Module 4
Data Representation and Arithmetic Algorithms

Number representation: Binary Data representation, two's complement representation and Floating-point representation.Integer Data arithmetic: Addition, Subtraction. Multiplication: Unsigned & Signed multiplication-Add & Shift Method,Booth's algorithm.Division of integers:Restoring and non-restoring division, signed division, basics of floating point representation IEEE 754 floating point(Single & double precision) number representation.Floating point arithmetic:Addition,subtraction

The ALU is that part of the computer that actually performs arithmetic and logical operations on data. All of the other elements of the computer system—control unit, registers, memory, I/O—are there mainly to bring data into the ALU for it to process and then to take the results back out.
Figure 4.1 indicates, in general terms, how the ALU is interconnected with the rest of the processor.

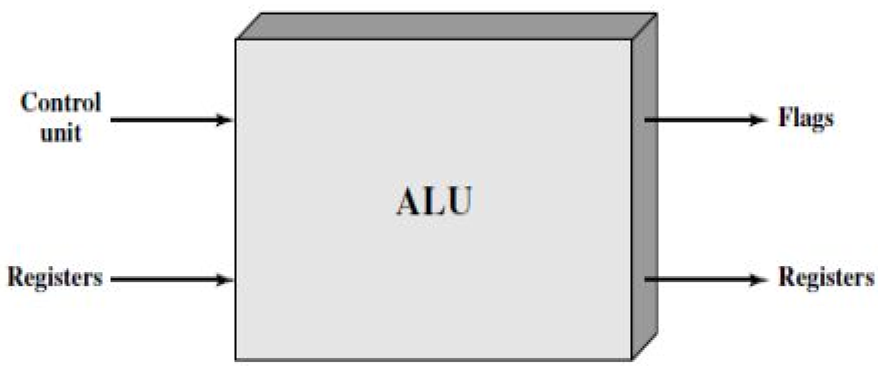


Figure 4.1 ALU Inputs and Outputs

Data are presented to the ALU in registers, and the results of an operation are stored in registers.These registers are temporary storage locations within the processor that are connected by signal paths to the ALU.The ALU may also set flags as the result of an operation.

For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor.The control unit provides signals that control the operation of the ALU and the movement of the data into and out of the ALU.

INTEGER REPRESENTATION

In the binary number system, arbitrary numbers can be represented with just the digits zero and one, the minus sign, and the period, or radix point.

$$-1101.0101_2 = -13.3125_{10}$$

For purposes of computer storage and processing, however, we do not have the benefit of minus signs and periods. Only binary digits (0 and 1) may be used to represent numbers. If we are limited to nonnegative integers, the representation is straightforward.

An 8-bit word can represent the numbers from 0 to 255, including

$$\begin{aligned} 00000000 &= 0 \\ 00000001 &= 1 \\ 00101001 &= 41 \\ 10000000 &= 128 \\ 11111111 &= 255 \end{aligned}$$

Sign-Magnitude Representation

To represent negative or positive integers, the most significant (leftmost) bit in the word is used as a sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative.The simplest form of representation that employs a sign bit is the sign-magnitude representation.

+ 18 = 00010010

- 18 = 10010010 (sign magnitude)

Drawbacks in Sign-Magnitude Representation

One is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.
Another drawback is that there are two representations of 0:

+0=**0**000000
-0=**1**000000 (sign magnitude)

This is inconvenient because it is slightly more difficult to test for 0 (an operation performed frequently on computers) than if there were a single representation. (+0=00000 and -0=000000)
Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU.
Instead, the most common scheme is twos complement representation

Twos Complement Representation

Twos complement representation also uses the most significant bit as a sign bit, to test whether an integer is positive or negative.
For positive number the MSB bit is 0 and remaining bits represent the magnitude of the number in the same fashion as for sign magnitude
For Eg +18==00010010
For negative number ,the MSB bit is 1 (a_{n-1}=1)and remaing bits are calulated as term number - 2^{n-1}
For Eg -18
For n=8 ,

MSB bit=>> a_{n-1} =>a_7=1
Remaining bits(a_6 to a_0) => 2^{n-1} -term number
=>128-18
=>110
=>1101110 (two's complement of 18)

Example 2
For a number 7,with n=4 ,where n is number of bits
Two complement positive and negative is calculated as
Positive 7

a_{n-1} =>a_3=0
remaining bits(a_2 ,a_1, a_0) =same as sign magnitude(111)
a_3=0,a_2=1,a_1=1 a_0=1 =>0111

Negative 7

$a_{n-1} \Rightarrow a_3=1$
$a_2, a_1, a_0 = 2^{n-1}$ - term number
$a_2, a_1, a_0 = 2^{n-1} - 7$
$a_2, a_1, a_0 = 2^{4-1} - 7$
$a_2, a_1, a_0 = 2^3 - 7$
$a_2, a_1, a_0 = 8 - 7$
$a_2, a_1, a_0 = 1$
$a_2, a_1, a_0 = 001$
$a_3=1, a_2=0, a_1=0\ a_0=1$
=>1001(two's complement of 7)

Table 4.1 compares the sign-magnitude and twos complement representations for 4-bit integers

Decimal Representation	Sign-Magnitude Representation	TwosComplement Representation	Biased Representation
+ 8	—	—	1111
+ 7	0111	0111	1110
+ 6	0110	0110	1101
+ 5	0101	0101	1100
+ 4	0100	0100	1011
+ 3	0011	0011	1010
+ 2	0010	0010	1001
+ 1	0001	0001	1000
+ 0	0000	0000	0111
- 0	1000	—	—
- 1	1001	1111	0110
- 2	1010	1110	0101
- 3	1011	1101	0100
- 4	1100	1100	0011
- 5	1101	1011	0010
- 6	1110	1010	0001
- 7	1111	1001	0000

- 8	—	1000	—

In **sign magnitude** there are two representations for 0 (+0->0000 and -0 -> 1000) ,and equal number of positive and negative integers are represented (+7 to -7)

In **twos complement** there is only one representation for 0 (+0->0000 and -0 -> 0000) ,and unequal number of negative and positive numbers represented.
For example, for an n-bit length (4), there is a representation for -2n-1(-8) but not for +2n-1 (+8).

In **biased representation** a fixed value, called the bias, is subtracted from the field to get the true exponent value. Typically, the bias equals (2^{k-1} - 1), where *k* is the number of bits in the binary exponent.
In this case, the 4-bit field yields the numbers 0 through 15. With a bias of 7 (2^3 - 1), the true exponent values are in the range - 7 to + 8. In this example, the base is assumed to be 2.

INTEGER ARITHMETIC

This section examines common arithmetic functions on numbers in twos complement representation.

Negation

In sign-magnitude representation, the rule for forming the negation of an integer is simple: invert the sign bit.
In twos complement notation, the negation of an integer can be formed with the following rules:

1. Take the Boolean complement of each bit of the integer (including the sign bit). That is, set each 1 to 0 and each 0 to 1.
2. Treating the result as an unsigned binary integer, add 1.

This two-step process is referred to as the twos complement operation, or the taking of the twos complement of an integer.

$$
\begin{aligned}
+18 &= 00010010 \text{ (twos complement)} \\
\text{bitwise complement} &= 11101101 \\
&\underline{+ \qquad\quad 1} \\
& 11101110 = -18
\end{aligned}
$$

As expected, the negative of the negative of that number is itself:

$$
\begin{aligned}
-18 &= 11101110 \text{ (twos complement)} \\
\text{bitwise complement} &= 00010001 \\
&\underline{+ \qquad\quad 1} \\
& 00010010 = +18
\end{aligned}
$$

Rules for Addition and Subtraction in twos complement

A. Addition proceeds as if the two numbers were unsigned integers.
B. If the result of the operation is positive, we get a positive number in twos complement form, which is the same as in unsigned-integer form.
C. If the result of the operation is negative, we get a negative number in twos complement form.
Addition in twos complement is illustrated in Figure.4.2

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

Figure 4.2 Addition of Numbers in Twos Complement Representation

The first four examples illustrate successful operations.

Note that, in some instances, there is a carry bit beyond the end of the word (indicated by shading), which is ignored.

On any addition, the result may be larger than can be held in the word size being used.

This condition is called overflow.

When overflow occurs, the ALU must signal this fact so that no attempt is made to use the result.

To detect overflow, the following rule is observed:

OVERFLOW RULE: If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign

SUBTRACTION RULE: To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend. Thus, subtraction is achieved using addition, as illustrated in Figure 4.3 The last two examples demonstrate that the overflow rule still applies.

Thus, subtraction is achieved using addition, as illustrated in Figure below. The last two examples demonstrate that the overflow rule still applies.

