COLLEGE OF COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY

QUEZON CITY
UNIVERSITY

Week 10
Arithmetic Unit (Part 3)

AR101 - BSIT

Computer Architecture and Organization

LEARNING OUTCOMES:

At the end of the session, the students should be able to:

- Apply fast Multiplication Technique.
- Differentiate the hardware circuitry of Integer Division.
- Differentiate single-precision and double-precision of floating-point numbers.
- Apply Arithmetic Operations in floating point numbers.

Multiplication Algorithm

A **multiplication algorithm** is an algorithm (or method) to multiply two numbers. Depending on the size of the numbers, different algorithms are used. Efficient multiplication algorithms have existed since the advent of the decimal system.

Computers initially used a very similar shift and add algorithm in base 2, but modern processors have optimized circuitry for fast multiplications using more efficient algorithms, at the price of a more complex hardware realization.

Multiplication Algorithm

Some chips implement long multiplication, in hardware or in microcode, for various integer and **floating-point word** sizes. In arbitrary-precision arithmetic, it is common to use long multiplication with the base set to $\mathbf{2^w}$, where *w* is the number of bits in a word, for multiplying relatively small numbers.

To multiply two numbers with *n* digits using this method, one needs about n^2 operations. More formally: using a natural size metric of number of digits, the time complexity of multiplying two *n*-digit numbers using long multiplication is $\Theta(n^2)$.

Multiplication Algorithm

When implemented in software, long multiplication algorithms must deal with overflow during additions, which can be expensive. A typical solution is to represent the number in a small base, *b*, such that, for example, 8*b* is a representable machine integer.

Several additions can then be performed before an overflow occurs. When the number becomes too large, we add part of it to the result, or we carry and map the remaining part back to a number that is less than *b*. This process is called ***normalization***

Multiplication Algorithm

Computers used a "shift and add" algorithm to multiply small integers. Both base 2 long multiplication and base 2 peasant multiplication reduce to this same algorithm.

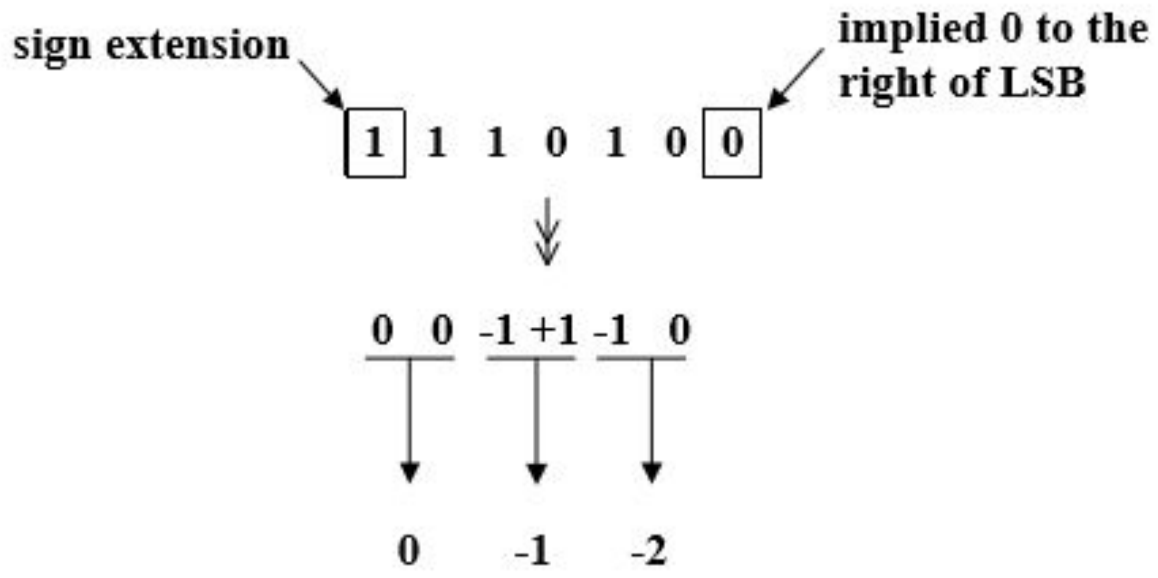
In base 2, multiplying by the single digit of the multiplier reduces to a simple series of logical **AND** operations. Each partial product is added to a running sum as soon as each partial product is computed.

