Module 2

Data Representation and Arithmetic Algorithm

Motivation:

Computer understands only binaries i.e. 0 or 1. The ALU computations also will be carried out internally in the form of binary numbers by making use of algorithms. Hence there is a need to understand the different number representations and the arithmetic algorithms. This gives the understanding of how arithmetic calculations are done by the computer.

Syllabus:

Lecture no	Content	Duration (Hr)	Self-Study (Hrs)
1	Signed number representation	1	1
2	Fixed point computation algorithms	1	1
3	Booth multiplication	1	2
4	Non-restoring Division	1	2
5	Restoring Division	1	2
6	Floating point arithmetic algorithms	1	2
7	IEEE 754 floating point number representation	1	2

Learning Objectives:

Learners shall be able to:

1. To explain the different number representations
2. To describe the arithmetic algorithms for fixed point numbers
3. To compute the multiplication of two numbers using Booth multiplication
4. To compute the division of two numbers using no-restoring and restoring division algorithms
5. To describe the Floating point arithmetic algorithms
6. To explain about IEEE 754 floating point number representation

Theoretical Background:

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations. To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed

generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation. If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we give algorithms. In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations. And these instructions perform a great activity in processing data in a digital computer. With the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods. A processor has an arithmetic processor (as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement. Most computers use the signed magnitude representation for the mantissa.

Key Definitions:

Half adder: This is a logic circuit which performs the addition of two binary digits and produces a sum and a carry output.

Full adder: This is a logic circuit which performs addition of three bits (two significant bits and a previous carry) and produces a sum and a carry output.

Ripple carry adder: This can be constructed using a number of full adder circuits connected in parallel. The carry output of each adder is connected to the carry output of the next higher-order adder. Either a half-adder can be used for the least significant position or the carry input of a full adder is made 0 because there is no carry into the least significant position

Mantissa underflow: In the process of aligning mantissa, digits may flow off the right end of the mantissa. In such case truncation methods are used.

Mantissa overflow: The addition of two mantissas of the same sign may result in a carryout of the most significant bit. If so, the mantissa is shifted right and the exponent is incremented.

Exponent underflow: This occurs when a negative exponent exceeds the maximum possible exponent value. In such cases, the number is designated as zero.

Exponent overflow: This occurs when a positive exponent exceeds the maximum possible exponent value. In some systems this may be designated as $+\infty$ or $-\infty$.

Guard bits: These are extra bits provided in the intermediate steps of floating-point arithmetic calculations in order to get maximum accuracy in the final result.

Arithmetic Logic Unit (ALU): It is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer.

Accumulator: It is a special storage register associated with the arithmetic logic unit for storing the results of steps in a calculation or data transfer.

Arithmetic Operator: It is the arithmetical signs of addition, subtraction, division and multiplication as used by a given programming language.

Central Processing Unit (CPU): It is the main part of the computer, its 'brain', consisting of the central memory, arithmetic logic unit and control unit.

Control Unit (CU): It is that part of the computer which accesses instructions in sequence, interprets them and then directs their implementation.

Course Content:

Lecture 1

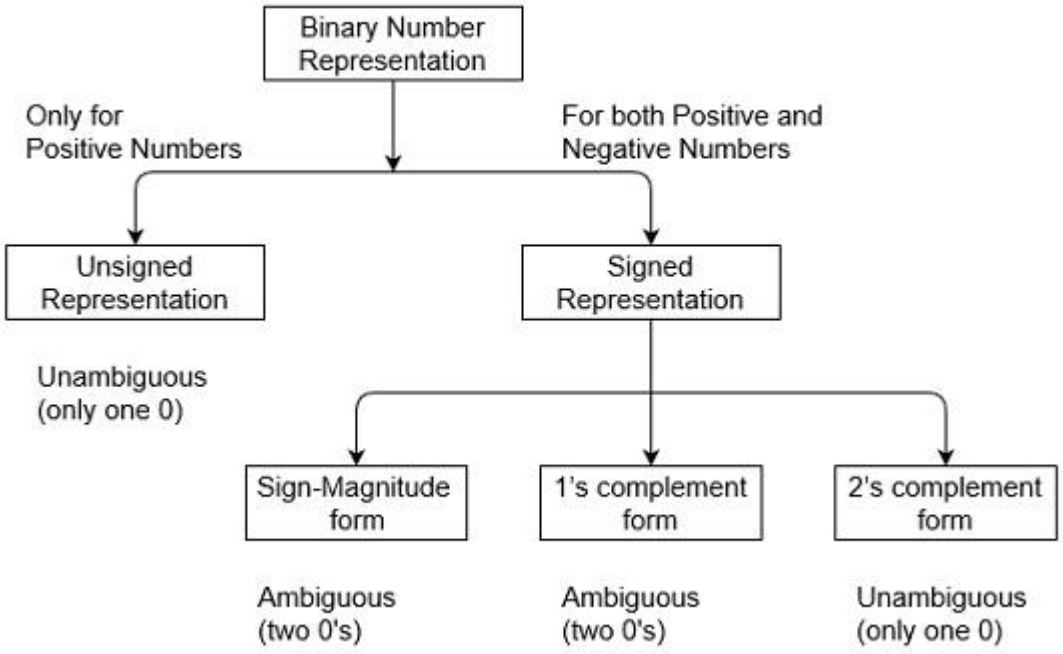
Signed number representation:

Variables such as integers can be represented in two ways, i.e., signed and unsigned. Signed numbers use sign flag to distinguish between negative values and positive values. Number representation techniques like: Binary, Octal, Decimal and Hexadecimal number representation techniques can represent numbers in both signed and unsigned ways. Binary Number System is one the type of Number Representation techniques. It is most popular and used in digital systems. Binary system is used for representing binary quantities which can be represented by any device that has only two operating states or possible conditions. For example, a switch has only two states: open or close.

In the Binary System, there are only two symbols or possible digit values, i.e., 0 and 1. Represented by any device that only 2 operating states or possible conditions. Binary numbers are indicated by the addition of either an *0b* prefix or an *2* suffix.

Representation of Binary Numbers:

Binary numbers can be represented in signed and unsigned way. Unsigned binary numbers do not have sign bit, whereas signed binary numbers uses signed bit as well or these can be distinguishable between positive and negative numbers. A signed binary is a specific data type of a signed variable.



1. Unsigned Numbers:

Unsigned numbers don't have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers is all positive numbers only. For example, representation of positive decimal numbers is positive by default. We always assume that there is a positive sign symbol in front of every number.

Representation of Unsigned Binary Numbers:

Since there is no sign bit in this unsigned binary number, so N bit binary number represent its magnitude only. Zero (0) is also unsigned number. This representation has only one zero (0), which is always positive. Every number in unsigned number representation has only one unique binary equivalent form, so this is unambiguous representation technique. The range of unsigned binary number is from 0 to (2^n-1).

Example-1: Represent decimal number 92 in unsigned binary number.

Simply convert it into Binary number; it contains only magnitude of the given number.