0010 = 2 +1001 = −7 1011 = −5 (a) M = 2 = 0010 S = 7 = 0111 −S = 1001	0101 = 5 +1110 = −2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7 (c) M = −5 = 1011 S = 2 = 0010 −S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow (e) M = 7 = 0111 S = −7 = 1001 −S = 0111	1010 = −6 +1100 = −4 10110 = Overflow (f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Figure 4.3 Subtraction of Numbers in Twos Complement Representation (M - S)

Figure 4.4 suggests the data paths and hardware elements needed to accomplish addition and subtraction. The central element is a binary adder, which is presented two numbers for addition and produces a sum and an overflow indication. The binary adder treats the two numbers as unsigned integers.For addition, the two numbers are presented to the adder from two registers, designated in this case as A and B registers. The result may be stored in one of these registers or in a third. The overflow indication is stored in a 1-bit overflow flag (0 = no overflow; 1 = overflow).

For subtraction, the subtrahend (B register) is passed through a twos complementer so that its twos complement is presented to the adder. Note that Figure 4.4 only shows the data paths. Control signals are needed to control whether or not the complementer is used, depending on whether the operation is addition or subtraction.

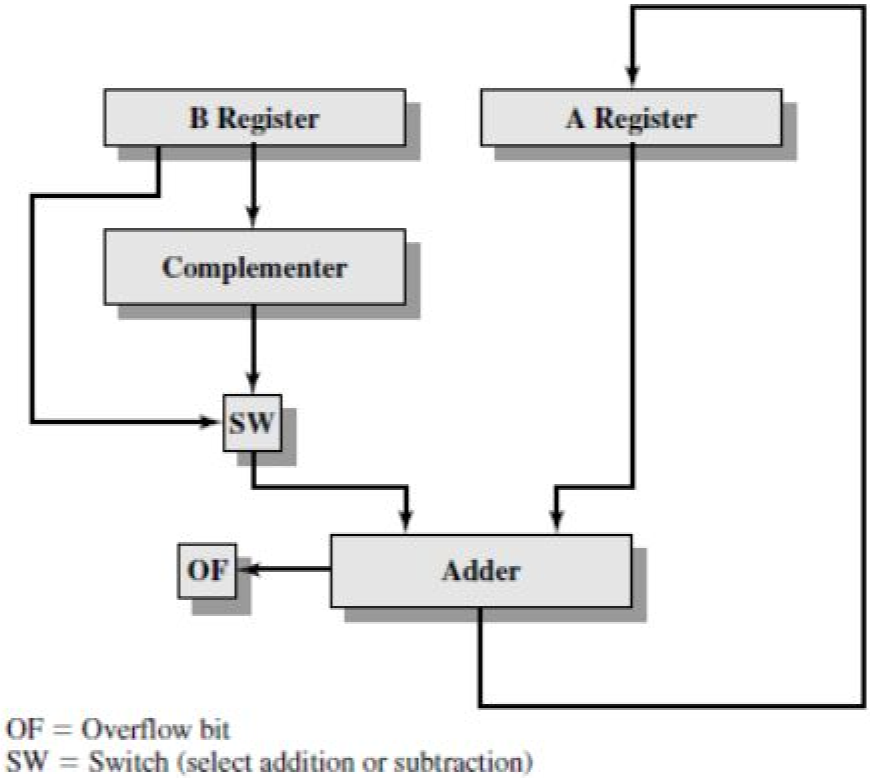


Figure 4.4 Block Diagram of Hardware for Addition and Subtraction

Multiplication

Compared with addition and subtraction, multiplication is a complex operation, whether performed in hardware or software.

UNSIGNED INTEGERS

Figure 4.5 illustrates the multiplication of unsigned binary integers, as might be carried out using paper and pencil.

Several important observations can be made:

1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.

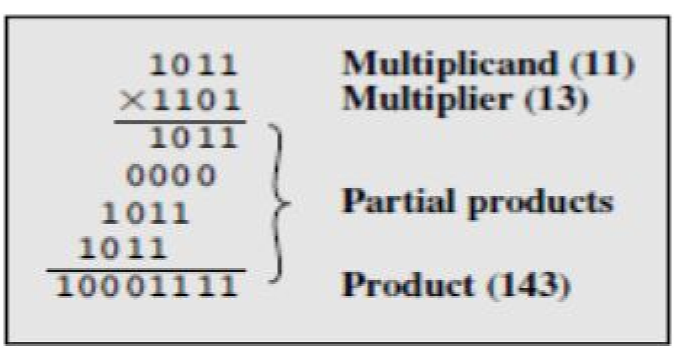


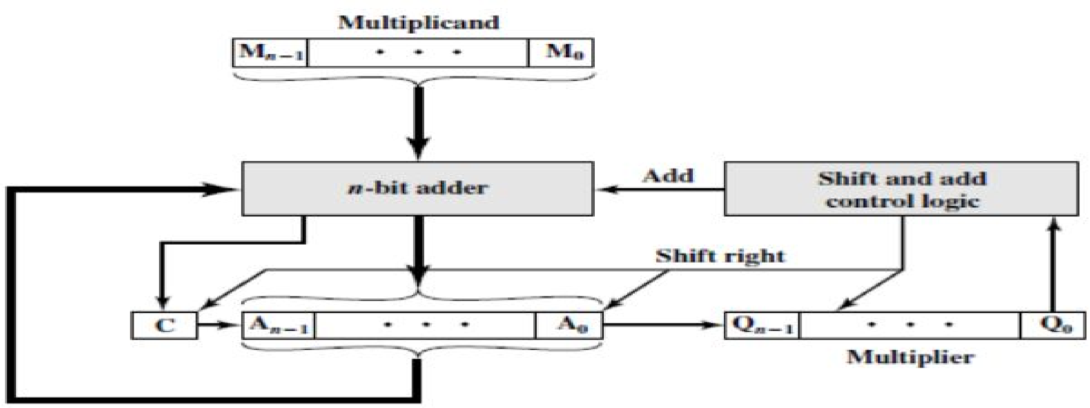
Figure 4.5 Multiplication of Unsigned Binary Integers

2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0.When the multiplier is 1, the partial product is the multiplicand
3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.
4. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length (e.g.,11*11=1001).

Compared with the pencil-and-paper approach, there are several things we can do to make computerized multiplication more efficient.

First, we can perform a running addition on the partial products rather than waiting until the end. This eliminates the need for storage of all the partial products; fewer registers are needed.

Second, we can save some time on the generation of partial products. For each 1 on the multiplier, an add and a shift operation are required; but for each 0, only a shift
is required.Figure shows a possible implementation employing these measures.



(a) Block diagram

C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth cycle
0	1000	1111	1011	Shift	

Figure 4.6 Hardware Implementation of Unsigned Binary Multiplication

The multiplier and multiplicand are loaded into two registers (Q and M). A third register, the A register, is also needed and is initially set to 0. There is also a 1-bit C register, initialized to 0, which holds a potential carry bit resulting from addition.
The operation of the multiplier is as follows.
Control logic reads the bits of the multiplier one at a time.

1. If Q_0 is 1, then the multiplicand is added to the A register and the result is stored in the A register, with the C bit used for overflow.
2. Then all of the bits of the C, A, and Q registers are shifted to the right one bit, so that the C bit goes into A_{n-1} , A_0 goes into Q_{n-1} and Q_0 is lost.
3. If Q_0 is 0, then no addition is performed, just the shift.

This process is repeated for each bit of the original multiplier.
The resulting 2n-bit product is contained in the A and Q registers.
A flowchart of the operation and example is shown in Figure.4.7

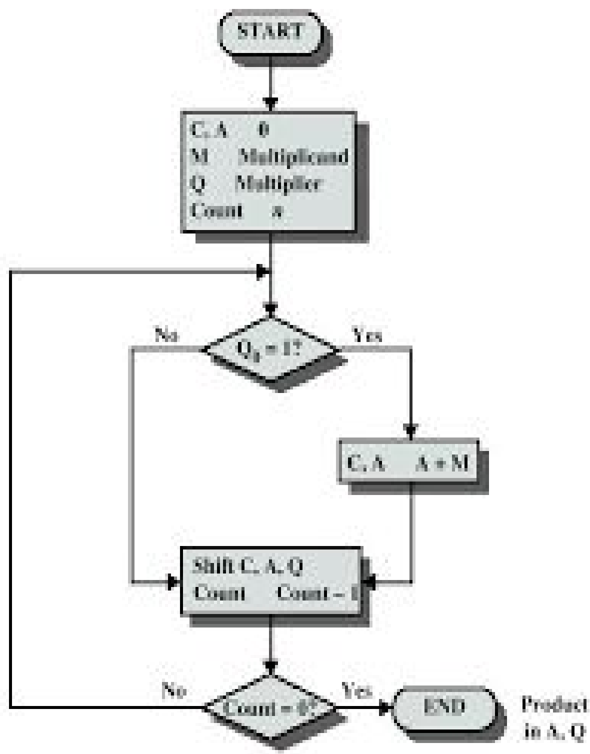


Figure 4.7 Flowchart for Unsigned Binary Multiplication

Note that on the second cycle, when the multiplier bit is 0, there is no add operation.

SIGNED INTEGERS

Booth's algorithm for twos Complement Multiplication

Booth's algorithm is depicted in Figure 4.8 and can be described as follows.
The multiplier and multiplicand are placed in the Q and M registers, respectively.
There is also a 1-bit register placed logically to the right of the least significant bit Q_0 of the Q register and designated Q_{-1}
The results of the multiplication will appear in the A and Q registers.