Most currently available microprocessors implement this or other similar algorithms (such as **Booth encoding**) for various integer and floating-point sizes in hardware multipliers or in microcode.

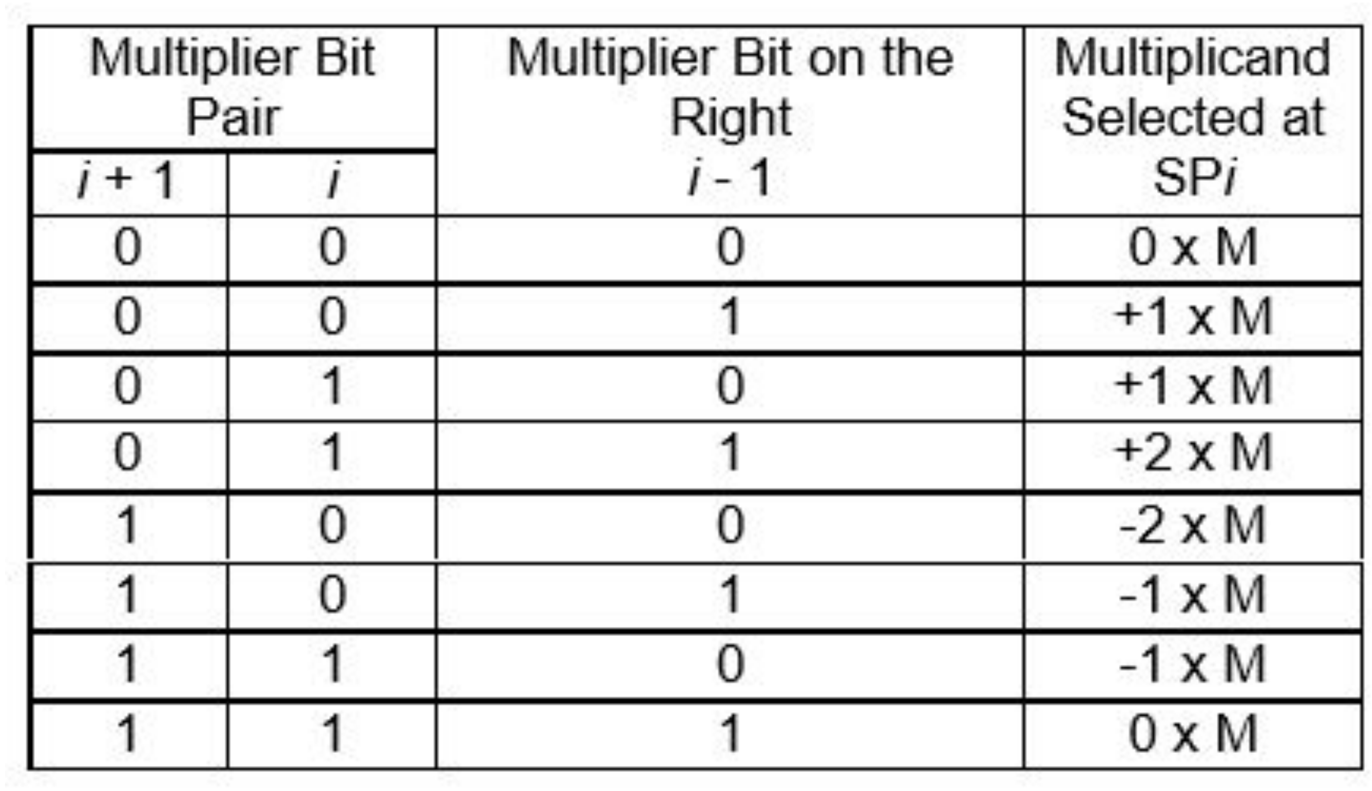
Fast Multiplication

- A technique called bit-pair recording halves the maximum number of summands. It is derived directly from the Booth algorithm and twice as fast as the worst-case Booth algorithm situation.

