2, 1]
[1, 5, 3, 2]
[4, 1, 5, 3]
[4, 1, 5, 3]
2
```

```
SIZE = 4
memory = []
hit = 0


def add(a):
    global hit

    if (len(memory) == SIZE):

        if (a in memory):
            hit = hit + 1
            memory.remove(a)
        else:
            memory.pop(-1)
```

```
    memory.insert(0, a)
    print(memory)


add(1)
add(2)
add(3)
add(5)
add(1)
add(4)
add(4)
print(hit)
```

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?

Hit rate = number of times page was present in cache / total number of accesses

miss ratio = number of times page was not present in cache / total number of accesses

2. What is the need for virtual memory?

Virtual memory is a technique by which memory space can be increased without physically increasing the memory size. By using virtual memory we can utilize the space in Secondary memory for Primary memory requirements

Conclusion

Thus we understood what LRU algorithm is. It replaces the least recently used page in the virtual memory with the new page. Using this algorithm we can manage pages in Cache or virtual memory.

Date: _____ **Signature of faculty in-charge**

Batch: B2 Roll No.: 110
Experiment / assignment / tutorial No.___7___
Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of FIFO 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)

CO3- Learn and evaluate memory organization and cache structure

---

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:

[1]

[2, 1]

[3, 2, 1]

[5, 3, 2, 1]

[5, 3, 2, 1]

[4, 5, 3, 2]

[4, 5, 3, 2]

2

SIZE = 4

memory = []

```
hit = 0



def add(a):
    global hit

    if (len(memory) != SIZE):
        memory.insert(0, a)
        print(memory)
        return
    if (a in memory):
        hit += 1
        print(memory)
        return
    else:
        memory.insert(0, a)
        memory.pop(-1)
        print(memory)



add(1)
add(2)
add(3)
add(5)
```

```
add(1)
add(4)
add(4)
print(hit)
```

Post Lab Descriptive Questions

1. What is meant by memory interleaving?

Interleaved memory is designed to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory by spreading memory addresses evenly across memory banks. In this way, contiguous memory reads and writes use each memory bank, resulting in higher memory throughput due to reduced waiting for memory banks to become ready for the operations.

ref- https://www.javatpoint.com/what-is-interleaved-memory

Memory interleaving is Dividing the memory into different banks for parallel accessing.

2. Explain Paging Concept?

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging. The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory.

ref- https://www.geeksforgeeks.org/paging-in-operating-system/

Conclusion

Thus we have understood FIFO algorithm in page replacement. By FIFO algorithm we replace the page which has come in first by the new required page. It is lesser efficiane than LRU in some cases. Stacks can be used to implement this algorithm.

Date: 24 NOV 22 **Signature of faculty in-charge**

Batch:	B2	**Roll No.:**	110

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)

CO3- Learn and evaluate memory organization and cache structure

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)

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

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 = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w} / 2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s-r) tags

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

Size of Tag = log2(total memory/word size)
Address length = Tag+word size

Set Associative Mapping Implementation:

m = v * k

i= j mod v

where

i=cache set number

j=main memory block number

v=number of sets

m=number of lines in the cache number of sets

k=number of lines in each set

ref - https://www.geeksforgeeks.org/cache-memory-in-computer-organization/

No of sets = length of memory / k

Implementation code: (direct mapping)

```
Memorysize=int(input('please enter Memory size in k words  '))*1024


cachesize=int(input('please enter cache size in k words '))*1024


import math

tag=math.log2(Memorysize/cachesize)

block=math.log2(Memorysize)

word=Memorysize/cachesize /block


print("tag ",tag)

print("block ",block)