1. A and Q_{-1} are initialized to 0.
2. The control logic scans the bits of the multiplier one at a time.
3. Now, as each bit is examined, the bit to its right is also examined.
4. If the two bits are the same (Q_0Q_{-1}=11 orQ_0Q_{-1}= 00), then all of the bits of the A,Q, Q_{-1} and registers are shifted to the right 1 bit.
5. If the two bits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two bits are Q_0Q_{-1}=01 orQ_0Q_{-1}= 10.
6. Following the addition or subtraction, the right shift occurs.
7. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1} not only is shifted into A_{n-2} but also remains in A_{n-1}

This is required to preserve the sign of the number in A and Q.
It is known as an **arithmetic shift, because it preserves the sign bit**.

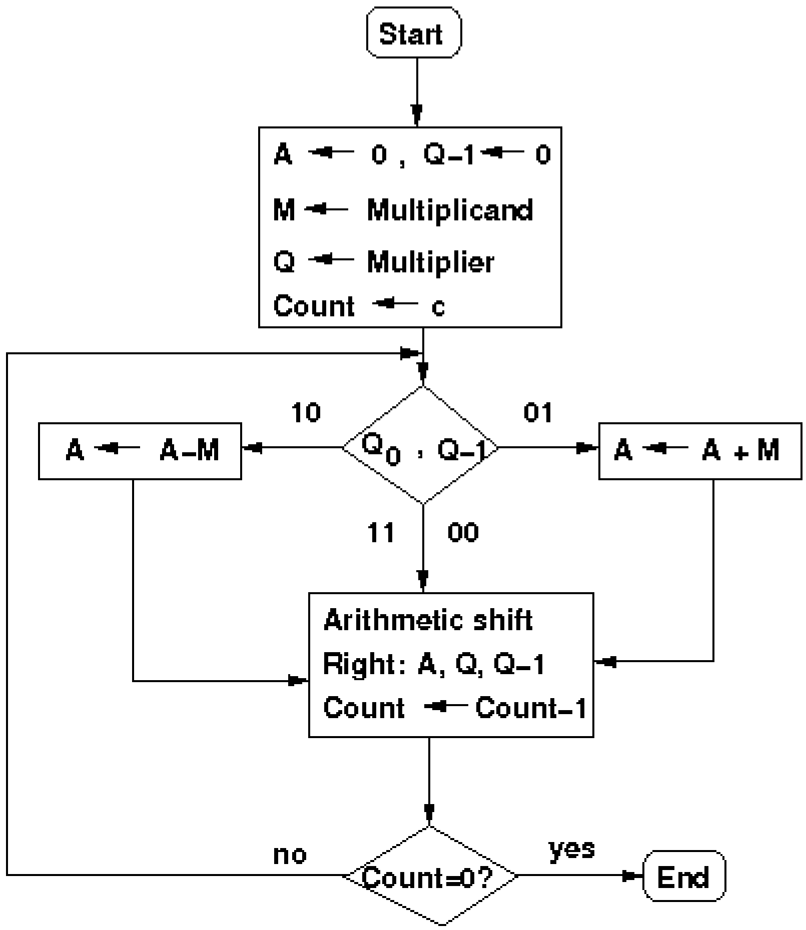


Figure 4.8 Flowchart for signed Binary Multiplication Booths algorithm

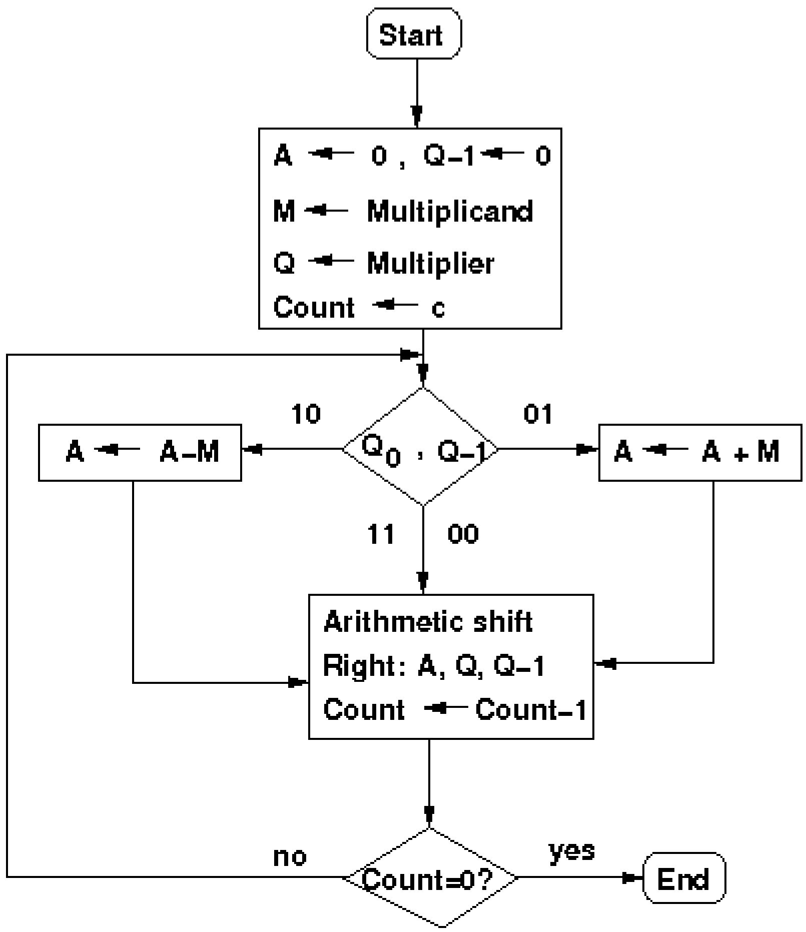
A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth cycle

Figure 4.9 Example of Booth's Algorithm (7 * 3)

Figure 4.9 shows the sequence of events in Booth's algorithm for the multiplication of 7 by 3 It performs a subtraction when the first 1 is encountered (10), an addition when (01) is encountered, and finally another subtraction when the first 1 of the next block of 1s is encountered. Thus, Booth's algorithm performs fewer additions and subtractions than a more straightforward algorithm.

Booth's Algorithm.

Draw flowchart for booths algorithm for two complement multiplication



1)Multiply (4) and (4) using Booth's Algorithm.

THE BINARY EQUIVALENT OF 4 IS : 0100

THE BINARY EQUIVALENT OF 4 IS : 0100

	OPERATION	A	Q	Q'	M
	INITIAL	0000	0100	0	0100
ROUND 1 Q_0=0 Q_{-1}=0	ARTHMETIC RIGHT SHIFT	0000	0010	0	0100
ROUND 2 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	0000	0001	0	0100
ROUND 3 Q_0=1 Q_{-1}=0	A=A-M	1100	0001	0	0100
	ARITHMETIC RIGHT SHIFT	1110	0000	1	0100
ROUND 4 Q_0=0,Q_1=1	A=A+M	0010	0000	1	0100
	ARITHMETIC RIGHT SHIFT	**0001**	**0000**	0	0100

THE ANSWER IN BINARY IS : 00010000
THE ANSWER IN DECIMAL IS : 16
----

2)Using Booth's Algorithm show the multiplication of 7x5.
May14

THE BINARY EQUIVALENT OF 7 IS : 0111

THE BINARY EQUIVALENT OF 5 IS : 0101

	OPERATION	A	Q	Q'	M
	INITIAL	0000	0101	0	0111
ROUND 1 Q_0=1 Q_{-1}=0	A=A-M	1001	0101	0	0111
	ARITHMETIC RIGHT SHIFT	1100	1010	1	0111
ROUND 2 Q_0=0 Q_{-1}=1	A=A+M(1100+0111)	0011	1010	1	0111
	ARITHMETIC RIGHT SHIFT	0001	1101	0	0111
ROUND 3 Q_0=1 Q_{--1}=0	A:=A-M (0001+1001)	1010	1101	0	0111
	ARITHMETIC RIGHT SHIFT	1101	0110	1	011
ROUND 4 Q_0=0,Q_1=1	A:=A+M (1101+0111)	0100	0110	1	0111
	ARITHMETIC RIGHT SHIFT	0010	0011	0	0111

THE ANSWER IN BINARY IS : 00100011
THE ANSWER IN DECIMAL IS : 35
----

3) multiply(4)*(-3) using Booth's Algorithm -Dec 17

THE BINARY EQUIVALENT OF 4 IS : 0100

THE BINARY EQUIVALENT OF -3 IS : 1101

	OPERATION	A	Q	Q'	M
	INITIAL	0000	1101	0	0100
ROUND Q_0=1 Q_{-1}=0	A:=A-M (0000+1100)	1100	1101	0	0100
	ARITHMETIC RIGHT SHIFT	1110	0110	1	0100
ROUND Q_0=0 Q_{-1}=1	A:=A+M (1110+0100)	0010	0110	1	0100
	ARITHMETIC RIGHT SHIFT	0001	0011	0	0100
ROUND Q_0=1 Q_{-1}=0	A:=A-M (0001+1100)	1101	0011	0	0100
	ARITHMETIC RIGHT SHIFT	1110	1001	1	0100
ROUND Q_0= 1 Q_{--1}=1	ARITHMETIC RIGHT SHIFT	1111	0100	1	0100