Bit-Pair Recording



Bit-Pair Recording



Multiplier Bit Pair		Multiplier Bit on the Right	Multiplicand Selected at
i + 1	*i*	*i* - 1	SP*i*
0	0	0	0 x M
0	0	1	+1 x M
0	1	0	+1 x M
0	1	1	+2 x M
1	0	0	-2 x M
1	0	1	-1 x M
1	1	0	-1 x M
1	1	1	0 x M

Bit-Pair Recording Examples

- Derive the bit-pair recorder of the following

A. 0 1 1 0 1

B. 1 0 1 1 1

C. 0 1 1 1 0

Bit-Pair Recording: Multiplication Examples

- Perform the following operation using bit-pair recording:

A.

$$\begin{array}{lcl} & 0\,1\,1\,0\,1 & (+13) \\ \times & 1\,1\,0\,1\,0 & (-6) \end{array}$$

B.

$$\begin{array}{lcl} & 0\,1\,0\,1\,0 & (+10) \\ \times & 1\,0\,1\,1\,1 & (-9) \end{array}$$

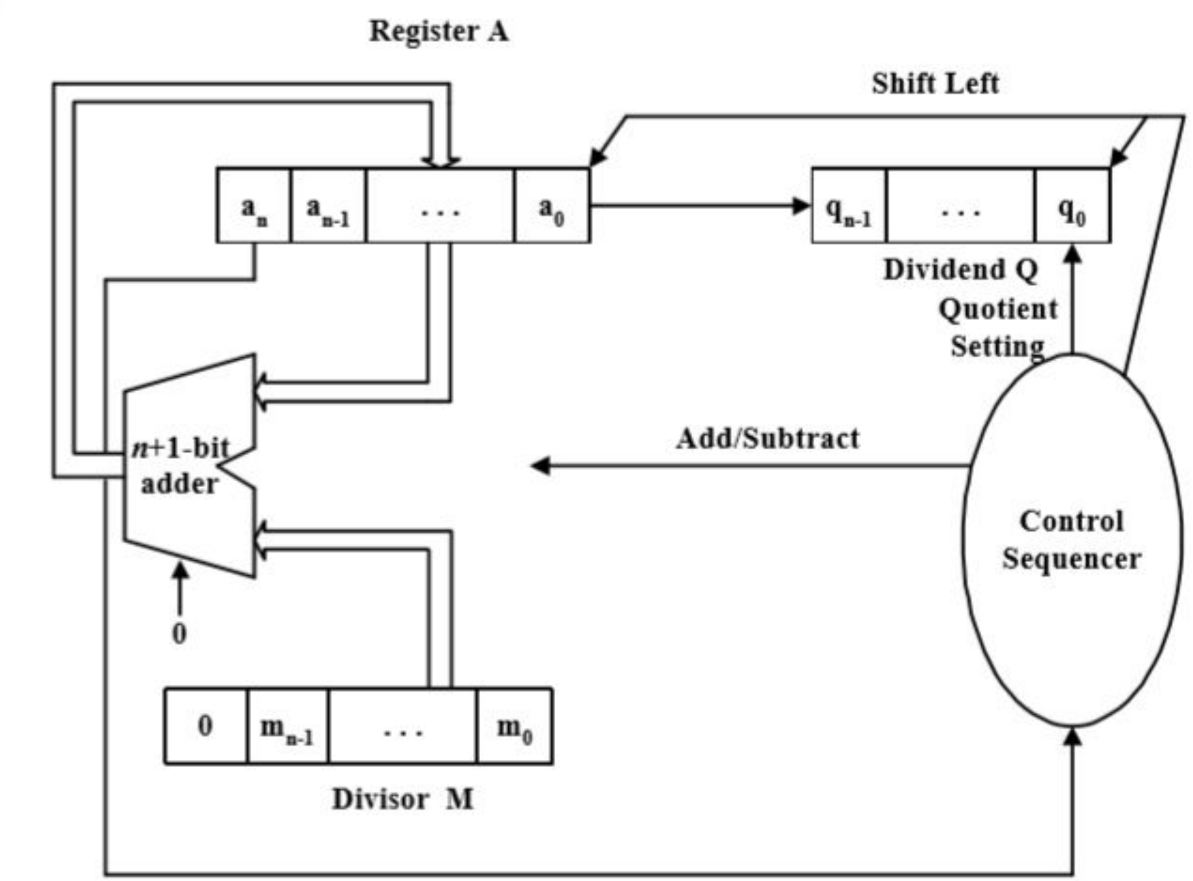
Integer Division

