Lecture : 15

Introduction to Computing

Instructor Name : Aasim Ali

Punjab University College of Information Technology

Representing Fractions

IEEE Standard 754

- The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). Many hardware floating point units use the IEEE 754 standard. The current version, IEEE 754-2008 published in August 2008, includes nearly all of the original IEEE 754-1985 standard and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (IEEE 854-1987). The international standard ISO/IEC/IEEE 60559:2011 (with identical content to IEEE 754) has been approved for adoption through JTC1/SC 25 under the ISO/IEEE PSDO Agreement[1] and published

Representing Fractions

IEEE 754 Binary Floating Point structure

IEEE 754 Binary Floating Point is a 32-bit representation for floating point numerals. The 32-bit representation consists of three parts:

- The first bit is used to indicate if the number is positive or negative.
- The next 8 bits are used to indicate the exponent of the number.
- Last 23 bits are used for the fraction.

Converting Fractions

- The first step in the conversion is determining the first bit. If the decimal digit is positive then this bit is 0, if the decimal digit is negative then this bit is 1.
- The next eight digits are used to express the exponent.
- The final 23 digits are used to express the fraction. This gives you 2^{23} precision.

  For example we have a number: -210.25

  As this number is negative, its left most bit will be 1. Now we will convert 210 in binary. 210 is 11010010 or 128+64+16+2

  Expressed as:

  $1* 2^7 + 1* 2^6 + 0* 2^5 + 1* 2^4 + 0* 2^3 + 0* 2^2 + 1* 2^1 + 0* 2^0$

Converting Fractions

- Next we need to convert the decimal part. To do this we have to convert the number into a binary sequence that equals $a*2^{-1} + a*2^{-2}$ and so on. This series basically means 1/2 + 1/4 + 1/8 + 1/16 and so on.
- In this example we have 0.25 which is equal to 1/4. we can written as: $0*2^{-1} + 1*2^{-2}$ So our final number is 11010010.01

- The next step is to normalize this number so that only one non zero decimal place is in the number. To do this you must shift the decimal place 7 positions to the left. This process leaves us with the number 1.101001001. This is then padded with 0's to fill in the full 23 bits - leaving us with 10100100100000000000000.

- So we now have the first bit, and the last 23 bits of the 32 bit sequence. We must now derive the middle 8 bits.

Converting Fractions

- To derive the middle 8 bits we take our exponent (7) and add 127 (the maximum number you can express with 8 bits (2^{8-1} or the numbers 0 to 127)) which gives us 134. This is 10000110 or $1*2^7 + 1*2^2 + 1*2^1$ or 128+4+2.
- Taken as whole our bit sequence will be:

  1 10000110 10100100100000000000000

- We can convert this bit sequence back into a number by reversing the process.

  1 01111101 00000000000000000000000

  sign: -ve; exponent: 125-127 = -2; mantissa: 001.0000

  which means -001.0000×2^{-2} ===> -0.01 in binary

  which is -0.25 in the decimal system

Coding Scheme

What is Coding Scheme

- The combinations of 0s and 1s that represent characters are defined by patterns called a coding scheme. In one coding scheme, the number 4 is represented as 00110100, the number 6 as 00110110, and the capital letter E as 01000101. Computer uses a coding scheme to represent characters.

- Two coding schemes that represent characters in a computer are ASCII and EBCDIC. The American Standard Code for Information Interchange, or ASCII, coding scheme is the most widely used coding scheme to represent data. The Extended Binary Coded Decimal Interchange Code, or EBCDIC, coding scheme sometimes is used on mainframe computers and high-end servers.

Coding Scheme

Binary Coded Decimal

- BCD is also known as 8421 coding scheme. BCD is a class of binary encodings of decimal numbers where each decimal digit is represented by a fixed number of bits, usually four bits. A number with d decimal digits will require 4d bits in BCD.

Decimal Digit	BCD 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Coding Scheme

- When a computer uses the ASCII or EBCDIC coding scheme, it stores each represented character in one byte of memory. Other binary formats exist, however, that the computer sometimes uses to represent numeric data. For example, a computer may store, or pack, two numeric characters in one byte of memory. The computer uses these binary formats to increase storage and processing efficiency.