$= (92)_{10}$

$= (1x2^6+0x2^5+1x2^4+1x2^3+1x2^2+0x2^1+0x2^0)_{10}$

$= (1011100)_2$

It's 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Since, range of unsigned binary number is from 0 to (2^n-1). Therefore, range of 5 bit unsigned binary number is from 0 to (2^5-1) which is equal from minimum value 0 (i.e., 00000) to maximum value 31 (i.e., 11111).

2. Signed Numbers:

Signed numbers contain sign flag, this representation distinguish positive and negative numbers. This technique contains both sign bit and magnitude of a number. For example, in representation of negative decimal numbers, we need to put negative symbol in front of given decimal number.

Representation of Signed Binary Numbers:

There are three types of representations for signed binary numbers. Because of extra signed bit, binary number zero has two representation, either positive (0) or negative (1), so ambiguous representation. But 2's complementation representation is unambiguous representation because of there is no double representation of number 0. These are: Sign-Magnitude form, 1's complement form, and 2's complement form which are explained as following below.

2.(a) Sign-Magnitude form:

For n bit binary number, 1 bit is reserved for sign symbol. If the value of sign bit is 0, then the given number will be positive, else if the value of sign bit is 1, then the given number will be negative. Remaining (n-1) bits represent magnitude of the number. Since magnitude of number zero (0) is always 0, so there can be two representation of number zero (0), positive (+0) and negative (-0), which depends on value of sign bit. Hence these representations are ambiguous generally because of two representation of number zero (0). Generally sign bit is a most significant bit (MSB) of representation. The range of Sign-Magnitude form is from ($2^{(n-1)}$-1) to ($2^{(n-1)}$-1).

For example, range of 6 bit Sign-Magnitude form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 11111) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 00000) and +0 (i.e., 0 00000).

2.(b) 1's complement form:

Since, 1's complement of a number is obtained by inverting each bit of given number. So, we represent positive numbers in binary form and negative numbers in 1's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 1's complement of given binary number. You can get negative number by 1's complement of a positive number and positive number by using 1's complement of a negative number. Therefore, in this representation, zero (0) can have two representations, that's why 1's complement form is also ambiguous form. The range of 1's complement form is *from (*$2^{(n-1)}$-1) to ($2^{(n-1)}$-1).

For example, range of 6 bit 1's complement form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

2.(c) 2's complement form:

Since, 2's complement of a number is obtained by inverting each bit of given number plus 1 to least significant bit (LSB). So, we represent positive numbers in binary form and negative numbers in 2's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 2's complement of given binary number. You can get negative number by 2's complement of a positive number and positive number by directly using simple binary representation. If value of most significant bit (MSB) is 1, then take 2's complement from, else not. Therefore, in this representation, zero (0) has only one (unique) representation which is always positive. The range of 2's complement form is *from (*$2^{(n-1)}$) to ($2^{(n-1)}$-1).

For example, range of 6 bit 2's complement form binary number is from (2^5) to (2^5-1) which is equal from minimum value -32 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

Let's check the take away from this lecture

1) _____ is a straightforward method of representing positive and negative numbers.

a) Radix

b) Complement

c) Sign Magnitude

d) Encode

2) The 1's complement of 1 in 4 bits is _____

a) 0001

b) 0

c) 1001

d) 1110

3) The binary number 111 in its 2's complement form is _____

a) 010

b) 001

c) 000

d) 111

Exercise

Q.1 List the different ways of representing signed numbers.

Q.2 State the drawback of signed magnitude representation.

Q.3 Find the 2's complement of 0101.

Learning from this lecture: Learners will be able to understand different ways of representing unsigned and signed numbers.

Lecture 2

Fixed point computation algorithms:

A fixed-point number representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating-point number representation. Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) as is the case for older or low-cost embedded microprocessors and microcontrollers.

Addition, Subtraction, Multiplication and Division of binary numbers:

It is possible to add and subtract binary numbers in a similar way to base 10 numbers.

For example, 1 + 1 + 1 = 3 in base 10 becomes 1 + 1 + 1 = 11 in binary.

In the same way, 3 – 1 = 2 in base 10 becomes 11 – 1 = 10 in binary.

When you add and subtract binary numbers you will need to be careful when 'carrying' or 'borrowing' as these will take place more often.

Key *Addition* Results for Binary Numbers

1 + 0 = 1

1 + 1 = 10

1 + 1 + 1 = 11

Key *Subtraction* Results for Binary Numbers

1 – 0 = 1

10 – 1 = 1

11 – 1 = 10

The operation performed while finding the binary product is similar to the conventional multiplication method. The four major steps in binary digit multiplication are:

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

Example 1: Solve 1010 × 101

Solution:

1010 × 101

1010

(×) 101

_____

1010

0000

_____

01010 ……. First Intermediate Sum

1010

_____

110010

Comparison with Decimal values:

$1010_2 = 10_{10}$

$1010_2 = 5_{10}$

$10 \times 5 = 50_{10}$

$(110010)_2 = 50_{10}$

Another example of binary multiplication with a decimal point is as follows:

Example 2: 1011.01 × 110.1

Solution:

```
      1 0 1 1.0 1
        1 1 0.1
    ---------------
      1 0 1 1 0 1
    0 0 0 0 0 0
    ---------------
    0 1 0 1 1 0 1  ..... First Intermediate sum
  1 0 1 1 0 1
  ---------------
  1 1 1 0 0 0 0 1  ..... Second Intermediate Sum
1 0 1 1 0 1
-------------
1 0 0 1 0 0 1.0 0 1  ..... Final Sum
```

There are different ways to solve division problems using binary operations. Long division is one of them and the easiest and the most efficient way.

The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:

- 1÷1 = 1

- 1÷0 = 0
- 0÷1 = Meaningless
- 0÷0 = Meaningless

Question: Solve 01111100 ÷ 0010

Solution:

Given

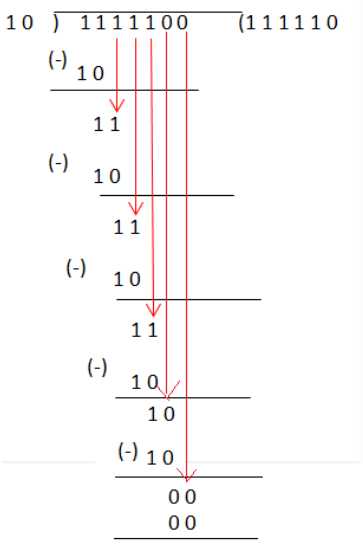
01111100 ÷ 0010

Here the dividend is 01111100, and the divisor is 0010

Remove the zero's in the **Most Significant Bit** in both the dividend and divisor, that doesn't change the value of the number.

So the dividend becomes 1111100, and the divisor becomes 10.

Now, use the long division method.