print("word ",word)
```

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.

Tag; holds the tag address of the memory location.
Block: holds the block number:
Word: holds the word number in the block

The memory is divided into tage, then tags are divided into blocks which are further divided into words. The block length is the length of the main memory.

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

Access time is increased, cost reduces, capacity increases.

Conclusion:

Thus we have understood the cache mapping techniques. We understood the set associative mapping, associative mapping and the direct mapping technique. All of these techniques have their own drawbacks and advantages. Set associative mapping sits midway between direct and associative mapping.

Date: 24 nov 22 **Signature of faculty in-charge**

Batch:	B2	Roll No.:	110

Experiment / assignment / tutorial No._____

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) Carry (CF)	All
ADC	Add with Carry	ADC D, S	(S) + (D) +(CF) (D) Carry (CF)	All

Syntax: ADD destination,source

SUB instruction:

Mnemonic	Meaning	Format	Operation	Flags Affected

SUB	Subtract	SUB D, S	(D) - (S) (D) Borrow (CF)	All
SBB	Subtract with Borrow	SBB D, S	(D) - (S) –(CF) (D)	All

MUL instruction:
Syntax: MUL source

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

DIV instruction:

Division (DIV or IDIV)	**Dividend**	**Operand (Divisor)**	**Quotient : Remainder**
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

2) 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.

3) Linking

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

The command is

TLINK<File name.OBJ>

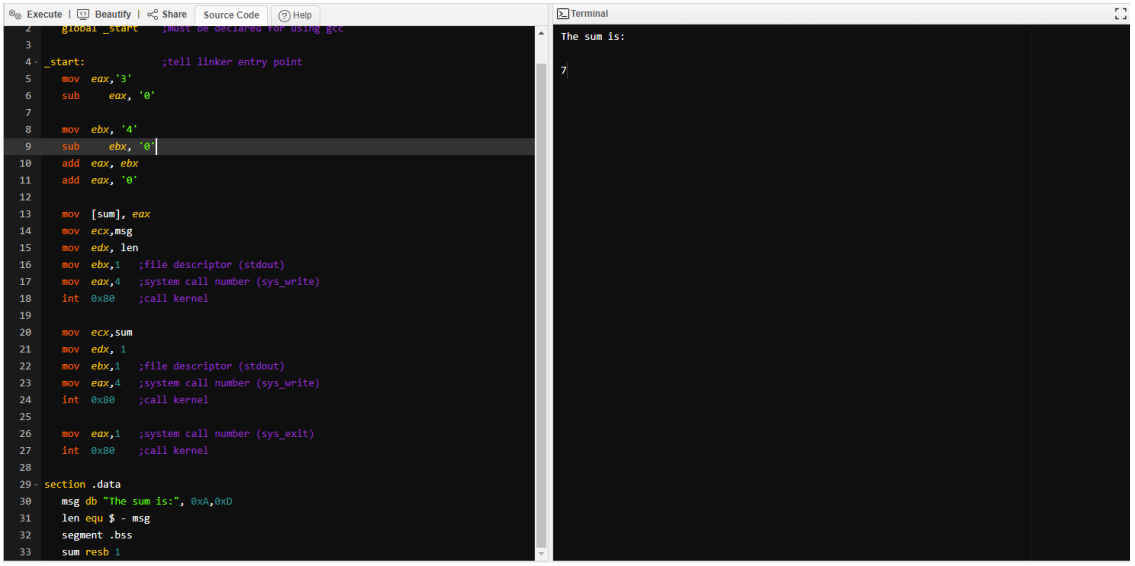
This command results in <*Filename.exe*>which can be executed in front of the command prompt.

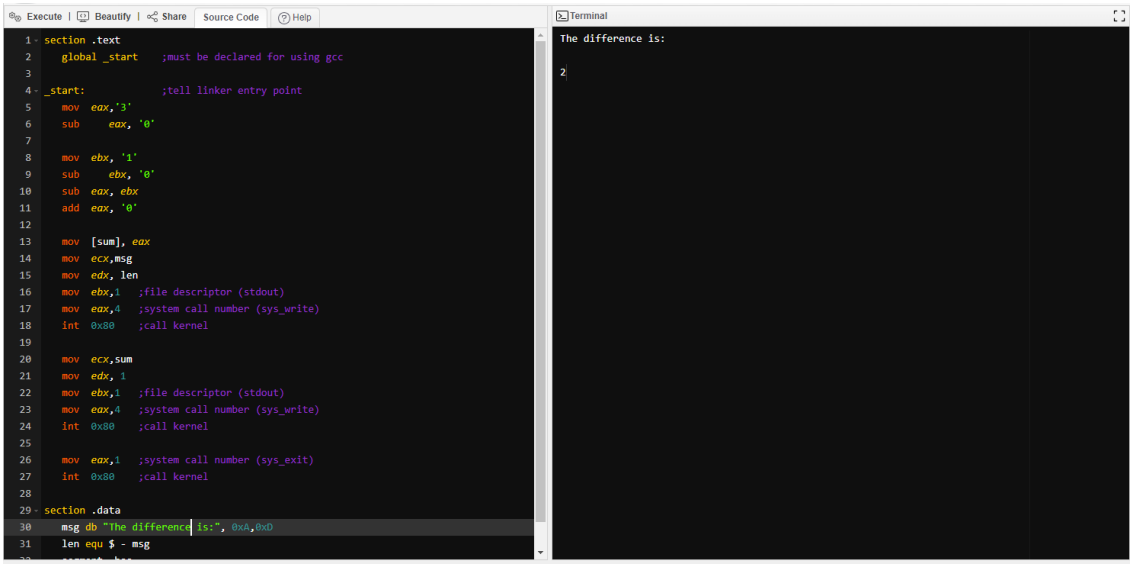
4) Executing the Program

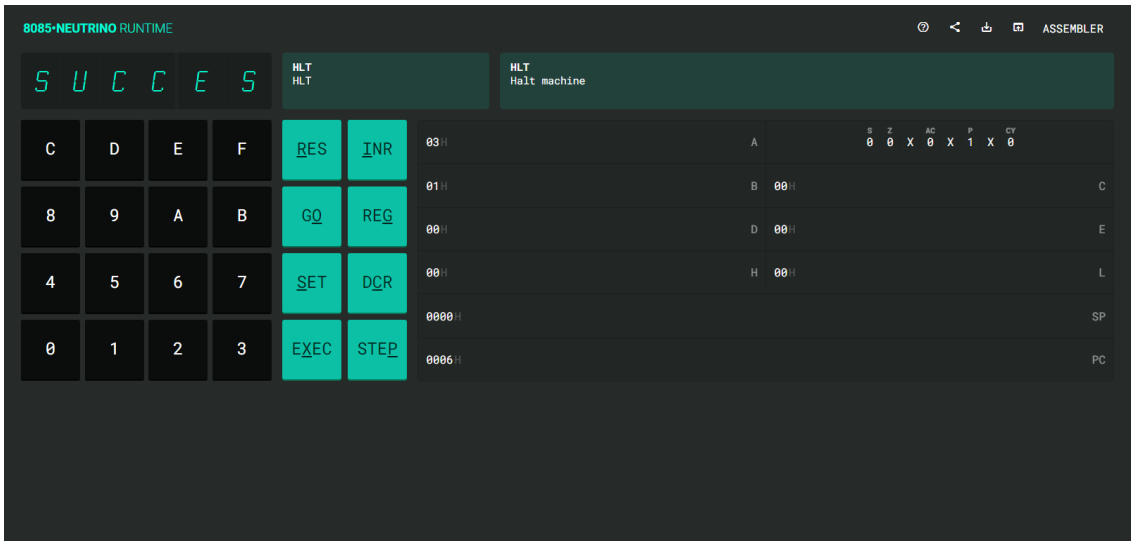
Open the program in debugger by the command(note only exe files can be open)by the command.

<*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)


Execute | Beautify | Share Source Code Help
global _start ;must be declared for using gcc