THE ANSWER IN BINARY IS : 11110100

THE ANSWER IN DECIMAL IS : -12

----

4)Using Booth's algorithm show the multiplication of -7 *4   Dec 16,

THE BINARY EQUIVALENT OF -7 IS : 1001
THE BINARY EQUIVALENT OF 4 IS : 0100

	OPERATION	A	Q	Q'	M
	INITIAL	0000	0100	0	1001
ROUND 1 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	0000	0010	0	1001
ROUND 2 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	0000	0001	0	1001
ROUND 3 Q_0= 1 Q_{-1}=0	A:=A-M (0000+0111)	0111	0001	0	1001
	ARITHMETIC RIGHT SHIFT	0011	1000	1	1001
ROUND 4 Q_0=0 Q_{-1}=1	A:=A+M (0011+1001)	1100	1000	1	1001
	ARITHMETIC RIGHT SHIFT	1110	0100	0	1001

THE ANSWER IN BINARY IS : 11100100
THE ANSWER IN DECIMAL IS : -28
----

5) Multiply (— 2)$_{10}$ and (— 5)$_{10}$ using Booth's Algorithm

THE BINARY EQUIVALENT OF -2 IS : 1110

THE BINARY EQUIVALENT OF -5 IS : 1011

	OPERATION	A	Q	Q'	M
	INITIAL	0000	1011	0	1110
ROUND 1 Q_0= 1 Q_{-1}=0	A:=A-M (0000+0010)	0010	1011	0	1110
	ARITHMETIC RIGHT SHIFT	0001	0101	1	1110
ROUND 2 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	0000	1010	1	1110
ROUND 3 Q_0= 0 Q_{-1}=1	A:=A+M (0000+1110)	1110	1010	1	1110
	ARITHMETIC RIGHT SHIFT	1111	0101	0	1110
ROUND 4 Q_0=1 Q_{-1}=0	A:=A-M (1111+0010)	0001	0101	0	1110
	ARITHMETIC RIGHT SHIFT	0000	1010	1	1110

THE ANSWER IN BINARY IS : 00001010

THE ANSWER IN DECIMAL IS : 10

----

6)Using Booth's algorithm show the multiplication of -3 .* -7. Dec15
THE BINARY EQUIVALENT OF -3 IS : 1101

THE BINARY EQUIVALENT OF -7 IS : 1001

	OPERATION	A	Q	Q'	M
	INITIAL	0000	1001	0	1101
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (0000+0011)	0011	1001	0	1101
	ARITHMETIC RIGHT SHIFT	0001	1100	1	1101
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (0001+1101)	1110	1100	1	1101
	ARITHMETIC RIGHT SHIFT	1111	0110	0	1101
ROUND 3 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	1111	1011	0	1101
ROUND 4 Q_0=1 Q_{-1}=0	A:=A-M (1111+0011)	0010	1011	0	1101
	ARITHMETIC RIGHT SHIFT	0001	0101	1	1101

THE ANSWER IN BINARY IS : 00010101
THE ANSWER IN DECIMAL IS : 21
----

7) Using Booth's algorithm show the multiplication of -6 .* -4

THE BINARY EQUIVALENT OF -6 IS : 1010

THE BINARY EQUIVALENT OF -4 IS : 1100

	OPERATION	A	Q	Q'	M
	INITIAL	0000	1100	0	1010
ROUND 1 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	0000	0110	0	1010
ROUND 2 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	0000	0011	0	1010
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (0000+0110)	0110	0011	0	1010
	ARITHMETIC RIGHT SHIFT	0011	0001	1	1010
ROUND 4 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	0001	1000	1	1010

THE ANSWER IN BINARY IS : 00011000

THE ANSWER IN DECIMAL IS : 24

Range 8 to 15
(Use five digits so 5 rounds)

8. Using Booth's algorithm show the multiplication of 9*9
THE BINARY EQUIVALENT OF 9 IS : 01001
THE BINARY EQUIVALENT OF 9 IS : 01001

	OPERATION	A	Q	Q'	M
	INITIAL	00000	01001	0	01001
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+10111)	10111	01001	0	01001
	ARITHMETIC RIGHT SHIFT	11011	10100	1	01001
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (11011+01001)	00100	10100	1	01001
	ARITHMETIC RIGHT SHIFT	00010	01010	0	01001
ROUND 3 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00001	00101	0	01001
ROUND 4 Q_0=1 Q_{-1}=0	A:=A-M (00001+10111)	11000	00101	0	01001
	ARITHMETIC RIGHT SHIFT	11100	00010	1	01001
ROUND 5 Q_0=0 Q_{-1}=1	A:=A+M (11100+01001)	00101	00010	1	01001
	ARITHMETIC RIGHT SHIFT	00010	10001	0	01001

THE ANSWER IN BINARY IS : 0001010001
THE ANSWER IN DECIMAL IS : 81
9. Using Booth's algorithm show the multiplication of 12 *13

THE BINARY EQUIVALENT OF 12 IS : 01100

THE BINARY EQUIVALENT OF 13 IS : 01101

	OPERATION	A	Q	Q'	M
	INITIAL	00000	01101	0	01100
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+10100)	10100	01101	0	01100
	ARITHMETIC RIGHT SHIFT	11010	00110	1	01100
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (11010+01100)	00110	00110	1	01100
	ARITHMETIC RIGHT SHIFT	00011	00011	0	01100
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (00011+10100)	10111	00011	0	01100
	ARITHMETIC RIGHT SHIFT	11011	10001	1	01100
ROUND 4 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	11101	11000	1	01100
ROUND 5 Q_0=0 Q_{-1}=1	A:=A+M (11101+01100)	01001	11000	1	01100
	ARITHMETIC RIGHT SHIFT	00100	11100	0	01100

THE ANSWER IN BINARY IS : 0010011100
THE ANSWER IN DECIMAL IS : 156

10.**Using Booth's algorithm show the multiplication of 14 X 5**
THE BINARY EQUIVALENT OF 14 IS : 01110

THE BINARY EQUIVALENT OF 5 IS : 00101

	OPERATION	A	Q	Q'	M
	INITIAL	00000	00101	0	01110
ROUND 1 $Q_0=1$ $Q_{-1}=0$	A:=A-M (00000+10010)	10010	00101	0	01110
	ARITHMETIC RIGHT SHIFT	11001	00010	1	01110
ROUND 2 $Q_0=0$ $Q_{-1}=1$	A:=A+M (11001+01110)	00111	00010	1	01110
	ARITHMETIC RIGHT SHIFT	00011	10001	0	01110
ROUND 3 $Q_0=1$ $Q_{-1}=0$	A:=A-M (00011+10010)	10101	10001	0	01110
	ARITHMETIC RIGHT SHIFT	11010	11000	1	01110
ROUND 4 $Q_0=0$ $Q_{-1}=1$	A:=A+M (11010+01110)	01000	11000	1	01110
	ARITHMETIC RIGHT SHIFT	00100	01100	0	01110
ROUND 5 $Q_0=0$ $Q_{-1}=0$	ARITHMETIC RIGHT SHIFT	00010	00110	0	01110

THE ANSWER IN BINARY IS : 0001000110
THE ANSWER IN DECIMAL IS : 70

11.**Using Booth's algorithm show the multiplication of 15 x 15**
THE BINARY EQUIVALENT OF 15 IS : 01111

THE BINARY EQUIVALENT OF 15 IS : 01111

	OPERATION	A	Q	Q'	M
	INITIAL	00000	01111	0	01111
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+10001)	10001	01111	0	01111
	ARITHMETIC RIGHT SHIFT	11000	10111	1	01111
ROUND 2 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	11100	01011	1	01111
ROUND 3 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	11110	00101	1	01111
ROUND 4 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	11111	00010	1	01111
ROUND 5 Q_0=0 Q_{-1}=1	A:=A+M (11111+01111)	01110	00010	1	01111
	ARITHMETIC RIGHT SHIFT	00111	00001	0	01111

THE ANSWER IN BINARY IS : 0011100001
THE ANSWER IN DECIMAL IS : 225

12. **Using Booth's algorithm show the multiplication of -11 x7**

THE BINARY EQUIVALENT OF -11 IS : 10101

THE BINARY EQUIVALENT OF 7 IS : 00111

	OPERATION	A	Q	Q'	M
	INITIAL	00000	00111	0	10101
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+01011)	01011	00111	0	10101
	ARITHMETIC RIGHT SHIFT	00101	10011	1	10101
ROUND 2 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	00010	11001	1	10101
ROUND 3 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	00001	01100	1	10101
ROUND 4 Q_0=0 Q_{-1}=1	A:=A+M (00001+10101)	10110	01100	1	10101
	ARITHMETIC RIGHT SHIFT	11011	00110	0	10101
ROUND 5 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	11101	10011	0	10101

