One's Complement

Example: – 4 – 3 = – 7

4 in binary = 0100.

Flipping the bits, you get –4 (1011) in binary.

3 in binary = 0011.

Flipping the bits, you get –3 (1100) in binary.

```
  1011 (11 in decimal, or 15-4)
+ 1100 (12 in decimal, or 15-3)
---------------------------------------
 1,0111 (23 in decimal (15+15-7))
```

So now take the extra 1 and remove it from the 5th spot and add it to the remainder

```
   0111
+     1
---------------------------------------
   1000 (-7 in 1's comp)
```

Recovery of the Numbers

1's Complement

Let $f(x) = 2^n - 1 - x$

Theorem: $f(f(x)) = x$

Proof: $f(f(x))$

$= f(2^n - 1 - x)$

$= 2^n - 1 - (2^n - 1 - x)$

$= x$

2's Complement

Let $g(x) = 2^n - x$

Theorem: $g(g(x)) = x$

Proof: $g(g(x))$

$= g(2^n - x)$

$= 2^n - (2^n - x)$

$= x$

Floating point numbers

Real Numbers

- Two's complement representation deal with signed integer values only.
- Without modification, these formats are not useful in scientific or business applications that deal with real number values.
- Floating-point representation solves this problem.

Floating-Point Representation

- If we are clever programmers, we can perform floating-point calculations using any integer format.
- This is called *floating-point emulation*, because floating point values aren't stored as such; we just create programs that make it seem as if floating-point values are being used.
- Most of today's computers are equipped with specialized hardware that performs floating-point arithmetic with no special programming required.
  - Not embedded processors!

Floating-Point Representation

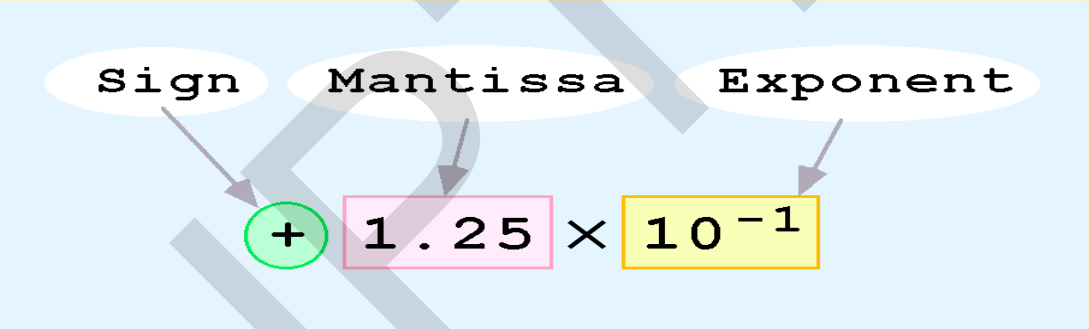
- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
  - For example: $0.5 \times 0.25 = 0.125$
- They are often expressed in scientific notation.
  - For example:

    $0.125 = 1.25 \times 10^{-1}$

    $5{,}000{,}000 = 5.0 \times 10^{6}$

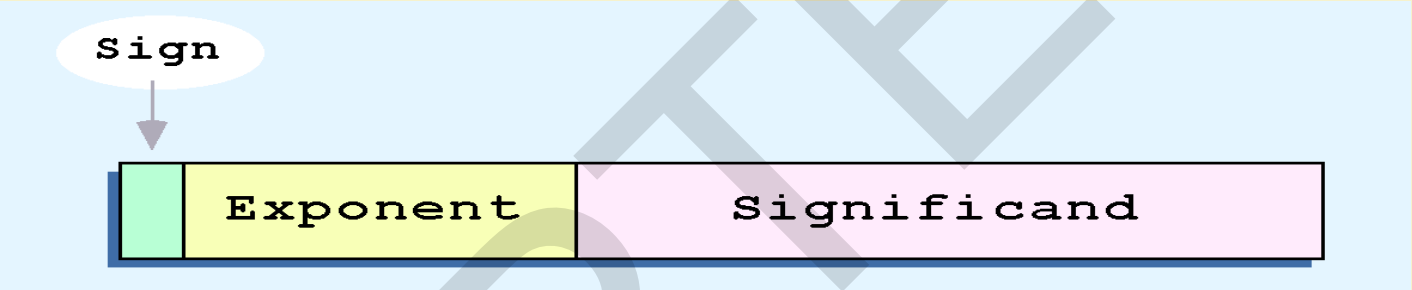
Floating-Point Representation

- Computers use a form of scientific notation for floating-point representation
- Numbers written in scientific notation have three components:



Floating-Point Representation

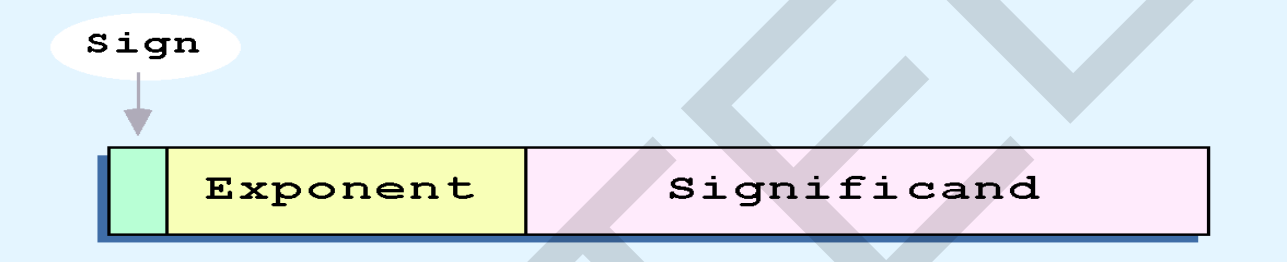
- Computer representation of a floating-point number consists of three fixed-size fields:



- This is the standard arrangement of these fields.

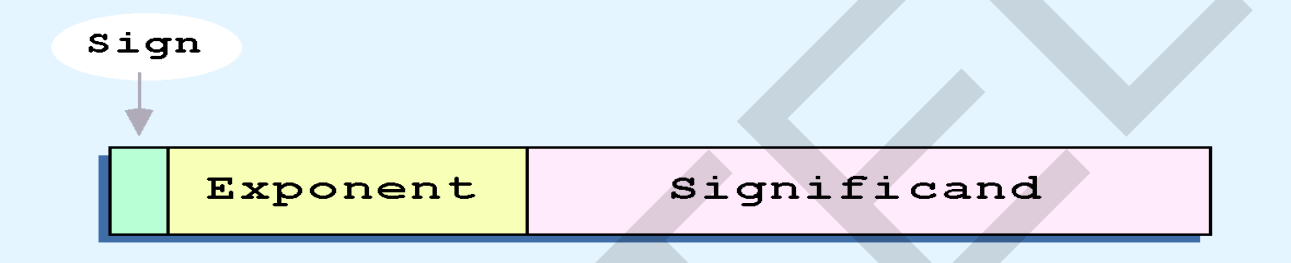
Note: Although "significand" and "mantissa" do not technically mean the same thing, many people use these terms interchangeably. We use the term "significand" to refer to the fractional part of a floating point number.

Floating-Point Representation



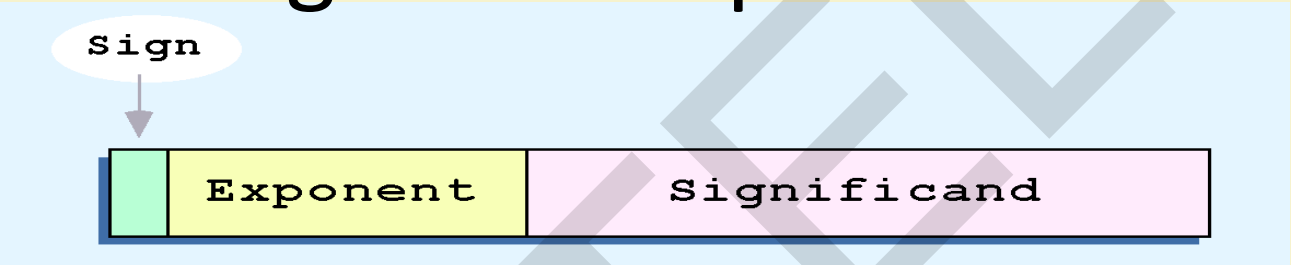
- The one-bit sign field is the sign of the stored value.
- The size of the exponent field determines the range of values that can be represented.
- The size of the significand determines the precision of the representation.

Floating-Point Representation



- We introduce a hypothetical “Simple Model” to explain the concepts
- In this model:
  - A floating-point number is 14 bits in length
  - The exponent field is 5 bits
  - The significand field is 8 bits

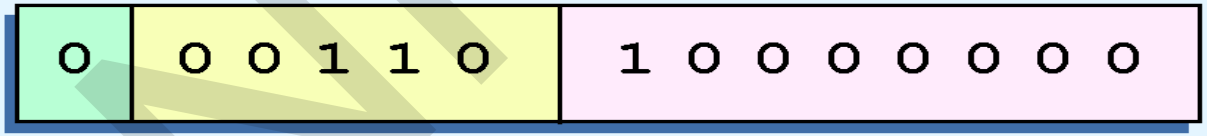
Floating-Point Representation



- The significand is always preceded by an implied binary point.
- Thus, the significand always contains a fractional binary value.
- The exponent indicates the power of 2 by which the significand is multiplied.