_start: ;tell linker entry point
mov eax,'3'
sub eax, '0'
mov ebx, '4'
sub ebx, '0'
add eax, ebx
add eax, '0'
mov [sum], eax
mov ecx,msg
mov edx, len
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov ecx,sum
mov edx, 1
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov eax,1 ;system call number (sys_exit)
int 0x80 ;call kernel
section .data
msg db "The sum is:", 0xA,0xD
len equ $ - msg
segment .bss
sum resb 1
Terminal
The sum is:
7


Execute | Beautify | Share Source Code Help
section .text
global _start ;must be declared for using gcc
_start: ;tell linker entry point
mov eax,'3'
sub eax, '0'
mov ebx, '1'
sub ebx, '0'
sub eax, ebx
add eax, '0'
mov [sum], eax
mov ecx,msg
mov edx, len
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov ecx,sum
mov edx, 1
mov ebx,1 ;file descriptor (stdout)
mov eax,4 ;system call number (sys_write)
int 0x80 ;call kernel
mov eax,1 ;system call number (sys_exit)
int 0x80 ;call kernel
section .data
msg db "The difference is:", 0xA,0xD
len equ $ - msg
Terminal
The difference is:
2



Algorithm for adding the two 8-bit numbers:

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

Algorithm for subtracting the two 8 bit numbers:

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Sub the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

Algorithm for Subtracting

1) Take twos complement
2) add the numbers
3) if first bit is 1 take twos complement again

Algorithm for multiplying the two 8 bit numbers:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry
in memory location.
8) Terminate the program.

Algorithm for dividing the two 8-bit numbers:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register(B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry .
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

References - **https://www.mpuat.ac.in/images/editorFiles/file/E-Content/Microprocessor%20Programming%20Lab%20Manual.pdf**

Conclusion:

Thus we have understood the working of assembly language. We performed addition and subtraction of two 8 bit numbers using assembly language and understood the constraints and logic of the working.

