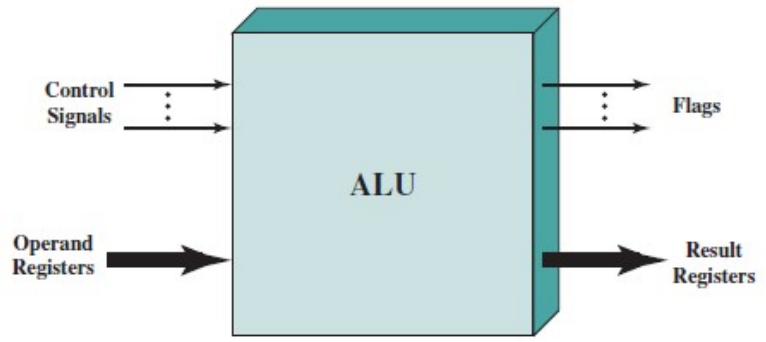
Arithmetic and Logic Unit



ALU Inputs and Outputs

Computer Arithmetic performed on two very different types of numbers: Integer and Floating Point

Integer:

An n-bit sequence of binary digits is interpreted as an unsigned integer A:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Sign Magnitude Representation:

An alternative convention, that involve treating the most significand (leftmost) bit in the word as sign bit.
If the sign bit is 0, the number is positive if the sign bit is 1, the number is negative.
In sign magnitude the (Rightmost) n-1 bits in a word of n bit holds the magnitude of the integer.

Drawbacks:

- Addition and Subtraction require consideration of both sig and magnitude of the number.
- Another problem is two representation for '0'

  + 0 = 0000 0000
  - 0 = 1000 0000

Two's Complement Representation:

Like Sign Magnitude, twos complement representation also uses the Most Significand bit as sign bit.

Range	-2^{n-1} through $2^{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract B from A, take the twos complement of B and add it to A.

The MSB is Negated:

Use of Value Box for Conversion between Twos Complement Binary to Decimal.

−128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

$-128 \quad +2 \quad +1 \quad = -125$

Convert binary 10000011 to decimal

Range Extension:

Rule is to move the sign bit to the new leftmost position and fill in with copies of the sign bit:

For Positive: fill it with '0'

For Negative: fill it with '1'

Integers are Fixed Point Representation:

Radix point is fixed and assumed to be to the right of the rightmost digit.

Integer Arithmetic:

Negation:

Rules

1) Take the complement of each bit of integer (including the sign bit)
2) Treating the result an unsigned integer add 1.

Special Case to Consider:

1) Consider A= 0000 0000, In that case: Twos complement of 0 = 1 0000 0000
There is a carry out of the MSB position which is ignored. Negation of 0 is 0.

2) If we take negation of bit pattern of 1 followed by all zeros, we get back the same number.

Addition:

- Addition proceeds as if the two numbers are unsigned integers.
- If a carry occurs, it is ignored.
- If result is larger than that can be held in the word size, it is said to be overflow condition , ALU must signal this fact so that no attempt is made to use the result.
- To detect overflow 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.

  Overflow can occur whether or not there is a carry.

Subtraction:

To subtract one number (subtrahend) from another (minuend) take the Twos complement (negation) of the subtrahend and add it to the minuend.

Multiplication:

Unsigned Integers:

i) Multiplication involves generation of partial products one for each bit / digit in the Multiplier. These partial products are then summed to produce the final product.

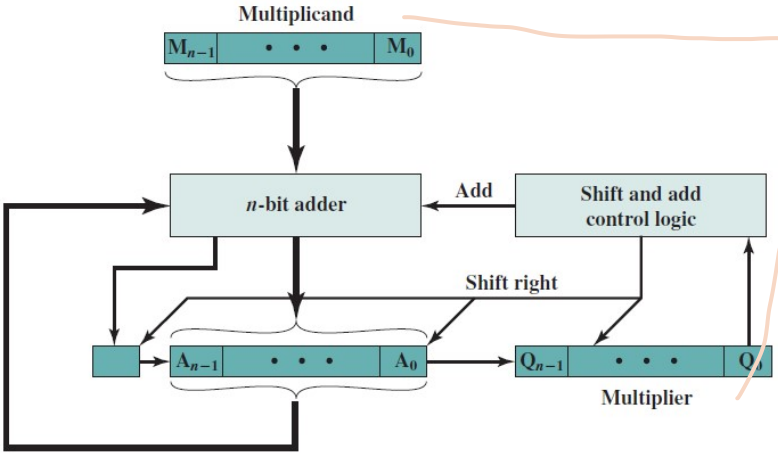
ii) Partial products can be easily defined for binary data:
When the Multiplier bit is 0, the partial product is 0
When the Multiplier bit is 1, the partial product is Multiplicand.