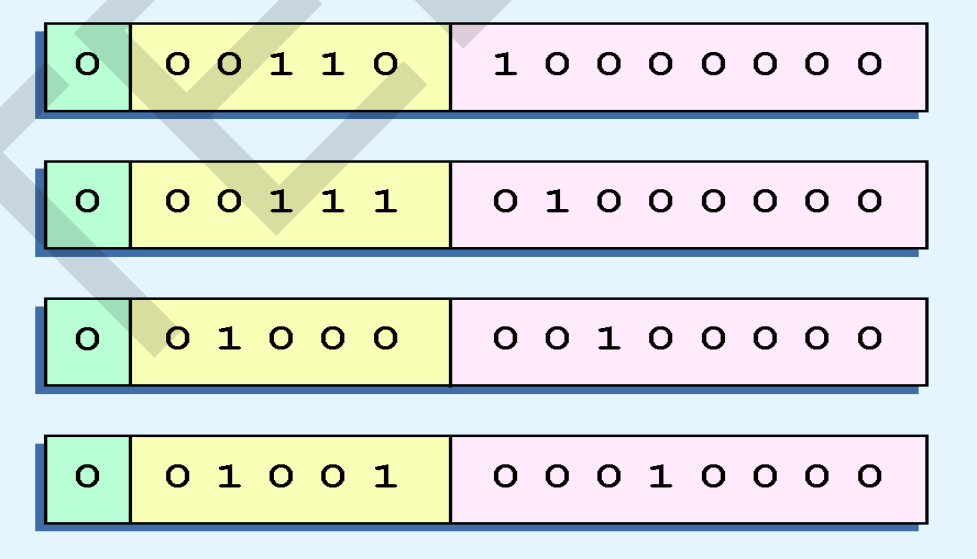
Floating-Point Representation

- Example:
  - Express 32_{10} in the simplified 14-bit floating-point model.
- We know that 32 is 2^5. So in (binary) scientific notation $32 = 1.0 \times 2^5 = 0.1 \times 2^6$.
  - In a moment, we'll explain why we prefer the second notation versus the first.
- Using this information, we put 110 (= 6_{10}) in the exponent field and 1 in the significand as shown.

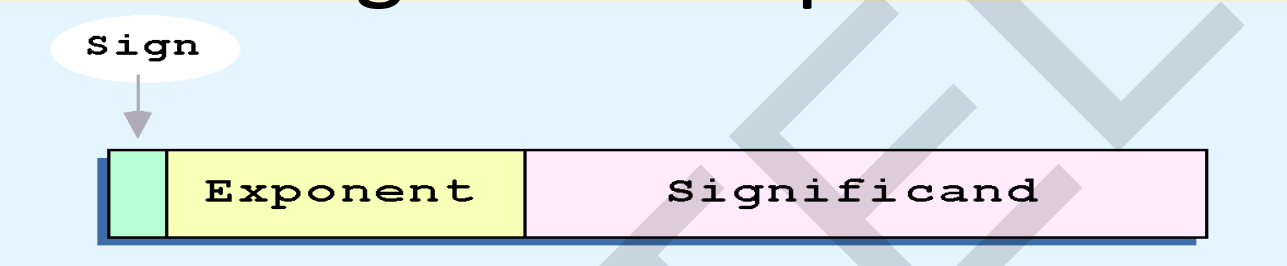


Floating-Point Representation

- The illustrations shown at the right are *all* equivalent representations for 32 using our simplified model.
- Not only do these synonymous representations waste space, but they can also cause confusion.



Floating-Point Representation



- Another problem with our system is that we have made no allowances for negative exponents. We have no way to express 0.5 ($=2^{-1}$)! (Notice that there is no sign in the exponent field.)

All of these problems can be fixed with no changes to our basic model.

Floating-Point Representation

- To resolve the problem of synonymous forms, we establish a rule that the first digit of the significand must be 1, with no ones to the left of the radix point.
- This process, called *normalization*, results in a unique pattern for each floating-point number.
  - In our simple model, all significands must have the form 0.1xxxxxxxx
  - For example, $4.5 = 100.1 \times 2^0 = 1.001 \times 2^2 = 0.1001 \times 2^3$. The last expression is correctly normalized.