- Longhand division examples

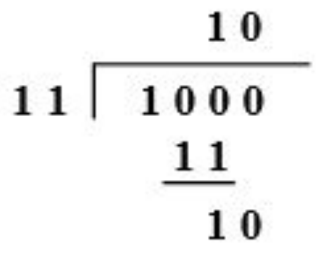
```
         21                           10101
     ________                   __________
 13 | 274                 1101 | 100010010
      26                          1101
      ---                         ----
       14                          10000
       13                           1101
       --                           ----
        1                             1110
                                      1101
                                      ----
                                         1
```

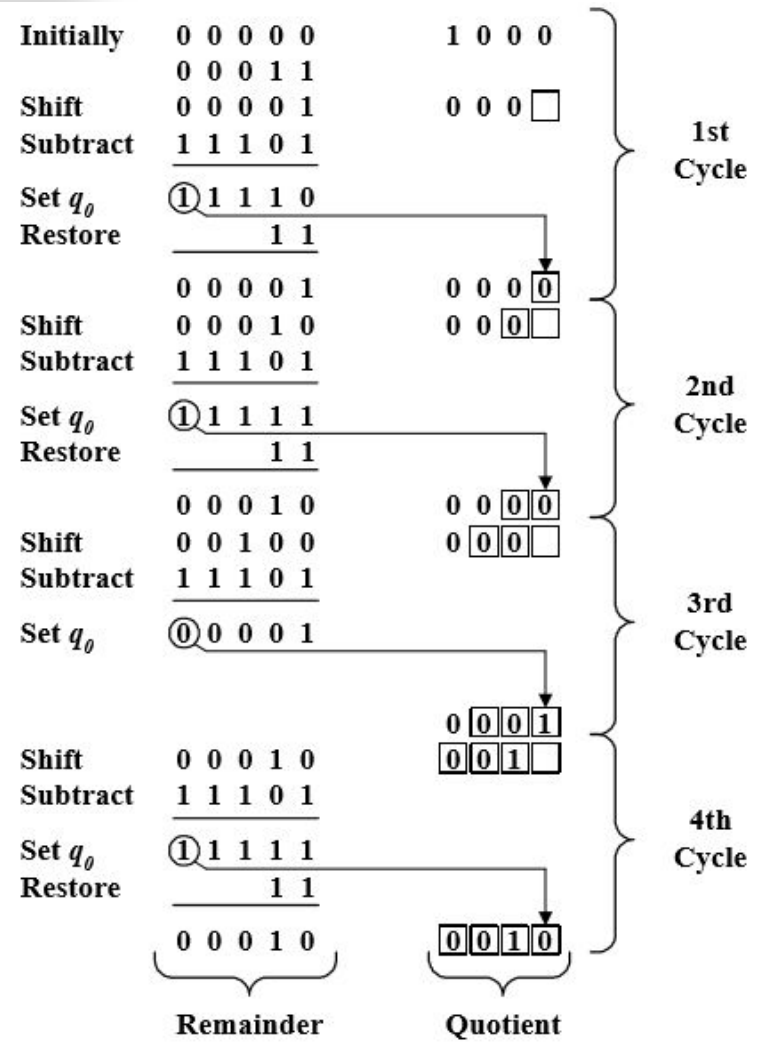
Restoring Division

- The logic circuit arrangement that implements restoring-division technique is shown below:



Restoring Division Example

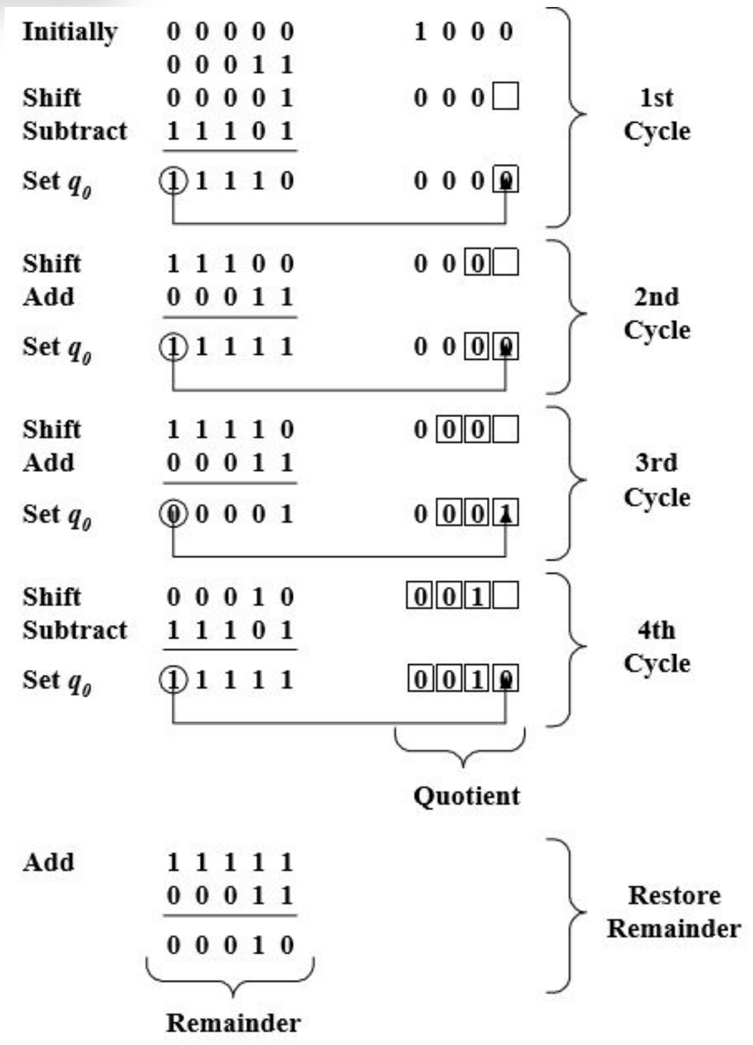




Non-Restoring Division

- If A is positive, we shift left and subtract M; that is, we perform 2A – M.
- If A is negative, we restore it by performing A + M. and then we shift it left and subtract M. This is equivalent to performing 2A + M.
- The q0 bit is appropriately set to 0 or 1 after the correct operation has been performed.

Non-Restoring Division Example



```
Initially   0 0 0 0 0        1 0 0 0
            0 0 0 1 1
Shift       0 0 0 0 1        0 0 0 □        1st
Subtract    1 1 1 0 1                       Cycle
Set q0      1 1 1 1 0        0 0 0 0

Shift       1 1 1 0 0        0 0 0 □        2nd
Add         0 0 0 1 1                       Cycle
Set q0      1 1 1 1 1        0 0 0 0

Shift       1 1 1 1 0        0 0 0 □        3rd
Add         0 0 0 1 1                       Cycle
Set q0      0 0 0 0 1        0 0 0 1

Shift       0 0 0 1 0        0 0 1 □        4th
Subtract    1 1 1 0 1                       Cycle
Set q0      1 1 1 1 1        0 0 1 0
                             Quotient

Add         1 1 1 1 1                       Restore
            0 0 0 1 1                       Remainder
            0 0 0 1 0
            Remainder
```

Floating-Point Numbers and Operation

- In the 2's-complement system, signed value F, represented by the n-bit binary fraction:

$$B = b_{0+}.\ B_{-1}\ b_{-2} \ldots\ b_{-(n-1)}$$

Is given by:

$$F(B) = -\ b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \ldots + b_{-(n-1)} \times 2^{-(n-1)}$$

Where the range of F is:

$$-1 \leq F \leq 1 - 2^{-(n-1)}$$

- Consider the range of values possible representable in a 32-bit, signed, fixed-point format. Interpreted as integers, the value range is approximately 0 to $\pm\ 2.15 \times 10^9$. If interpreted as fractions, the range is approximately $\pm\ 4.55 \times 10^{-10}$ to ± 1.