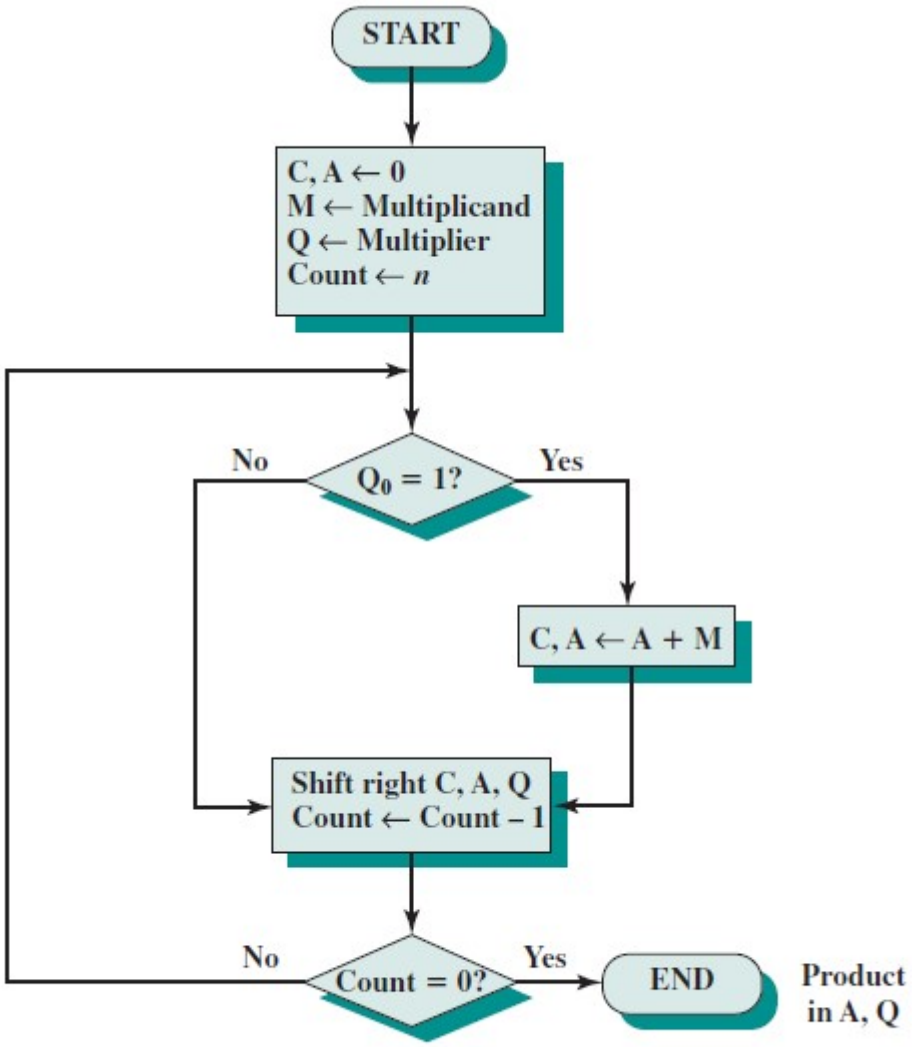
iii) The total product is produced by summing the partial products. Each successive partial product is shifted one position to the left relative to the preceding partial product.

iv) Multiplication of two **n** bit integers results in a product of up to 2n bit in length.

v) In computerized multiplication we perform running addition. This eliminates the need for storage of all partial products.

vi) For each 1 on the Multiplier an Add & a Shift operation is required and for 0 a Shift operation is required.