THE ANSWER IN BINARY IS : 1110110011
THE ANSWER IN DECIMAL IS : -77

13.Using Booth's algorithm show the multiplication of -12 X 13

THE BINARY EQUIVALENT OF -12 IS : 10100

THE BINARY EQUIVALENT OF 13 IS : 01101

	OPERATION	A	Q	Q'	M
	INITIAL	00000	01101	0	10100
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+01100)	01100	01101	0	10100
	ARITHMETIC RIGHT SHIFT	00110	00110	1	10100
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (00110+10100)	11010	00110	1	10100
	ARITHMETIC RIGHT SHIFT	11101	00011	0	10100
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (11101+01100)	01001	00011	0	10100
	ARITHMETIC RIGHT SHIFT	00100	10001	1	10100
ROUND 4 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	00010	01000	1	10100
ROUND 5 Q_0=0 Q_{-1}=1	A:=A+M (00010+10100)	10110	01000	1	10100
	ARITHMETIC RIGHT SHIFT	11011	00100	0	10100

THE ANSWER IN BINARY IS : 1101100100
THE ANSWER IN DECIMAL IS : -156

14.Using Booth's algorithm show the multiplication of -12 x 5

THE BINARY EQUIVALENT OF -12 IS : 10100
THE BINARY EQUIVALENT OF 5 IS : 00101

	OPERATION	A	Q	Q'	M
	INITIAL	00000	00101	0	10100
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+01100)	01100	00101	0	10100
	ARITHMETIC RIGHT SHIFT	00110	00010	1	10100
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (00110+10100)	11010	00010	1	10100
	ARITHMETIC RIGHT SHIFT	11101	00001	0	10100
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (11101+01100)	01001	00001	0	10100
	ARITHMETIC RIGHT SHIFT	00100	10000	1	10100
ROUND 4 Q_0=0 Q_{-1}=1	A:=A+M (00100+10100)	11000	10000	1	10100
	ARITHMETIC RIGHT SHIFT	11100	01000	0	10100
ROUND 5 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	11110	00100	0	10100

THE ANSWER IN BINARY IS : 1111000100
THE ANSWER IN DECIMAL IS : -60

15.**Using Booth's algorithm show the multiplication of 13 X -11**

THE BINARY EQUIVALENT OF 13 IS : 01101

THE BINARY EQUIVALENT OF -11 IS : 10101

	OPERATION	A	Q	Q'	M
	INITIAL	00000	10101	0	01101
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+10011)	10011	10101	0	01101
	ARITHMETIC RIGHT SHIFT	11001	11010	1	01101
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (11001+01101)	00110	11010	1	01101
	ARITHMETIC RIGHT SHIFT	00011	01101	0	01101
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (00011+10011)	10110	01101	0	01101
	ARITHMETIC RIGHT SHIFT	11011	00110	1	01101
ROUND 4 Q_0=0 Q_{-1}=1	A:=A+M (11011+01101)	01000	00110	1	01101
	ARITHMETIC RIGHT SHIFT	00100	00011	0	01101
ROUND 5 Q_0=1 Q_{-1}=0	Q_0=1 Q_{-1}=0 A:=A-M (00100+10011)	10111	00011	0	01101
	ARITHMETIC RIGHT SHIFT	11011	10001	1	01101

THE ANSWER IN BINARY IS : 1101110001
THE ANSWER IN DECIMAL IS : -143

16.**Using Booth's algorithm show the multiplication of 14 * -15**

THE BINARY EQUIVALENT OF 14 IS : 01110

THE BINARY EQUIVALENT OF -15 IS : 10001

	OPERATION	A	Q	Q'	M
	INITIAL	00000	10001	0	01110
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+10010)	10010	10001	0	01110
	ARITHMETIC RIGHT SHIFT	11001	01000	1	01110
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (11001+01110)	00111	01000	1	01110
	ARITHMETIC RIGHT SHIFT	00011	10100	0	01110
ROUND 3 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00001	11010	0	01110
ROUND 4 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	11101	0	01110
ROUND 5 Q_0=1 Q_{-1}=0	Q_0=1 Q_{-1}=0 A:=A-M (00000+10010)	10010	11101	0	01110
	ARITHMETIC RIGHT SHIFT	11001	01110	1	01110

THE ANSWER IN BINARY IS : 1100101110
THE ANSWER IN DECIMAL IS : -210

17. **Using Booth's algorithm show the multiplication of -8 x -8**

THE BINARY EQUIVALENT OF -8 IS : 11000

THE BINARY EQUIVALENT OF -8 IS : 11000

	OPERATION	A	Q	Q'	M
	INITIAL	00000	11000	0	11000
ROUND 1 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	01100	0	11000
ROUND 2 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	00110	0	11000
ROUND 3 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	00011	0	11000
ROUND 4 Q_0=1 Q_{-1}=0	A:=A-M (00000+01000)	01000	00011	0	11000
	ARITHMETIC RIGHT SHIFT	00100	00001	1	11000
ROUND 5 Q_0=1 Q_{-1}=1	ARITHMETIC RIGHT SHIFT	00010	00000	1	11000

THE ANSWER IN BINARY IS : 0001000000
THE ANSWER IN DECIMAL IS : 64

18.**Using Booth's algorithm show the multiplication of -15 X -15**

THE BINARY EQUIVALENT OF -15 IS : 10001

THE BINARY EQUIVALENT OF -15 IS : 10001

	OPERATION	A	Q	Q'	M
	INITIAL	00000	10001	0	10001
ROUND 1 Q_0=1 Q_{-1}=0	A:=A-M (00000+01111)	01111	10001	0	10001
	ARITHMETIC RIGHT SHIFT	00111	11000	1	10001
ROUND 2 Q_0=0 Q_{-1}=1	A:=A+M (00111+10001)	11000	11000	1	10001
	ARITHMETIC RIGHT SHIFT	11100	01100	0	10001
ROUND 3 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	11110	00110	0	10001
ROUND 4 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	11111	00011	0	10001
ROUND 5 Q_0=1 Q_{-1}=0	A:=A-M (11111+01111)	01110	00011	0	10001
	ARITHMETIC RIGHT SHIFT	00111	00001	1	10001

THE ANSWER IN BINARY IS : 0011100001
THE ANSWER IN DECIMAL IS : 225

19.**Using Booth's algorithm show the multiplication of -11 X -12**

THE BINARY EQUIVALENT OF -11 IS : 10101
THE BINARY EQUIVALENT OF -12 IS : 10100

	OPERATION	A	Q	Q'	M
	INITIAL	00000	10100	0	10101
ROUND 1 Q_0=1 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	01010	0	10101
ROUND 2 Q_0=0 Q_{-1}=0	ARITHMETIC RIGHT SHIFT	00000	00101	0	10101
ROUND 3 Q_0=1 Q_{-1}=0	A:=A-M (00000+01011)	01011	00101	0	10101
	ARITHMETIC RIGHT SHIFT	00101	10010	1	10101
ROUND 4 Q_0=0 Q_{-1}=1	A:=A+M (00101+10101)	11010	10010	1	10101
	ARITHMETIC RIGHT SHIFT	11101	01001	0	10101
ROUND 5 Q_0=1 Q_{-1}=0	A:=A-M (11101+01011)	01000	01001	0	10101
	ARITHMETIC RIGHT SHIFT	00100	00100	1	10101

THE ANSWER IN BINARY IS : 0010000100
THE ANSWER IN DECIMAL IS : 132

RESTORING DIVISION ALGORITHM

The algorithm can be summarized as follows:

1. Load the twos complement of the divisor into the M register; that is, the M register contains the negative of the divisor.

Load the dividend into the A, Q registers. The dividend must be expressed as a 2n-bit positive number.

Thus, for example, the 4-bit 0111 becomes 00000111.

2. Shift A, Q left 1 bit position.

3. Perform A←A - M. This operation subtracts the divisor from the contents of A.

4. a. If the result is nonnegative (most significant bit of A = 0), then set $Q_0 \leftarrow 1$.

b. If the result is negative (most significant bit of A = 1), then set $Q_0 \leftarrow 0$ and restore the previous value of A.

5. Repeat steps 2 through 4 as many times as there are bit positions in Q.

6. The remainder is in A and the quotient is in Q.

NOTE:if the given dividend or divisor is negative load the dividend into A,Q as unsigned integer and follow the above algorithm ,After that the value of A and Q is two complemented according to following rules

Specifically, sign(A) = sign(Dividend)

and sign(Q) = sign(Dividend) * sign(Divisor).