Floating-Point Numbers and Operation

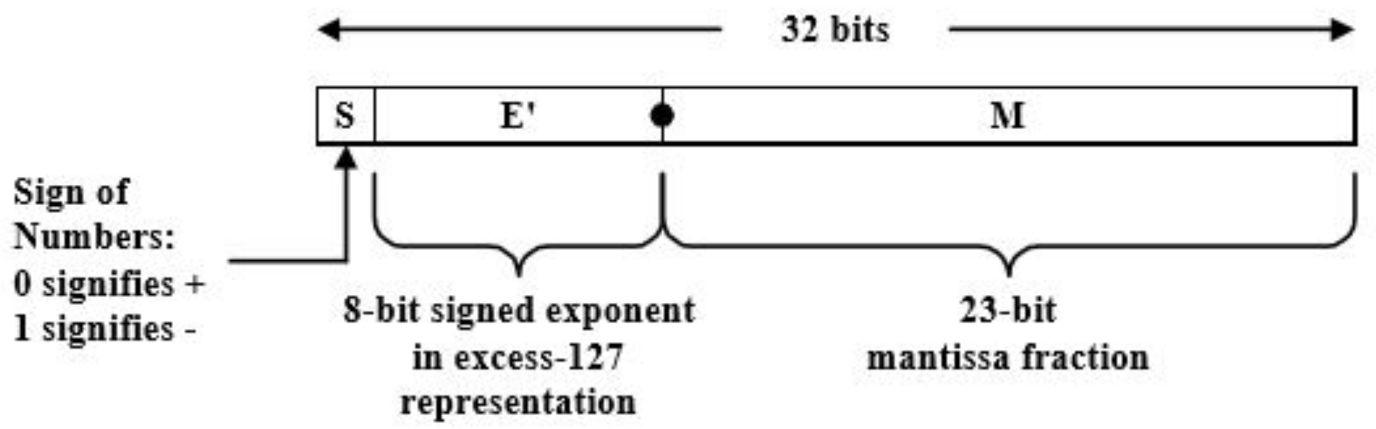
- In the decimal scientific notation numbers may be written as 6.0246×10^{23}, 6.6254×10^{-27}, -1.0341×10^{2}, -7.3000×10^{-14} and so on.

- By convention, when the decimal point is placed to the right of the first (nonzero) significant digit, the number is said to be **normalized.**

- A floating-point representation is one in which a number is represented by its sign, a string of significant digits, commonly called **mantissa,** and an exponent to implied base for the scale factor.