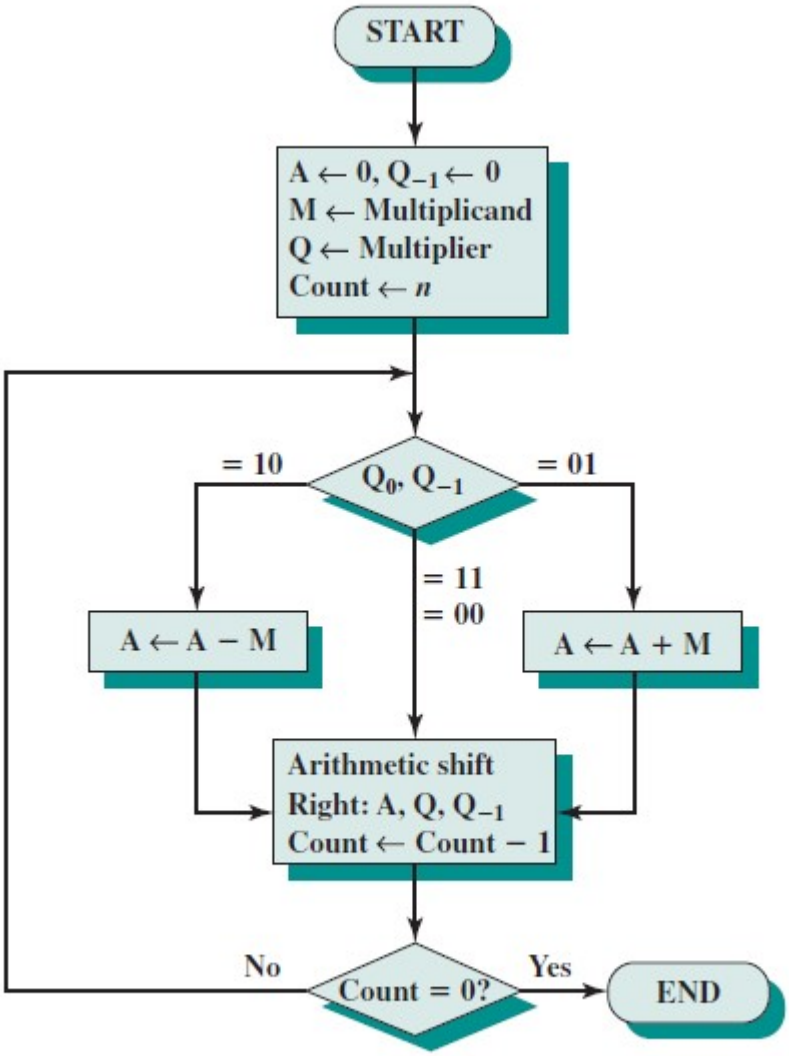




Flowchart for Unsigned Binary Multiplication

Twos Complement Multiplication:

Addition and Subtraction on Twos complement are performed by treating them unsigned integer, but this cannot be done with multiplication.
For Multiplication of two numbers represented in Twos Complement notation Booth's Algorithm is used:

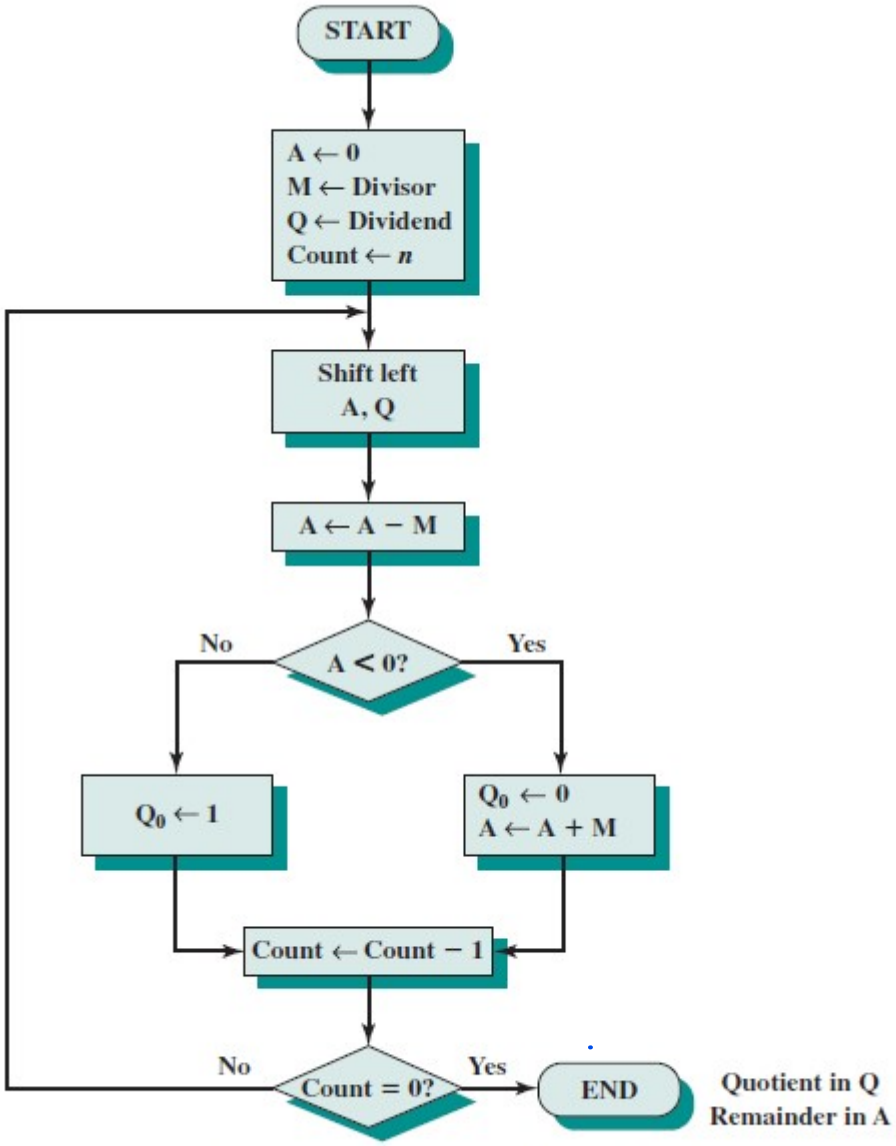


Booth's Algorithm for Twos Complement Multiplication

Arithmetic Shift Right preserves the sign bit by not only shifting A_{n-1} to A_{n-2} but also keeping it there.

Division:

- Division is more complex then multiplication.
- Operation involve repetitive shifting and addition or subtraction.



Flowchart for Unsigned Binary Division

Restoring Division Algorithm

The algorithm assumes that the divisor V and the dividend D are both positive and:

a) | V | < | D |

b) If | V | = | D | then Quotient Q = 1 and Remainder R = 0

c) If | V | > | D | then Quotient Q = 0 and Remainder R = D

To do Twos Complement Division we need to convert the operands into unsigned values and at the end to account for the signs do complementation where needed:

Sign of Remainder Sign (R) = Sign (D)

Sign of Quotient Sign (Q) = Sign (D) x Sign (V)

Floating Point Representation:

- Floating point numbers can be represented in scientific notation
- We dynamically slide the decimal (Radix) point to a convenient location and use the exponent of 10 or 2 (depending on base of number system) to keep track of decimal point.
- This allows a range of very large and very small numbers to be represented with only few digits.
- We can represented a n umber in the form:

  $\pm S \times B^{\pm E}$

  Sign: Plus or Minus
  Significand S
  Exponent E
- Base B is implicit and need not to be stored as it is same for all numbers.
- The exponent is stored in biased representation. (A fixed value, called the bias, is subtracted from the filled to get the true exponent value). Typically bias equals $2^{k-1} -1$ where k is the number of bits in binary exponent. (8 bit exponent field yield the number 0 through 255 with bias $2^{8-1} -1 = 127$ true exponent value are in range - 127 to + 128)
- Advantage of bias representation is that non negative Floating Point Number can be treated as integer for comparison purpose.
- The leftmost bit store the sign of the number.
- Floating point numbers are stored after Normalization.
- A normal number is one in which the MSB of significand is 1. A Normal number is of form:

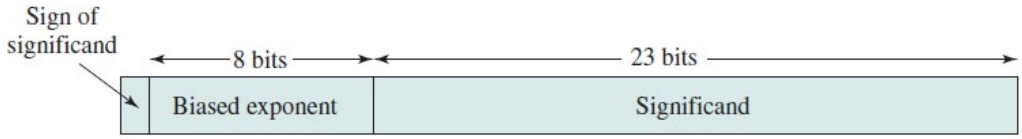
  $\pm 1 . b b b b b \ldots . b \times 2^{\pm E}$

  where b is either binary digit (0 or 1)
- Because the MSB is always 1, again it is unnecessary to store it.
- Given a non-normalize number, the number is normalized by shifting the radix point to the right of the left most 1 bit and adjusting the exponent accordingly.

Summary:

The sign is stored in the first bit of the word.

- The first bit of the true significand is always 1 and need not to be stored in the significand field.
- The value of bias (127 is case of 8 bit exponent) is added to the true exponent to be stored in exponent field.
- The base is 2 need not to be stored.