Unsigned Multiplication (Shift and Add method):

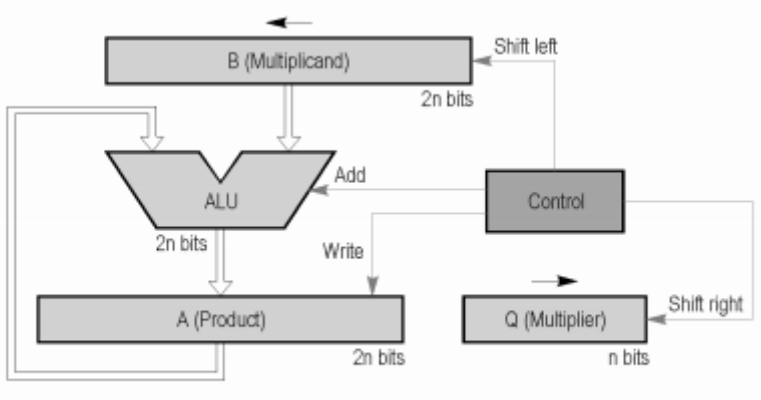
Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.

As an example, consider the multiplication of two unsigned 4-bit numbers, 8 (1000) and 9 (1001).

```
Multiplicand          1000 ×
Multiplier            1001
                      1000
                     0000
                    0000
                   1000
Product            1001000
```

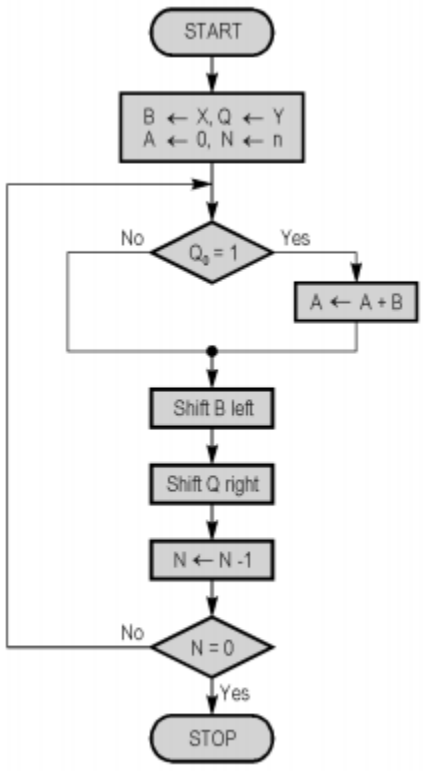
In the case of binary multiplication, since the digits are 0 and 1, each step of the multiplication is simple. If the multiplier digit is 1, a copy of the multiplicand (1 × multiplicand) is placed in the proper positions; if the multiplier digit is 0, a number of 0 digits (0 × multiplicand) are placed in the proper positions.

Consider the multiplication of positive numbers. The first version of the multiplier circuit, which implements the shift-and-add multiplication method for two n-bit numbers, is shown in Figure



First version of the multiplier circuit.

The 2n-bit product register (A) is initialized to 0. Since the basic algorithm shifts the multiplicand register (B) left one position each step to align the multiplicand with the sum being accumulated in the product register, we use a 2n-bit multiplicand register with the multiplicand placed in the right half of the register and with 0 in the left half.



The first version of the multiplication algorithm.

Figure shows the basic steps needed for the multiplication. The algorithm starts by loading the multiplicand into the B register, loading the multiplier into the Q register, and initializing the A register to 0. The counter N is initialized to n. The least significant bit of the multiplier register (Q0) determines whether the multiplicand is added to the product register. The left shift of the multiplicand has the effect of shifting the intermediate products to the left, just as when multiplying by paper and pencil. The right shift of the multiplier prepares the next bit of the multiplier to examine in the following iteration.

Example Perform the multiplication 9 × 12 (1001 × 1100) using the final version of the multiplication algorithm. **Answer** Table shows the revised multiplication example for the final version of the algorithm.

Step	**A**	**Q**	**B**	**Operation**
0	0000	1100	1001	Initialization
1	0000	0011	1001	Shift right A_Q
2	0000	0011	1001	Shift right A_Q
3	1001	0011	1001	Add B to A
0100	1001	1001	Shift right A_Q	
4	1101	1001	1001	Add B to A
0110	1100	1001	Shift right A_Q	

Let's check the take away from this lecture

1) If the radix point (binary point) is fixed and assumed to be to on the right of the right most digit, the representation of such number is called

a) Fixed point

b) Floating point

c) Radix point

d) None of these

2) Perform binary addition of 1101 + 0010 is _____

a) 1110

b) 1111

c) 0111

d) 1,1101

3) The multiplication of 110 * 111 is performed. What is a general term used for 111?

a) Dividend

b) Quotient

c) Multiplicand

d) Multiplier

Exercise
Q.1 What is a fixed-point number?
Q.2 Perform the following: i) 1100 + 0001 ii) 1100 - 0101
Q.3 Draw the block diagram of the basic multiplier circuit and explain.

Learning from this lecture: Learners will be get knowledge about the basic binary arithmetic operations and the algorithm used for unsigned multiplication.

Lecture 3

Booth Multiplication

There are many methods to multiply 2's complement numbers. The easiest is to simply find the magnitude of the two multiplicands, multiply these together, and then use the original sign bits to determine the sign of the result. If the multiplicands had the same sign, the result must be positive, if they had different signs, the result is negative. Multiplication by zero is a special case (the result is always zero, with no sign bit).

Booth's Algorithm

- Booth's algorithm can be used to multiply two numbers represented using two's complement (2C)
- Booth's algorithm will work for both positive and negative numbers represented using 2C
- The algorithm uses a combination of addition, subtraction and Arithmetic Right Shifts –subtraction is performed by taking the two's complement of the subtrahend (the number to be subtracted) and adding it to the minuend

Hardware Implementation

Booth's algorithm uses the following 4 registers (A, Q, Q_{-1})

- A– each bit in A is initialized to 0
  - A is reused and eventually stores the most-significant bits of the result
  - A is the same size as M
- Q is initialized to store the multiplier
  - Q is reused and eventually stores the least-significant bits of the result
- Q_{-1} is initialized to 0
  - this is placed to the right of Q
  - this is a 1-bit register which remembers the least-significant bit of Q (Q_0)

- M stores the multiplicand

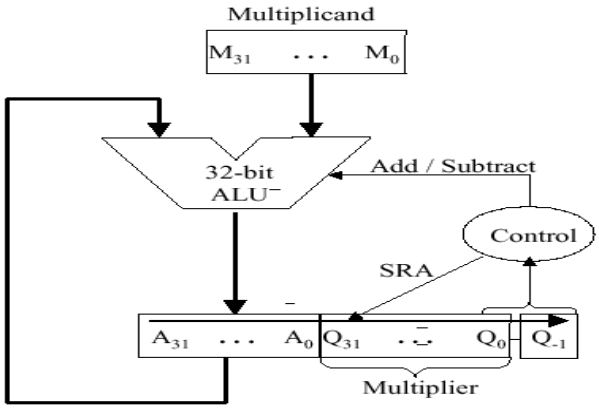


Fig 2.1: Hardware Implementation for signed numbers

- If Q_0 and Q_{-1} are the same (e.g. $Q_0 = 1$, $Q_{-1} = 1$ or $Q_0 = 0$, $Q_{-1} = 0$)

  –shift all of the bits of the A, Q, and Q_{-1}registers to the right (Arithmetic Shift Right, ASR)

- Else

  –if $Q_0$is 0 and $Q_{-1}$is 1, add the multiplicand to the contents of the A register

  –else if Q_0 is 1 and Q_{-1} is 0, subtract the multiplicand from the contents of the A register

  –shift all of the bits of the A, Q, and Q_{-1} registers to the right (ASR)

- Repeat for each bit of the multiplier
- The product is stored in the A and Q registers

The flowchart for Booth's algorithm is shown in fig 2.2.