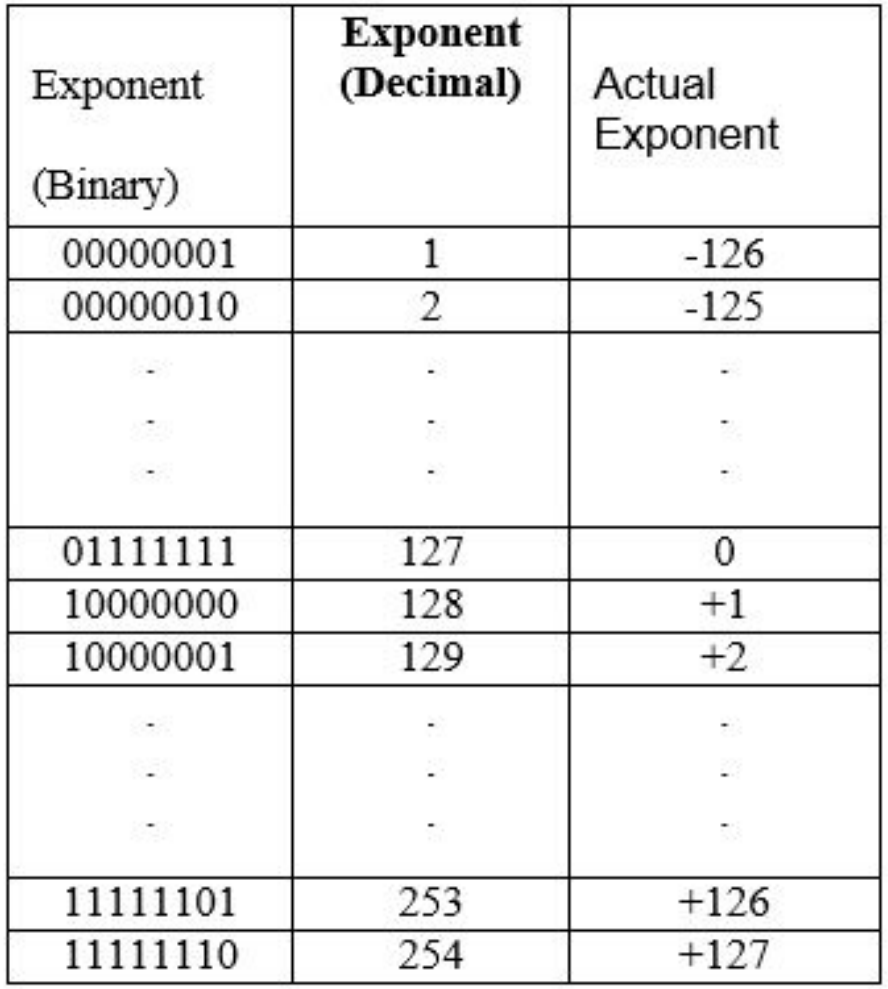
Floating-Point Number: Single-Precision

- The standard for representing floating-point numbers in 32 bits has been developed and specified in detail by the IEEE.



value represented = $\pm 1.M \times 2^{E'-127}$

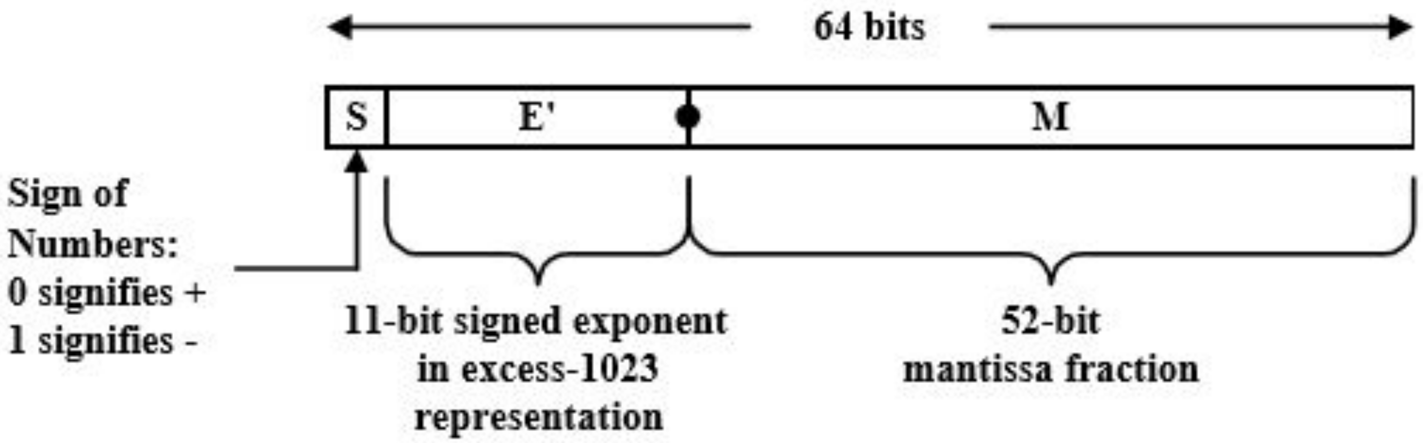
Examples



Exponent (Binary)	**Exponent (Decimal)**	Actual Exponent
00000001	1	-126
00000010	2	-125
- - -	- - -	- - -
01111111	127	0
10000000	128	+1
10000001	129	+2
- - -	- - -	- - -
11111101	253	+126
11111110	254	+127

Floating-Point Number: Double-Precision

- To provide more precision and range floating-point numbers, the IEEE standard also specifies a **double-precision** format for floating-point number representation:


64 bits
S
E'
M
Sign of Numbers: 0 signifies + 1 signifies -
11-bit signed exponent in excess-1023 representation
52-bit mantissa fraction

Examples

- Example # 1:

  Represent 5.75_{10} using single-precision

- Example # 2:

  Convert 50_{10} to IEEE floating-point standard using double-precision format

End of Lesson…

References:

[1] Multiplication algorithm – https://en.wikipedia.org/wiki/Multiplication_algorithm

[2] Williams, Stallings (2010), Computer Organization and Architecture: Designing for Performance (8th Edition, Prentice Hall, New Jersey).

[3] Stalling, William, Computer Organization and Architecture: Principles of Structure and Function (4th Edition)