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

Explain instructions ADC and SBB with example

ADC is add with carry and SBB is subtract with borrow.

ADC adds two numbers and adds carry if present as denoted by the carry flag.

SBB subtracts the two numbers with borrow.

Example ADC A,B or SBB A,C

Date: _____ **Signature of faculty in-charge**

Batch:	**B2**	**Roll No.:**	**16010121110**

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:

Books/ Journals/ Websites referred:

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

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.

1	Binary Rep	0101
2	2's Comp	5
3	1's Comp	5
4	Unsigned	5
5	Signed	5

#	Binary	2's Comp	1's Comp	Unsigned	Signed
1	0000	0	0	0	0
2	0001	1	1	1	1
3	0010	2	2	2	2
4	0011	3	3	3	3
5	0100	4	4	4	4
6	0101	5	5	5	5
7	0110	6	6	6	6
8	0111	7	7	7	7
9	1000	-8	-7	8	-0
10	1001	-7	-6	9	-1
11	1010	-6	-5	10	-2
12	1011	-5	-4	11	-3
13	1100	-5	-3	12	-4

Observations

Title of Study Experiment:

Representation of Integers and their Arithmetic

Brief description of experiment under study:

The objective of this experiment is to learn about the representation of signed and unsigned integers. Specifically, we will learn the advantages of using 2's complement representation of numbers for performing arithmetic over other representations.

Learning's recorded:

Integers are primitive data structures provided by default by almost all processors and programming languages. In this lab, we studied the representation of integers and the arithmetic operations on them.

Knowledge gained / Inference Obtained :

We learnt how to perform binary arithmetic. We learnt to add two numbers in binary.

Post Lab Descriptive Questions

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

This vlab can be used to learn binary addition. But the vlab doesn't show how the addition is done using any animation. This can be added to the vlab.

Conclusion

Thus we have understood the working of binary arithmetic, specifically binary addition. We used virtual labs for the performance of this experiment. We added two binary numbers.

Date: 30 nov 22 **Signature of faculty in-charge**