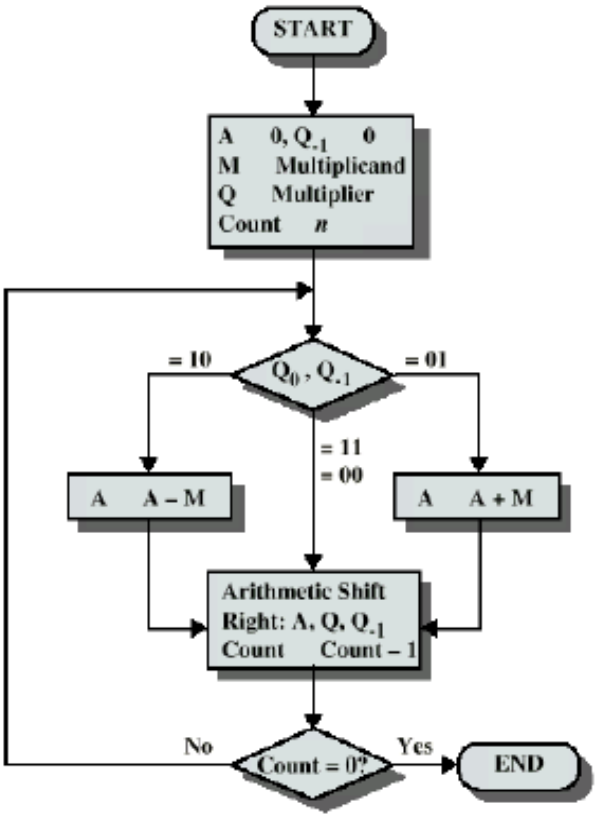


Fig 2.2: Flowchart for Booth's algorithm

Booth's algorithm example

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

Let's check the take away from this lecture

1) What is the default value of accumulator in booth's multiplication of two 4-bit binary numbers?

a) 0

b) 1

c) 0000

d) 00000

2) What will be the value obtained after multiplication of (-2) * (-3) using Booth's Algorithm?

a) 6

b) -6

c) -2

d) -3

3) One extra bit is added on the left of a binary number, in case of Binary Multiplication using Booth's Algorithm.

a) True

b) False

Exercise
Q.1 Why Booth's algorithm is used for binary multiplication?
Q.2 Draw the block diagram of Booth's multiplier circuit and explain.
Q.3 Solve -2*4 using Booth multiplication.

Learning from this lecture: Learners will be able to understand the steps of Booth multiplication algorithm in detail.

Lecture 4

Restoring Division

It is the *unsigned* division algorithm that is similar to Booth's algorithm.

Hardware Implementation:

The ALU schematic diagram in given in Figure 2.3. The analysis of the algorithm and circuit is very similar to the preceding discussion of Booth's algorithm.

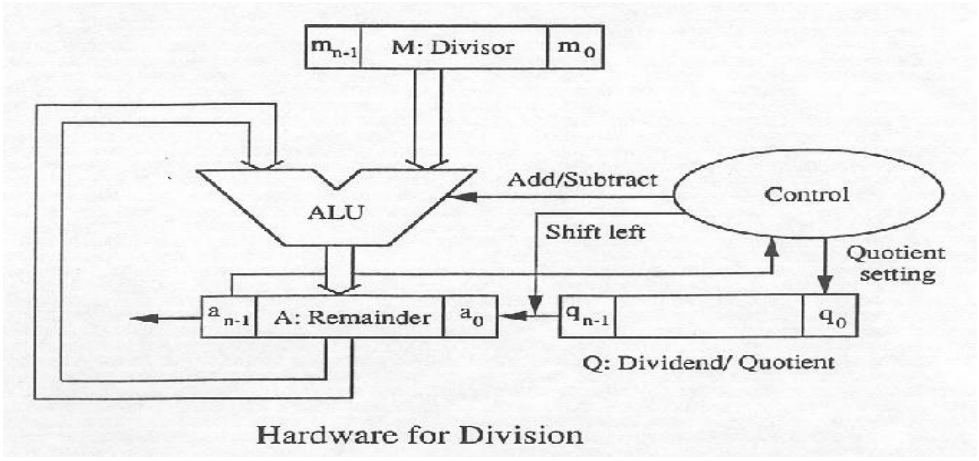


Fig 2.3: Hardware implementation for division operation

Algorithm for hardware division (restoring)

Do n times:

{

left shift A and Q by 1 bit

$A \leftarrow A - M$;

if $A < 0$ i.e. $a_{n-1} = 1$, then $q_0 \leftarrow 0$, $A \leftarrow A + M$ (restore)

else $q_0 \leftarrow 1$

}

Note: When A and Q are left shifted, the MSB of Q becomes the LSB of A, and the MSB of A is lost. The LSB of Q is made available for the next quotient bit.

Flowchart of restoring division operation:

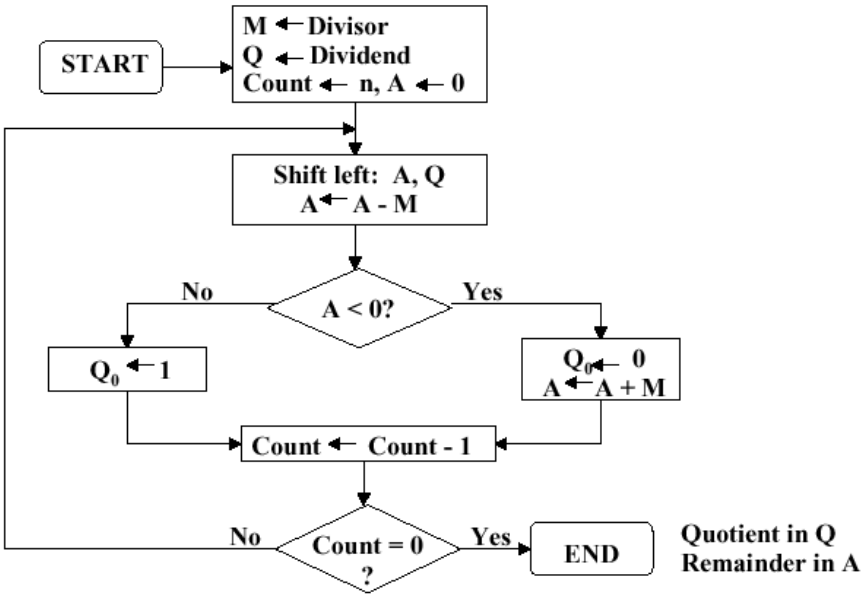


Fig 2.4: Flowchart for restoring division operation

Examples of restoring division:

a) Divide 7 by 3.