ASCII	SYMBOL	EBCDIC
00110000	0	11110000
00110001	1	11110001
00110010	2	11110010
00110011	3	11110011
00110100	4	11110100
00110101	5	11110101
00110110	6	11110110
00110111	7	11110111
00111000	8	11111000
00111001	9	11111001
01000001	A	11000001
01000010	B	11000010
01000011	C	11000011
01000100	D	11000100
01000101	E	11000101
01000110	F	11000110
01000111	G	11000111
01001000	H	11001000
01001001	I	11001001
01001010	J	11010001
01001011	K	11010010
01001100	L	11010011
01001101	M	11010100
01001110	N	11010101
01001111	O	11010110
01010000	P	11010111
01010001	Q	11011000
01010010	R	11011001

Coding Scheme

Unicode

- The 256 characters and symbols that are represented by ASCII and EBCDIC codes are sufficient for English and western European languages but are not large enough for Asian and other languages that use different alphabets. Further compounding the problem is that many of these languages use symbols, called ideograms, to represent multiple words and ideas.

- One solution to this situation is Unicode. Unicode was initially a 16-bit coding scheme that deemed to be enough for representing all the world's current languages, as well as classic and historical languages, in more than 65,000 characters and symbols.

Binary Logic

Binary logic consists of binary variables and logical operations. The variables are designated by letters of the alphabet such as A,B, C, x, y, z etc. with each variable having two and only tow distinct possible values: 1 and 0.

There are three basic logical operations: AND, OR and NOT.

1. **AND:** This operations is represented by a dot or by the absence of an operator. For example, x.y=z or xy=z is read as "x AND y is equal to z." The logical operation AND is interpreted to mean that z=1 if and only if x=1 and y=1; otherwise z=0 (x and y are binary variables and can be equal to either 1 or 0, nothing else.)

Binary Logic

2. **OR:** This operations is represented by plus sign. For example, x + y = z is read as "x OR y is equal to z." The logical operation OR is interpreted to mean that z=1 if x=1 or if y=1 or if both x=1 and y=1. If both x=0 and y=0, then z=0.

3. **NOT:** This operation is represented by a prime (sometimes by a bar). For example, a'=b (or ā=b) is read as "not a is equal to b," meaning that b is what a is not. In other words, if a=1 then b=0, but if a=0, then b=1.

- **Operations AND and OR have some similarities to multiplication and addition respectively.**

Binary Logic

4. **XOR:** This operations is represented by $\oplus$ sign. For example, $x \oplus y = z$ is read as "x XOR y is equal to z." The logical operation XOR is interpreted to mean that z=1 if x=1 and y=0 or if x=0 and y=1. If x=1 and y=1 or x=0 and y=0 then z=0.

5. **XNOR:** This operation is represented by a $\equiv$. For example, $x \equiv y = z$ is read as "x XNOR y is equal to z." The logical operation XOR is interpreted to mean that z=1 if x=1 and y=1 or if x=0 and y=0. If x=1 and y=0 or x=0 and y=1 then z=0.

Binary Logic

Truth Table:

- A truth table is a table of all possible combinations of the variables showing the relation between the values that the variables may take and the result of the operation.

- For example, the truth tables for the operations AND and OR with variable x and y are obtained by listing all possible values that the variables may have when combined in pairs. The result of the operation for each combination is then listed in a separate row.

Binary Logic

Truth Tables of Logical Operations

AND

x	y	x . y
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x′
0	1
1	0

XOR

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

Another Notation:

NOT A can be written as $\overline{A}$

A AND B can be written as $A \cdot B$

A OR B can be written as $A + B$

Homework

Homework From Notes:

- Q: 1-1 to 1-11 Complete
- Q: 1-12 Part (a)
- Q: 1-13
- Q: 1-15 Part (b) and (c)
- Q: 1-16
- Q: 1-18
- Q: 1-20