For example if -7/3 or 7/-3 or -7/-3

perform restoring alorithm taking Q as postive 7 ,M as postive 3 and get end result as usual A Q=0000 0111 just like 7/3

after getting result now change the values of A and Q as follows

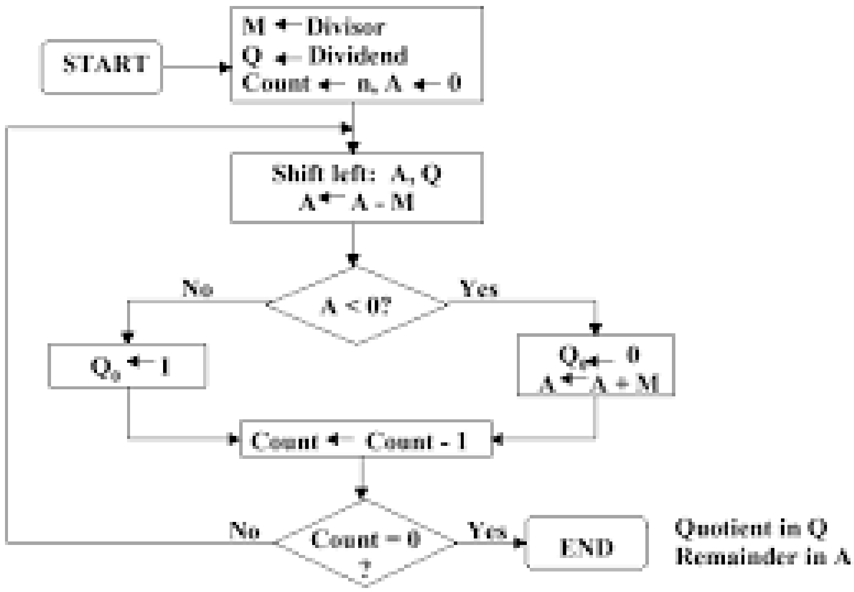
a)if dividend(7) alone is negative(-7) and divisor(3) is positive(3) then received A(remainder 0001) alone is two complemented to get 1111 ie -1 and Q (0010) is kept as it is.ie +2

b)if divisor(3) alone is negative(-3) and dividend(7) is positive(7) then received A(remainder 0001) alone is kept as it is and Q (0010) is two complemented to get 1110 ie -2

c)if both dividend(7) is negative(-7) and divisor(3) is negative(-3) then received A(remainder 0001) is two complemented to get 1111 ie -1 and Q (0010) is kept as it is.ie +2

Dividend	Divisor	For A-(remainder) sign is Sign(Dividend)	For Q(Quotient) sign is Sign(Dividend)*sign(Divisor)
+7	+3	+1 because Sign of Dividend 7 is Plus	+2 Sign(7)*sign(3) Plus*Plus=Plus
+7	-3	+1 because Sign of Dividend 7 is Plus	-2 Sign(7)*sign(-3) Plus*Minus=Minus
-7	+3	-1 because Sign of Dividend 7 is Minus	-2 Sign(-7)*sign(3) Minus*Plus=Minus
-7	-3	-1 because Sign of Dividend 7 is Minus	+2 Sign(-7)*sign(-3) Minus*Minus=Plus

1.Draw flowchart for restoring division algorithm.



POSITIVE DIVIDEND AND POSITIVE DIVISOR

1 A.Divide 11 by 2 using restoring division algorithm

HF	OPERATION	A						Q					M	M’’
		CY	A4	A3	A2	A1	A0	Q4	Q3	Q2	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	0	1	1	00010	11110
	Shift Left	0	0	0	0	0	0	1	0	1	1		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	0	1	1		00010	11110
ROUND 1	carry= 1 Q0=0	1	1	1	1	1	0	1	0	1	1	0	00010	11110
	A	1	1	1	1	1	0						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	0	1	0	1	1	0	00010	11110
ROUND 2	Shift Left	0	0	0	0	0	1	0	1	1	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-MA=A+M’’	1	1	1	1	1	1	0	1	1	0		00010	11110
	carry=1 Q0=0							0	1	1	0	0	00010	11110

	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	0	1	1	0	0	00010	11110
ROUND 3	Shift Left	0	0	0	0	1	0	1	1	0	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	1	0	0		00010	11110
	carry= 0 Q0=1	0	0	0	0	0	0	1	1	0	0	1	00010	11110
ROUND 4	Shift Left	0	0	0	0	0	1	1	0	0	1		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=1 Q0=0	1	1	1	1	1	1	1	0	0	1	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	1	0	0	1	0	00010	11110
ROUND 5	Shift Left	0	0	0	0	1	1	0	0	1	0		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=0 Q0=1	0	0	0	0	0	1	0	0	1	0	1	00010	11110

Sign(A)=Sign(Dividend)

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Since Dividend 11 is POSITIVE sign of A is POSITIVE

so A has no change 00001

Since Dividend 11 is positive sign and divisor 2 is positive sign

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Sign (Q)=Plus*Plus=Plus

so Q has no change Q is +5

FINAL RESULT	0	0	0	0	0	1	0	0	1	0	1		

NEGATIVE DIVIDEND and POSITIVE DIVISOR

1.B Divide -11 by 2 using restoring division algorithm

	OPERATION		A					Q					M	M’’
		CY	A4	A3	A2	A1	A0	Q4	Q3	Q2	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	0	1	1	00010	11110
	Shift Left	0	0	0	0	0	0	1	0	1	1		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	0	1	1		00010	11110
ROUND 1	carry= 1 Q0=0	1	1	1	1	1	0	1	0	1	1	0	00010	11110
	A	1	1	1	1	1	0						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	0	1	0	1	1	0	00010	11110
	Shift Left	0	0	0	0	0	1	0	1	1	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-MA=A+M’’	1	1	1	1	1	1	0	1	1	0		00010	11110
ROUND 2	carry=1 Q0=0							0	1	1	0	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	0	1	1	0	0	00010	11110
ROUND 3	Shift Left	0	0	0	0	1	0	1	1	0	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	1	0	0		00010	11110

	carry= 0 Q0=1	0	0	0	0	0	0	1	1	0	0	1	00010	11110
ROUND 4	Shift Left	0	0	0	0	0	1	1	0	0	1		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=1 Q0=0	1	1	1	1	1	1	1	0	0	1	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	1	0	0	1	0	00010	11110
ROUND 5	Shift Left	0	0	0	0	1	1	0	0	1	0		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=0 Q0=1	0	0	0	0	0	1	0	0	1	0	1	00010	11110

Sign(A)=Sign(Dividend)

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Since Dividend 11 is negative sign of A is negative

so A is twos complemented 00001=11110+1=11111(-1)

Since Dividend 11 is negative sign and divisor 2 is positive

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Sign (Q)=Minus*Plus=Minus

so Q is twos complemented 00101=11010+1=11011(-5)

FINAL RESULT	1	1	1	1	1	1	1	1	0	1	1		

POSITIVE DIVIDEND and NEGATIVE DIVISOR

1.C Divide 11/ -2 using restoring division algorithm

	OPERATION		A					Q					M	M’’
		CY	A4	A3	A2	A1	A0	Q4	Q3	Q2	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	0	1	1	00010	11110
ROUND 1	Shift Left	0	0	0	0	0	0	1	0	1	1		00010	11110

	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	0	1	1		00010	11110
	carry= 1 Q0=0	1	1	1	1	1	0	1	0	1	1	0	00010	11110
	A	1	1	1	1	1	0						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	0	1	0	1	1	0	00010	11110
	Shift Left	0	0	0	0	0	1	0	1	1	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-MA=A+M’’	1	1	1	1	1	1	0	1	1	0		00010	11110
ROUND 2	carry=1 Q0=0							0	1	1	0	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	0	1	1	0	0	00010	11110
	Shift Left	0	0	0	0	1	0	1	1	0	0		00010	11110
	M’’	1	1	1	1	1	0							
ROUND 3	A=A-M A=A+M’’	1	1	1	1	1	0	1	1	0	0		00010	11110
	carry= 0 Q0=1	0	0	0	0	0	0	1	1	0	0	1	00010	11110
ROUND 4	Shift Left	0	0	0	0	0	1	1	0	0	1		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=1 Q0=0	1	1	1	1	1	1	1	0	0	1	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	1	0	0	1	0	00010	11110

ROUND 5	Shift Left	0	0	0	0	1	1	0	0	1	0		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=0 Q0=1	0	0	0	0	0	1	0	0	1	0	1	00010	11110

Sign(A)=Sign(Dividend)

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Since Dividend 11 is POSITIVE sign of A is Positive

so A has no change 00001

Since Dividend 11 is Positive sign and divisor 2 is positive

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Sign (Q)=Plus*Minus=Minus

so Q is two complemented 00101=11010+1=11011(-5)

FINAL RESULT		0	0	0	0	0	1	1	1	0	1	1		

NEGATIVE DIVIDEND AND NEGATIVE DIVISOR

1.D Divide -11/ -2 using restoring division algorithm

	OPERATION		A					Q					M	M’’
		CY	A4	A3	A2	A1	A0	Q4	Q3	Q2	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	0	1	1	00010	11110
ROUND 1	Shift Left	0	0	0	0	0	0	1	0	1	1		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	0	1	1		00010	11110
	carry= 1 Q0=0	1	1	1	1	1	0	1	0	1	1	0	00010	11110
	A	1	1	1	1	1	0						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	0	1	0	1	1	0	00010	11110
ROUND 2	Shift Left	0	0	0	0	0	1	0	1	1	0		00010	11110