A	Q	M = 0011	
0000	0111	Initial values	
0000	1110	Shift	1
1101		A = A - M	
0000	1110	A = A + M	
0001	1100	Shift	2
1110		A = A - M	
0001	1100	A = A + M	
0011	1000	Shift	3
0000		A = A - M	
0000	1001	$Q_0 = 1$	
0001	0010	Shift	4
1110		A = A - M	
0001	0010	A = A + M	

b) Divide 8 by 3

Initially, the divisor 3= $(0011)_2$is in register M, the dividend 8= $(1000)_2$is in register Q, and register A is zero. Note that subtraction by 3= $(0011)_2$is implemented by adding its 2's complement 1101.

	[M]	0011		
	[A]	0000	[Q]	1000
left shift A/Q		0001		000.
$A = [A] - [M]$	+	1101		
$A < 0$		1110		0000
$A = [A] + [M]$	+	0011		
		0001		0000
left shift A/Q		0010		000.
$A = [A] - [M]$	+	1101		
$A < 0$		1111		0000
$A = [A] + [M]$	+	0011		
		0010		0000
left shift A/Q		0100		000.
$A = [A] - [M]$	+	1101		
$A > 0$		0001		0001
left shift A/Q		0010		001.
$A = [A] - [M]$	+	1101		
$A < 0$		1111		0010
$A = [A] + [M]$	+	0011		
		0010		0010

The quotient $(0010)_2$ =2 is in register Q, and the reminder $(0010)_2$ =2 is in register A

Let's check the take away from this lecture

1) When 1101 is used to divide 100010010 the remainder is _____ .

a) 101

b) 11

c) 0

d) 1

2) In binary division process, a series of performed operations is said to be of

a) Additions

b) Subtractions

c) Multiplications

d) Complement

3) The length of a register is equals to the number of

a) bits

b) words

c) variables

d) functions

Exercise

Q.1 Divide 110/10 using binary division rules.

Q.2 Draw the flowchart for restoring division method.

Q.3 Perform 6/3 using restoring division.

Learning from this lecture: Learners will be able to understand the concept of restoring division method.

Lecture 5

Non-Restoring Division

Algorithm:

S1: Do n times

If the sign of A is 0, shift A and Q left one binary position and subtract M from A; otherwise, shift A and Q left and add M to A.

S2: If the sign of A is 1, add M to A.

The flowchart for non-restoring division is shown in figure 2.5.

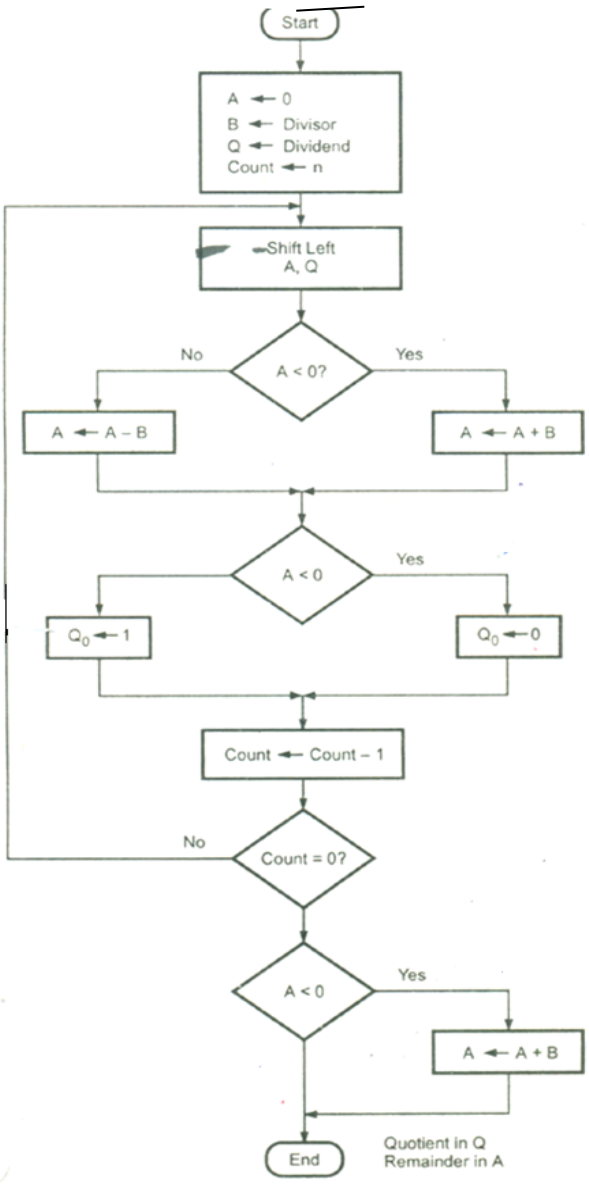


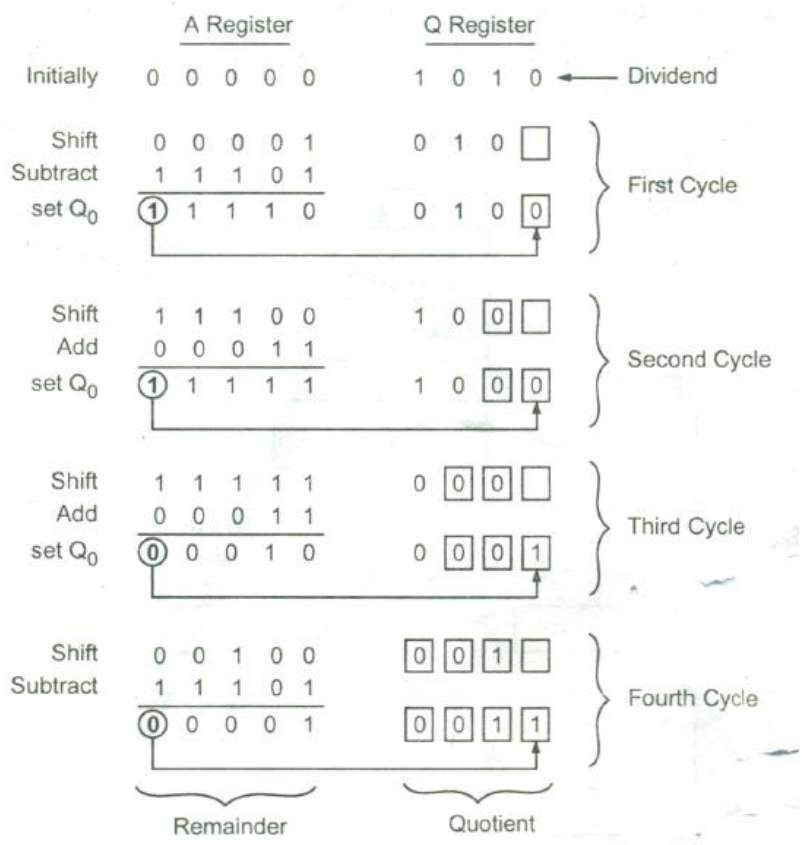
Fig 2.5: Flowchart for non restoring division operation

Consider 4-bit dividend and 2-bit divisor:

Dividend = 1 0 1 0 (10)

Divisor = 0 0 1 1 (3)

The following are the steps involved in the non-restoring binary division.