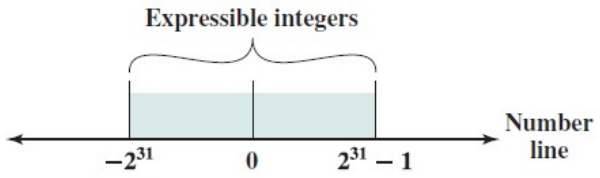
(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$
$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$
$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$
$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$

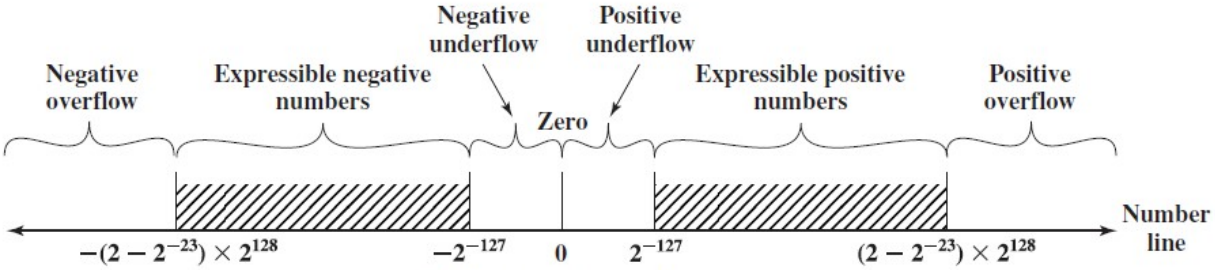
(b) Examples

Typical 32-Bit Floating-Point Format

- In a 32-bit word length of 23 bit significand , 8 bit for biased exponent and one bit for sign, 2^{32} different numbers can be represented in range:
  - Negative Numbers: $-(2-2^{-23}) \times 2^{128}$ and -2^{-127}
  - Positive Numbers: $+2^{-127}$ and $+(2-2^{-23}) \times 2^{128}$



(a) Twos complement integers

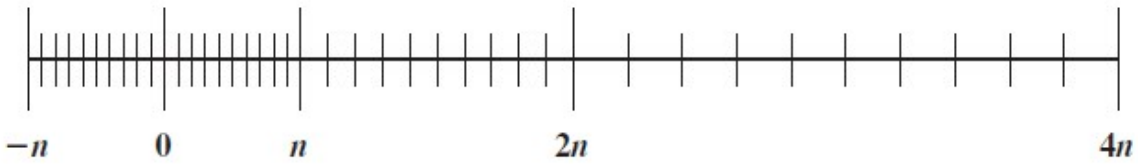


(b) Floating-point numbers

Expressible Numbers in Typical 32-Bit Formats

Range and Precision:

- A large exponent field yield a large range.
- Precision increase by increasing buts for significand.
- A large exponent base gives a greater range at the expense of less precision.



Density of Floating-Point Numbers

Floating Point Representation in IEEE Format:

- An exponent of zero together with a fraction of zero represents +ve or – ve zero (depending on sign bit)
- An exponent of all ones together with fraction of zero represent positive or negative infinity.
  (In floating point arithmetic overflow should be treated as error or represented as ∞)
- An exponent of zero together with a non-zero fraction represent a **subnormal** number (bit to the left of radix point is 0 and true exponent is – 126)
- An exponent of all ones together with a non-zero fraction is the value NaN (Not a Number) use to represent exception conditions)

Floating Point Arithmetic:

For Addition and Subtraction : Both Operands should have the same exponent value (Process of making exponent same is called **Alignment**)
Multiplication and Division are straight forward.

Let $X = X_S \times B^{Xe}$
$Y = Y_S \times B^{Ye}$

Then

$X + Y = (X_S \times B^{Xe-Ye} + Y_S) \times B^{Ye}$
$X - Y = (X_S \times B^{Xe-Ye} - Y_S) \times B^{Ye}$

$X \times Y = (X_S \times Y_S) \times B^{Xe+Ye}$
$X / Y = (X_S / Y_S) \times B^{Xe-Ye}$

Conditions:

1) Exponent Overflow: This may be designated as $+\infty$ or $-\infty$
2) Exponent Underflow: This may be represented as 0.
3) Significand Underflow: In the process of aligning significand digits may flow off the right end of the significand.
4) Significand Overflow: Addition of 2 Significand of same sign may result in a carry out of MSB. This can be fixed by realignment.

Addition and Subtraction:

Four basic phases of the algorithm for addition and subtraction:

1) Check for Zeros
2) Alight the Significands
3) Add or Subtract the Significands

4) Normalize the Result

If either operand is 0, the other is reported as the result (Sign of the Subtrahend is changed if operation is Subtraction)

Precision Consideration

Guard Bits:

Registers in ALU contain contain additional bits for holding implied bit and used to pad out the right end of the Significand with 0s.

*****