	M’’	1	1	1	1	1	0							
	A=A-MA=A+M’’	1	1 >	1	1	1	1	0	1	1	0		00010	11110
	carry=1 Q0=0							0	1	1	0	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	0	1	1	0	0	00010	11110
ROUND 3	Shift Left	0	0	0	0	1	0	1	1	0	0		00010	11110
	M’’	1	1	1	1	1	0							
	A=A-M A=A+M’’	1	1	1	1	1	0	1	1	0	0		00010	11110
	carry= 0 Q0=1	0	0	0	0	0	0	1	1	0	0	1	00010	11110
ROUND 4	Shift Left	0	0	0	0	0	1	1	0	0	1		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=1 Q0=0	1	1	1	1	1	1	1	0	0	1	0	00010	11110
	A	1	1	1	1	1	1						00010	11110
	A=A+M	0	0	0	0	1	0						00010	11110
	RESTORE	0	0	0	0	0	1	1	0	0	1	0	00010	11110
ROUND 5	Shift Left	0	0	0	0	1	1	0	0	1	0		00010	11110
	A=A-M	1	1	1	1	1	0						00010	11110
	carry=0 Q0=1	0	0	0	0	0	1	0	0	1	0	1	00010	11110

Sign(A)=Sign(Dividend)

Sign (Q)=Sign(Dividend) * Sign(Quotient)

Since Dividend 11 is negative sign of A is negative

so A is two complemented 00001=11110+1=11111(-1)

Since Dividend 11 is negative sign and divisor 2 is negative sign

Sign (Q)=Sign(Dividend) * Sign(Quotient)

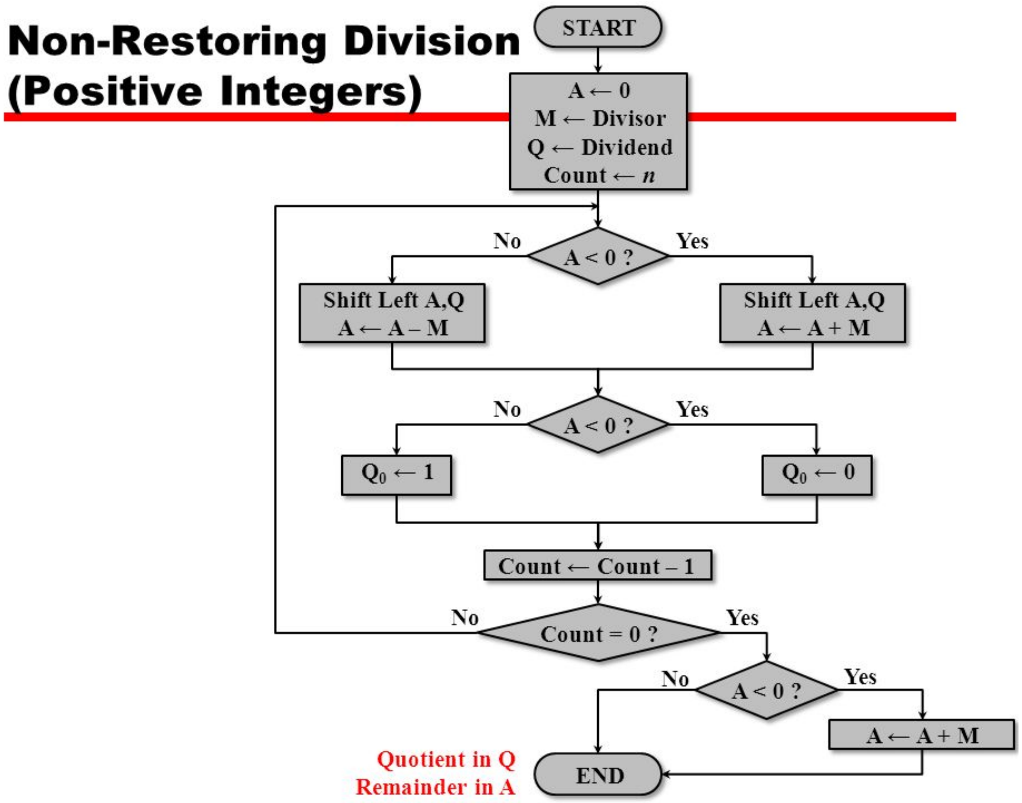
Sign (Q)=Minus*Minus=Plus

so Q has no change Q is +5

FINAL RESULT	1	1	1	1	1	1	0	0	1	0	1		

Non restoring division algorithm

Draw flowchart for non restoring division algorithm.



1.Divide 13 by 2 using non restoring division algorithm.

	OPERATION	A						Q					M	M''
		CY	A4	A3	A_2/span>	A1	A0	Q4	Q3	Q_2/span>	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	1	0	1	00010	11110
	Shift Left	0	0	0	0	0	0	1	1	0	1		00010	11110
ROUND 1	carry= 0 A=A-M A=A+M'	1	1	1	1	1	0							
	carry= 1 Q0=0	1	1	1	1	1	0	1	1	0	1	0	00010	11110
ROUND 2	Shift Left	1	1	1	1	0	1	1	0	1	0		00010	11110
	carry= 1 A=A+M	0	0	0	0	1	0							

	carry=1 Q0=0	1	1	1	1	1	1	1	0	1	0	0	00010	11110
	Shift Left	1	1	1	1	1	1	0	1	0	0		00010	11110
ROUND 3	carry= 1 A=A+M	0	0	0	0	1	0							
	carry= 0 Q0=1	0	0	0	0	0	1	0	1	0	0	1	00010	11110
	Shift Left	0	0	0	0	1	0	1	0	0	1		00010	11110
ROUND 4	carry= 0 A=A-M A=A+M’	1	1	1	1	1	0						00010	11110
	carry=0 Q0=1	0	0	0	0	0	0	1	0	0	1	1	00010	11110
	Shift Left	0	0	0	0	0	1	0	0	1	1		00010	11110
ROUND 5	carry= 0 A=A-M A=A+M’	1	1	1	1	1	0						00010	11110
	carry=1 Q0=0	1	1	1	1	1	1	0	0	1	1	0	00010	11110
Negative remainder	carry= 1 A=A+M	0	0	0	0	1	0							
Final answer		0	0	0	0	0	1	0	0	1	1	0		

Remainder=A=00001=1 quotient=Q=00110=6

2.Divide 11 by 4 using non restoring division algorithm

	OPERATION		A					Q					M	M’’
		CY	A4	A3	A_2/span>	A1	A0	Q4	Q3	Q_2/span>	Q1	Q0		
	INITIAL	0	0	0	0	0	0	0	1	0	1	1	00100	11100
ROUND 1	Shift Left	0	0	0	0	0	0	1	0	1	1			
	carry= 0 A=A-M A=A+M’	1	1	1	1	0	0							

	carry= 1 Q0=0	1	1	1	1	0	0	1	0	1	1	0		
ROUND 2	Shift Left	1	1	1	0	0	1	0	1	1	0		00100	11100
	carry= 1 A=A+M	0	0	0	1	0	0							
	carry=1 Q0=0	1	1	1	1	0	1	0	1	1	0	0		
ROUND 3	Shift Left	1	1	1	0	1	0	1	1	0	0		00100	11100
	carry= 1 A=A+M	0	0	0	1	0	0							
	carry= 1 Q0=0	1	1	1	1	1	0	1	1	0	0	0		
ROUND 4	Shift Left	1	1	1	1	0	1	1	0	0	0		00100	11100
	carry= 1 A=A+M	0	0	0	1	0	0							
	carry=0Q0=1	0	0	0	0	0	1	1	0	0	0	1		
ROUND 5	Shift Left	0	0	0	0	1	1	0	0	0	1		00100	11100
	carry= 0 A=A-M A=A+M’	1	1	1	1	0	0							
	carry=1 Q0=0	1	1	1	1	1	1	0	0	0	1	0		
Negative remainder	carry= 1 A=A+M	0	0	0	1	0	0							
Final answer		0	0	0	0	1	1	0	0	0	1	0		

IEEE 754 standards for Floating Point number representation

1.Explain IEEE 754 standards for Floating Point number representation.

Power table

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
1	2	4	8	16	32	64	128	256	512	1024	.5	.25	.125	.0625	.03125	.015625

The following is a step by step roadmap to go from a decimal number to its IEEE 754 32 bit floating point representation. The example will use -176.375 as an example

STEP 1: OBSERVE THE SIGN

For -176.375 the sign is negative. This means the first bit will be 1, if positive the first bit will be 0. Going forward the sign will be ignored, but then used again in the last step.

STEP 2: FROM DECIMAL TO BINARY

Transform the decimal to binary (ignoring the sign). To do this subtract the largest power of 2 relative to the decimal until you reach 0. Note that floating point is an approximation and cannot be perfectly represented – but the potential rounding error is very small.

Calculations

Use2^7	Use 2^5	;Use 2^4	Use 2^{-2}	Use 2^{-3}
176.375	48.375	16.375	0.375	0.125
-128.000	-32.000	- 16.000	- 0.250	-0.125
48.375	16.375	0.375	0.125	0.000

As a sum : $2^7 + 2^5 + 2^4 + 2^{-2} + 2^{-3} = 176.375$

As binary : 10110000.011 (Place a 1 in each position used)

STEP 3: MOVE TO SCIENTFIC NOTATION AND GET SIGNIFICANT

IEEE Floating points need to be in the format of 1.xxuxxx * 2y . The significant is the xxxxx component (ignore the 1.) and has 23 bits. If your significant is shorter then 23 bits add trailing zeros.