A non-restoring division example

Let's check the take away from this lecture

1) Which algorithm in mathematics expresses the outcome of the process of division of integers by another:

a) Addition algorithm

b) Multiplication algorithm

c) Division algorithm

d) None of these

2) Which algorithm is based on add/subtract and shift category

a) Restoring division

b) Non-restoring division

c) Booth Multiplication

d) Both a and b

3) Which are the types of addition in the 2's complement system:

a) Both number positive

b) A Positive number and a smaller negative number

c) A negative number and a smaller positive number

d) Both number negative

e) All of these

Exercise

Q.1 Draw the block diagram for the hardware implementation of division.

Q.2 Draw the flowchart for non restoring division method.

Q.3 Perform -6/3 using restoring division.

Learning from this lecture: Learners will be able to understand the concept of non-restoring division method.

Lecture 6

Floating point arithmetic algorithms

1. Floating point representation

i. The Basics

Floating point gets around the limitations of fixed point by using a format similar to scientific notation.

$3.52 \times 10^3 = 1520$

A scientific notation number, as you probably know, consists of a mantissa (3.52 in the example above) a radix (always 10), and an exponent (3 in the example above). Hence, the general format of a scientific notation value is:

$\text{mantissa} \times \text{radix}^{\text{exponent}}$

The *normalized* form always has a mantissa greater than or equal to 1.0, and less than 10.0. We can denormalize the value and express it in many other ways, such as 35.2×10^2, or 0.00325×10^0. For each position we shift the digits of the mantissa relative to the decimal point, we increase or decrease the value of the mantissa by a factor of 10. To compensate for this, we simply increase or decrease the exponent by 1. Denormalizing is necessary when adding scientific notation values:

$$\begin{array}{rl} & 3.52 \times 10^3 \\ + & 1.97 \times 10^5 \\ \hline \end{array}$$

$$\begin{array}{rl} & 1 \\ & 0.0352 \times 10^5 \\ + & 1.97 \times 10^5 \\ \hline & 2.0052 \times 10^5 \end{array}$$

Adjusting the mantissa and exponent is also sometimes necessary to normalize results. For example, $9.9 \times 10^2 + 9.9 \times 10^2$ is 19.8×10^2, which must be normalized to 1.98×10^3.

A binary floating system stores a signed binary mantissa and a signed binary exponent, and usually uses a radix of 2. Using a radix of 2 (or any power of 2) allows us to normalize and denormalize by shifting the binary digits in the mantissa and adjusting the integer exponent on the radix of 2.(Shifting binary digits in the mantissa n bits to the left or right multiplies or divides the mantissa by 2^n.)

$00010_2 \times 2^3 = 01000_2 \times 2^1$.

The standard floating point formats are defined by the IEEE society. The IEEE formats are slightly more complex that necessary to understand floating point in general, so we will start with a simpler example here.

ii. A Simple Floating Point Format

Suppose a 32-bit floating point format has a 24-bit two's complement mantissa, an 8-bit two's complement exponent, and a radix of 2. The general structure is:

mantissa x $2^{exponent}$

Where mantissa is a 24-bit two's complement integer, and exponent is an 8-bit two's complement integer.

The binary format is as follows:

Floating Point Format

Mantissa	**Exponent**
24 bits, two's compliment	8 bits, two's compliment

1. What is the value of the following number?

000000000000000000010010 11111100

The mantissa is 000000000000000000010010, or +(2 + 16) = +18.

The exponent is 11111100 = -(00000011 + 1) = -00000100 = -4.

The value is therefore $+18 \times 2^{-4}$

2. What is the largest positive value we can represent in this system?

The largest positive value will consist of the largest positive mantissa and the largest positive exponent. The largest mantissa is 011111111111111111111111, which in two's complement is $+2^{23}$-1 (+8388607). The largest exponent is 01111111, which in two's complement is $+2^7$-1 (+127).

Hence, the largest positive value is $+8388607 \times 2^{+127} = 1.42 \times 10^{45}$.

3. What is the smallest positive value?

To find the smallest positive value in the form mantissa x $radix^{exponent}$, we choose the smallest positive mantissa, and the smallest negative exponent (the negative exponent with the greatest magnitude).

Since the mantissa is an integer, the smallest positive value possible is 1.

Since the exponent is an 8-bit two's complement value, the smallest negative exponent is 10000000_2, of $-2^7 = -128$.

Hence the smallest positive value is 1×2^{-128}, or $2.93873587706 \times 10^{-39}$.

4. What is the second smallest positive value? What is the difference between the smallest and second smallest?
5. Represent -2.75 in this floating point system.
    a. Convert the number to fixed point binary using the methods described in previous sections:

$-2.75_{10} = -(10.11_2)$

    b. Multiply by $radix^{exponent}$ equal to 1:

$-2.75_{10} = -(10.11_2) \times 2^0$

    c. Shift the binary point to make the mantissa a whole number: -(

e therefore must divide ($radix^{exponent}$) by the same factor:

$-2.75_{10} = -(1011_2) \times 2^{-2}$

    a. Convert the mantissa and exponent into the specified formats (1011_2)
    b. By moving the binary point two places to the right, we multiply the mantissa by 2^2. W two's complement in this case):

Mantissa: -(00000000000000000001011) = 11111111111111111110101

Exponent: $-2_{10} = 11111110$

Binary representation = 11111111111111111110101111111110

2. How many different values can this system represent?

Overflow and Underflow

Overflow occurs when the result of a floating point operation is larger than the largest positive value, or smaller than the smallest negative value. In other words, the magnitude is too large to represent.

Underflow occurs when the result of a floating point operation is smaller than the smallest positive value, or larger than the largest negative value. In other words, the magnitude is too small to represent.

The example 32-bit format above cannot represent values larger than about 10^{45} or smaller than about 10^{-39}.

One technique to avoid overflow and underflow is to alternate operations that increase and decrease intermediate results. Rather than do all the multiplications first, which could cause overflow, or all the divisions first, which could cause underflow, we could alternate multiplications and divisions to moderate the results along the way. Techniques like these must often be used in complex scientific calculations.

2. Floating Point Addition and Subtraction

Consider two floating point numbers:

X = m1· r e1 and

Y m2 .r e2 Assume: e1>= e2

Let us see the rules for addition and subtraction.

Step 1:

Select the number with a smaller exponent and shift its mantissa right, a number of steps equal to the difference in exponents e2 –e1.

For examples, if the numbers are

1.75 x102 and 6.8 x104, then the number 1.75 x102 is selected and converted to 0.0175x104.

Step 2:

Set the exponent of the result equal to the larger exponent.

Step 3:

Perform addition/subtraction on the mantissas and determine the sign of the result.

Step 4:

Normalize the result, if necessary.

Problems in Floating Point Arithmetic:

Mantissa Overflow:

The addition of two mantissas of the same sign may result in a carry out of the most significant bit. If so, the mantissa is shifted right and the exponent is incremented.

Mantissa Underflow:

In the process of aligning mantissas, digits may flow off the right end of the mantissa. In such case truncation methods such as chopping, rounding are used.

Exponent Overflow:

Exponent overflow occurs when a positive exponent exceeds the maximum possible exponent value. In some systems this may be designated as + 00 or - 00.

Exponent Underflow:

Exponent underflow occurs when a negative exponent exceeds the maximum possible exponent value. In such cases, the number is designated as zero.

Flowchart and Algorithm for Floating Point Addition and Subtraction

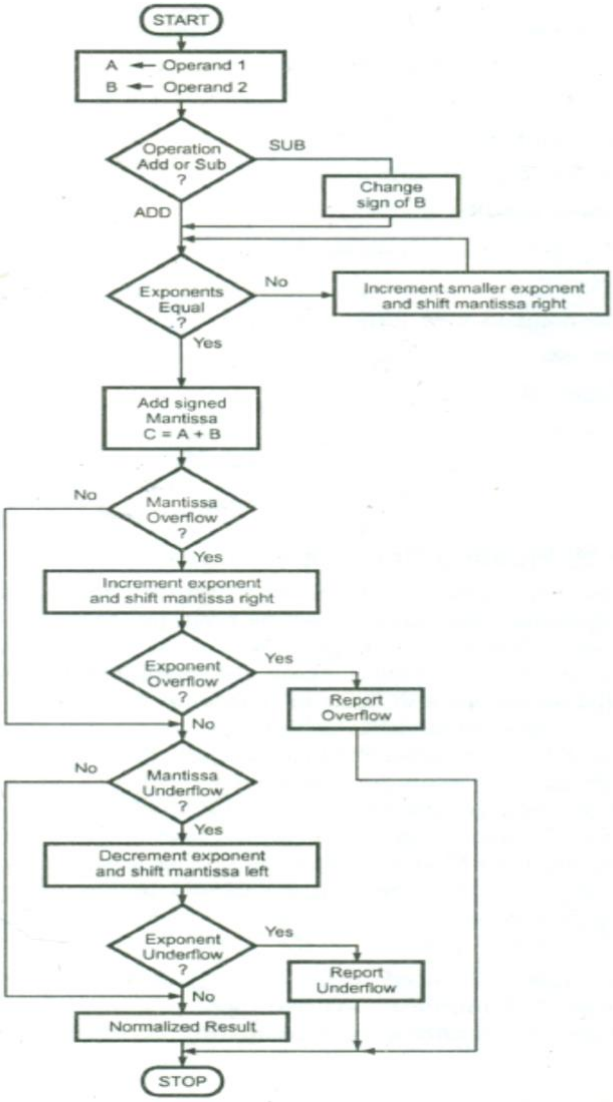


Fig 2.6: Flowchart for floating point addition & subtraction operation

Algorithm

Declare registers:

AM [0 :M - 1] ; For mantissa of A

BM [0 :M - 1] ; For mantissa of B

AE [0 :E - 1] ; For exponent of A

BE [0 : E - 1] ; For exponent of B

E [0 : E - 1] ; For temporary storage of exponent

Step 1: Read Numbers:

AM <- Mantissa of A

BM <- Mantissa of B

AE <- exponent of A

BE <- exponent of B

Step 2: Compare the two exponents: E = AE – BE

Step 3: Equalize

if E <0 then right shift AM ;

E = E + 1 and go to step 3 ;

if E >0 then right shift BM ;

E = E - 1 go to step 3 ;

Step 4: Add Mantissas: AM = AM + BM and E = Max (AE, BE)

Step 5: Check exponent overflow:

if overflow occurs during mantissa addition then

if E = Emax then Report overflow

else

right shift AM

E == E + 1 go to stop

Step 6: Adjust for zero result

if AM == 0 then E == 0 ; go to stop

Step 7: Normalize result

if AM normalized then go to stop

Step 8: Check exponent underflow

if E >Emin then

left shift AM ;

E == E - 1 go to step 7

Step 9: Stop

Let's check the take away from this lecture

1) Floating point numbers are normally a multiples of the size of a

a) Bit

b) Nibble

c) Word

d) Byte

2) The decimal numbers represented in the computer are called as floating point numbers, as the decimal point floats through the number.

a) True

b) False

3) _____ constitutes the representation of the floating number.

a) Sign

b) Significant digits

c) Scale factor

d) All of the mentioned

Exercise

Q.1 Describe the standard way of representing floating point numbers.

Q.2 Draw the flowchart for floating point addition and subtraction.

Q.3 List the problems encountered in floating point arithmetic.

Learning from this lecture: Learners will be able to understand the concept of floating point number representation and the algorithm used for floating point arithmetic.

Lecture 7

IEEE 754 floating point number representation

IEEE 754 floating point formats:

a) Single precision format

b) Double precision format

The IEEE (Institute of Electrical and Electronics Engineers) has produced a standard for floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented, as well as how arithmetic should be carried out on them.

The IEEE 754 standard specifies a **binary32** as having:

- Sign Bit : 1 bit
- Exponent width: 8 bits
- Significand precision : 24 (23 explicitly stored)

Sign bit determines the sign of the number, which is the sign of the significant as well. Exponent is either an 8 bit signed integer from −128 to 127 or an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 binary32 definition. For this case an exponent value of 127 represents the actual zero.