$10110000.011 = 1.0110000011 * 2^7$

Significant = 01100000110000000000000 (had to add 13 trailing zeros)

STEP 4: CALCULATE EXPONENT IN BINARY

The exponent is represented by 8 bits (256 states) and is shifted by 127. In our example ($1.0110000011 * 2^7$) the exponent is 7. So we need to express 134 (from 7+127) in binary. Using the same technique as step 2:

As a sum : 128 + 4 + 2 = 134

As a sum : $2^7 + 2^2 + 2^1 = 134$

As binary : 10000110 (Place a 1 in each position used)

STEP 5: COMBINE SIGN, EXPONENT, AND SIGNIFICANT

The format is:

Sign	Exponent	Significant
(1bit)	(8bits)	(23bits)
1	10000110	01100000110000000000000

Or: 11000011001100000110000000000000 (Hex : 0xc3306000)

Problems

2.Represent $(12.25)_{10}$ in single and double precision IEEE 754 standards for Floating Point number representation.

Use 2^3	Use 2^2	Use 2^{-2}
12.25	4.25	0.25
- 8.00	- 4.00	-0.25
4.25	0.25	.0

As a sum : $2^3 + 2^2 + 2^{-2} = 12.25$

As binary : 1100.01(Place a 1 in each position used)

STEP 1: MOVE TO SCIENTFIC NOTATION AND GET SIGNIFICANT

IEEE Floating points need to be in the format of 1.xxxxx * 2^y . The significant is the xxxxx component (ignore the 1.) and has 23 bits. If your significant is shorter then 23 bits add trailing zeros.

$1100.01=1.10001 * 2^3$

Significant = 10001000000000000000000 (had to add 18 trailing zeros)

STEP 2: CALCULATE EXPONENT IN BINARY

I)SINGLE PRECISION

The exponent is represented by 8 bits (256 states) and is shifted by 127.

($1.10001 * 2^3$) the exponent is 3. So we need to express 130 (from 3+127) in binary. Using the same technique as step 2:

As a sum : 128 + 2 = 130

As a sum : $2^7 + 2^1 = 130$

As binary : 10000010 (Place a 1 in each position used)

II)DOUBLE PRECISION

The exponent is represented by 11 bits (2046 states) and is shifted by 1023.

($1.10001 * 2^3$) the exponent is 3. So we need to express 1026 (from 3+1023) in binary. Using the same technique as step 2:

As a sum : 1024 + 2 = 1026

As a sum : $2^{10} + 2^1 = 1026$

As binary : 10000000010 (Place a 1 in each position used)

STEP 5: COMBINE SIGN, EXPONENT, AND SIGNIFICANT

The format is:

I)SINGLE PRECISION

Sign	Exponent	Significant
(1bit)	(8bits)	(23bits)
0	10000010	10001000000000000000000

II)DOUBLE PRECISION

Sign	Exponent	Significant
(1bit)	(11 bits)	(52bits)
0	10000000010	10001000000000000000000...

3.Represent $(127.125)_{10}$ in single and double precision IEEE 754 standards for Floating Point number representation.

Ans:

127 As binary 01111111

Use 2^6	Use 2^5	Use 2^4	Use 2^3	Use 2^2	Use 2^1	Use 2^0	Use 2^{-3}
127.125	63.125	31.125	15.125	7.125	3.125	1.125	0.125
-64.000	-32.000	-16.000	-08.000	-4.000	-2.000	- 1.000	0.125
63.125	31.125	15.125	7.125	3.125	1.125	0.125	0.000

As a sum : $2^6 + 2^5 + 2^4 + 2^3 + 2^{2+}2^1 + 2^0 + 2^{-3} = 127.125$

As binary : 1111111.001 (Place a 1 in each position used)

STEP 3: MOVE TO SCIENTFIC NOTATION AND GET SIGNIFICANT

IEEE Floating points need to be in the format of 1.xxxxx * 2y . The significant is the xxxxx component (ignore the 1.) and has 23 bits. If your significant is shorter then 23 bits add trailing zeros.

$1111111.001=1.111111001*2^6$

Significant = 11111100100000000000 000 (had to add 14 trailing zeros)

STEP 4: CALCULATE EXPONENT IN BINARY

I)SINGLE PRECISION

The exponent is represented by 8 bits (256 states) and is shifted by 127. In our example ($1.111111001*2^6$) the exponent is 6. So we need to express 133 (from 6+127) in binary. Using the same technique as step 2:

As a sum : 128 + 4 + 1 = 133

As a sum : $2^7 + 2^2 + 2^0 = 133$

As binary : 10000101 (Place a 1 in each position used)

II)DOUBLE PRECISION

The exponent is represented by 11 bits (256 states) and is shifted by 1024. In our example ($1.111111001*2^6$) the exponent is 6. So we need to express 1029 (from 6+1023) in binary. Using the same technique as step 2:

As a sum : 1024 + 4 + 1 = 1029

As a sum : $2^{10} + 2^2 + 2^0 = 1029$

As binary : 10000000101 (Place a 1 in each position used)

STEP 5: COMBINE SIGN, EXPONENT, AND SIGNIFICANT

The format is:

I)SINGLE PRECISION

Sign	Exponent	Significant
(1bit)	(8bits)	(23bits)
0	10000101	11111100100000000000000

II)DOUBLE PRECISION

Sign	Exponent	Significant
(1bit)	(11bits)	(23bits)
0	10000000101	11111100100000000000000...

4.Represent $(28.75)_{10}$ in single and double precision IEEE 754 standards for Floating Point number representation.

Binary for 28 is 00011100

Binary for .75 is .11

$(.75-.5(2^{-1})=.25$

$.25-.25(2^{-2})=0$

$(2^{-1})\quad(2^{-2})=.11$

28.75=00011100.11

$1.110011*2^4$

Significant = 11001100000000000000 000

I)SINGLE PRECISION

Exponent Baising e=4

Exponent=e+127

Exponent=4+127=131

Binary for 131 is 10000011

II)DOUBLE PRECISION

Exponent Baising e=4

Exponent=e+1023

Exponent=4+1023=1027

Binary for 131 is 10000000011

The format is:

I)SINGLE PRECISION

Sign	Exponent	Significant
(1bit)	(8bits)	(23bits)
0	10000011	11001100000000000000000

II)DOUBLE PRECISION

Sign	Exponent	Significant
(1 bit)	(11 bits)	(52 bits)
0	10000000011	11001100000000000000000

5.Represent $(-32.75)_{10}$ in single and double precision IEEE 754 standards for Floating Point number representation.

Binary for 32 is 00100000

Binary for .75 is .11

$(.75-.5(2^{-1})=.25$

$.25-.25(2^{-2})=0$

(2^{-1}) (2^{-2})=.11

-32.75=00100000.11

$1.0000011*2^5$

Significant = 0000 0110 0000 0000 0000 000

I)SINGLE PRECISION

Exponent Baising e=5

Exponent=e+127

Exponent=5+127=132

Binary for 132 is 10000100

II)DOUBLE PRECISION

Exponent Baising e=4

Exponent=e+1023

Exponent=5+1023=1028

Binary for 1027 is 10000000100

The format is:

I)SINGLE PRECISION

Sign	Exponent	Significant
(1bit)	(8bits)	(23bits)
1	10000100	0000 0110 0000 0000 0000 000

II)DOUBLE PRECISION

Sign	Exponent	Significant
(1 bit)	(11 bits)	(52 bits)
1	10000000100	0000 0110 0000 0000 0000 000...

Question Bank

1	Draw flowchart of Booth's algorithm.	5
2	Draw flowchart of Booth's algorithm.and multiply(4)*(-3) using Booth's Algorithm	10
3	Multiply $(— 2)_{10}$ and $(— 5)_{10}$ using Booth's Algorithm	5
4	Multiply (4) and (4) using Booth's Algorithm.	5
5	Using Booth's Algorithm show the multiplication of 7x5.	5
6	Using Booth's algorithm show the multiplication of -9 *4	5
7	Draw flow chart of restoring division algorithm.	5
8	Divide 15 by 4 using restoring division algorithm.	5
9	Divide 17 by 7using restoring division algorithm.	5
10	Divide 11 by 2 using restoring division algorithm.	5
11	Draw flow chart of non restoring division algorithm.	5
12	Using non restoring Division method, divide 7 by 3.	5
13	Using non restoring Division method, divide 11 by 7.	5
14	Explain IEEE 754 standards for Floating Point number representation.	5
15	Represent (12.25)10 in double precision IEEE 754 standards for Floating Point number representation.	5
16	Represent (127.125)10 in single and double precision IEEE 754 standards for Floating Point number representation.	5
17	Represent (28.75)10 in single and double precision IEEE 754 standards for Floating Point number representation.	5
18	Show IEEE 754 standards for Binary Floating Point Representation for 32 bit single format and 64 bit double format.	5