The true significant includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the significand appear in the memory format but the total precision is 24 bits (equivalent to $\log_{10}(2^{24}) \approx 7.225$ decimal digits). The bits are laid out as follows:

Single Precision

The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next eight bits are the exponent bits, 'E', and the final 23 bits are the fraction 'F':

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
```

0 1 8 9 31

Double Precision

The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F':

S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

0 1 11 12 63

IEEE 754 Standards for Single Precision Representation Examples:

Example 1:

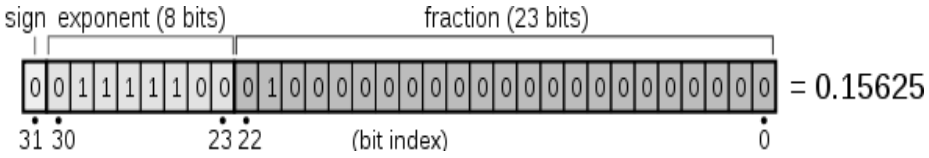
- $1.0_{two} \times 2^{-1}$

31	30~23	22~0
0	11111111	00000000000000000000000

- $1.0_{two} \times 2^{+1}$

31	30~23	22~0
0	00000001	00000000000000000000000

Example 2:



Example 3: Convert 10.4_{ten} to single precision floating point

(1) Normalize
$1010.0110_{two} \times 2^0 = 1.0100110 \times 2^3$
(2) Determine Sign Bit
positive, so S = 0
(3) Determine Exponent:
2^3 so 3 + bias (= 127) = 130 = 10000010_{two}
(4) Determine Significand
drop leading 1 of mantissa, expand to 23 bits = 01001100000000000000000

0	10000010	01001100000000000000000
S	Exponent	Significand

Example 4:

- Show the IEEE 754 representation of -0.75_{ten} in single and double precision
  - Single precision
    - $\rightarrow -3/4_{ten}$ or $-3/2^2_{ten}$
    - $\rightarrow -11_{two}/2^2_{ten}$ or -0.11_{two}
    - $\rightarrow -1.1_{two} \times 2^{-1}$
    - $\rightarrow (-1)^1 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000_{two}) \times 2^{(126-127)}$

31	30~23	22~0
1	01111110	10000000000000000000000
1 bit	8 bits	23 bits

Example 5:

- Double precision
  → $(-1)^1 \times (1 + .1000\ 0000 \ldots 0000\ 0000_{two}) \times 2^{(1022-1023)}$

31	30~20	19~0
1	01111111110	10000000000000000000
1 bit	11 bits	20 bits

00000000 00000000 00000000 00000000
32 bits

Example 6:

- What decimal number is represented by this single precision float?

31	30~23	22~0
1	10000001	01000000000000000000000
1 bit	8 bits	23 bits

  - sign=1
  - exponent=129
  - fraction=1×2^{-2}=1/4 or 0.25

  → $(-1)^S \times (1+\text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$
  $= (-1)^1 \times (1+0.25) \times 2^{(129-127)} = -5.0$

Let's check the take away from this lecture

1) The sign followed by the string of digits is called as _____

a) Significant

b) Determinant

c) Mantissa

d) Exponent

2) In IEEE 32-bit representations, the mantissa of the fraction is said to occupy _____ bits.

a) 24

b) 23

c) 20

d) 16

3) In double precision format, the size of the mantissa is _____

a) 32 bit

b) 52 bit

c) 64 bit

d) 72 bit

Exercise

Q.1 Explain Single Precision IEEE 754 floating point representation.

Q.2 Explain Double Precision IEEE 754 floating point representation.

Q.3 Show the IEEE 754 representation of 10.4 in single precision and double precision format.

Learning from this lecture: Learners will be able to understand the concept of IEEE 754 floating point number representations.

Conclusion

The study of Data representation and arithmetic algorithms helps to understand fixed-point and floating-point number representations. Also this study helps to understand the multiplication and division algorithms used by the ALU of the computer.

Short Answer Questions:

1. Write note on Binary Data representation.

Ans: Computers store all data as patterns of 0's and 1's. Information systems using 0's and 1's are collectively known as *binary information systems*.

Each 0 or 1 in a binary value is called a *bit*, which is short for *binary digit*.

A collection of 8 bits is called a *byte*. A byte is a very common unit of storage for electronic memory. It is usually the smallest chunk of data that programs process, although many languages support processing individual bits as well.

2. Define the algorithm used for signed multiplication

Ans: A powerful algorithm for signed-number multiplication is Booth's algorithm, which generates a 2n-bit product and treats both positive and negative numbers uniformly.

This algorithm suggests that we can reduce the number of operations required for multiplication by representing multiplier as a difference between two numbers.

3. State the difference between restoring and non-restoring algorithm

Ans:

Restoring algorithm:

-Does not need restoring of remainder

-Slower algorithm

Non-restoring algorithm:

-Needs restoring of remainder if it is -ive

-Faster algorithm

Long Answer Questions:

1. Explain Booth's algorithm with flowchart and an example?

Ans: Refer the contents under lecture 3.

2. Explain restoring division method with an example?

Ans: Refer the contents under lecture 4.

3. Explain non restoring division method with an example?

Ans: Refer the contents under lecture 5.

4. Explain the floating point representation IEEE standard 754?

Ans: Refer the contents under lecture 7.

5. Explain the algorithm for floating point addition and subtraction?

Ans: Refer the section 'Floating point Addition and Subtraction' under lecture 8.

Set of Questions for FA/IA/ESE

1. Explain floating point addition & subtraction algorithm with flow chart. (10M)
2. Explain Booth's algorithm with flowchart & example. (10M)
3. Write short note on Floating Point number representation (5M)
4. Explain & solve following problem by using non restoring division algorithm. Hence divide $(10)_{10}$ with $(3)_{10}$. (10M)
5. Explain IEEE format for floating point number representation. (10M)
6. Explain & solve following problem by using restoring division algorithm. Hence divide $(163)_{10}$ with $(11)_{10}$ (10M)
7. Explain Booth's algorithm for signed multiplication. (10M)
8. Perform division of the following numbers using restoring division :- Dividend = 17; Divisor = 03. (10M)
9. Multiply (-7) with (3) by using Booth's algorithm. Give the flow table of multiplication. (10M)
10. Explain non restoring division algorithm for performing 19/4. (10M)
11. Explain multiplication of signed numbers -13 * -5 using Booth's algorithm. (10M)
12. Draw the flow chart for Two's Complement Multiplication. (10M)
13. Using unsigned Binary Division method, divide 7 by 3. (10M)
14. Using Booth's algorithm show the multiplication of -3 * -7. (10M)
15. Multiply (-10) and (-4) using Booths Algorithm. (10M)
16. Express $(35.25)_{10}$ in the IEEE single precision standard of floating point representation. (5M)

References:

1.William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.

2. Carl Hamacher, Zvonko Vrasenic and Safwat Zaky, "Computer Organization", Fifth edition, Tata McGraw-Hill.

Self-assessment

Q.1) Write short note on fixed-point number representation.

Q.2) Write short note on floating-point number representation.

Q.3) Explain the algorithm for unsigned multiplication.

Q.4) Explain the algorithm for signed multiplication.

Q.5) Explain restoring division method.

Q.6) Explain Non-restoring division method.

Q.7) Describe IEEE 754 floating point number representation.

Self-evaluation

Name of Student		
Class		
Roll No.		
Subject		
Module No.		
S.No		Tick Your choice
1.	Do you understand the fixed-point number representation?	o Yes o No
2.	Do you understand the floating-point number representation?	o Yes o No
3.	Do you understand the algorithms used for binary multiplication and division?	o Yes o No
4.	Do you understand the difference between restoring and non restoring division?	o Yes o No
5.	Do you understand the IEEE 754 floating point number representation?	o Yes o No
6.	Do you understand module 2 ?	o Yes, Completely. o Partialy. o No, Not at all.