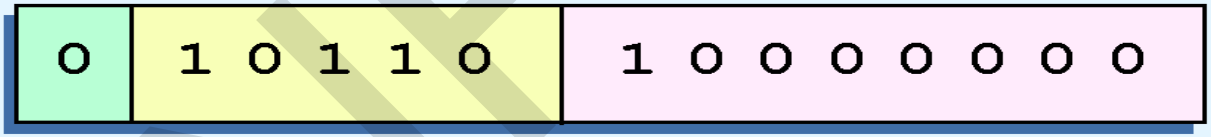
In our simple instructional model, we use no implied bits.

Floating-Point Representation

- To provide for negative exponents, we will use a *biased exponent*.
- A bias is a number that is approximately midway in the range of values expressible by the exponent. We subtract the bias from the value in the exponent to determine its true value.
  - In our case, we have a 5-bit exponent. We will use 16 for our bias. This is called *excess-16* representation.
- In our model, exponent values less than 16 are negative, representing fractional numbers.

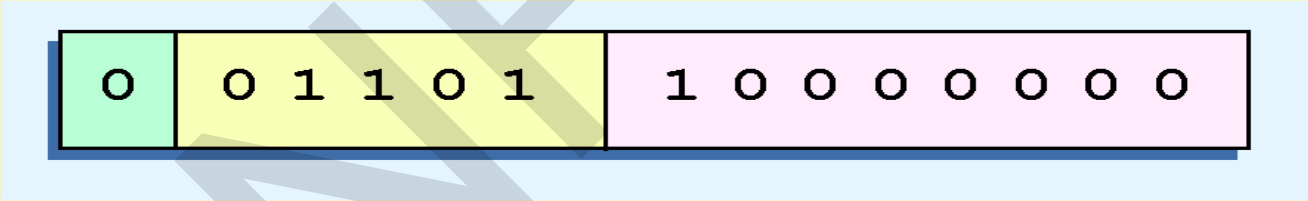
Example 1

- Example:
  - Express 32_{10} in the revised 14-bit floating-point model.
- We know that $32 = 1.0 \times 2^5 = 0.1 \times 2^6$.
- To use our excess 16 biased exponent, we add 16 to 6, giving 22_{10} (=10110_2).
- So we have:



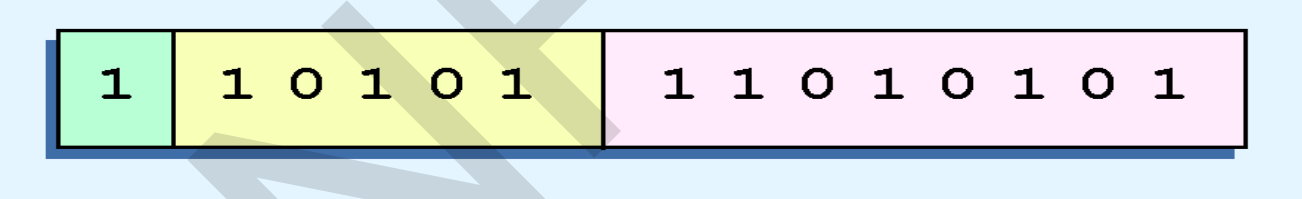
Example 2

- Example:
  - Express 0.0625_{10} in the revised 14-bit floating-point model.
- We know that 0.0625 is 2^{-4}. So in (binary) scientific notation $0.0625 = 1.0 \times 2^{-4} = 0.1 \times 2^{-3}$.
- To use our excess 16 biased exponent, we add 16 to -3, giving 13_{10} ($=01101_2$).



Example 3

- Example:
  - Express -26.625_{10} in the revised 14-bit floating-point model.
- We find $26.625_{10} = 11010.101_2$. Normalizing, we have: $26.625_{10} = 0.11010101 \times 2^5$.
- To use our excess 16 biased exponent, we add 16 to 5, giving 21_{10} ($=10101_2$). We also need a 1 in the sign bit.



Floating-Point Standards

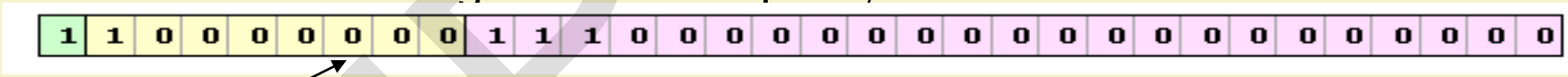
- The IEEE has established a standard for floating-point numbers
- The IEEE-754 *single precision* floating point standard uses an 8-bit exponent (with a bias of 127) and a 23-bit significand.
- The IEEE-754 *double precision* standard uses an 11-bit exponent (with a bias of 1023) and a 52-bit significand.

Floating-Point Representation

- In both the IEEE single-precision and double-precision floating-point standard, the significant has an implied 1 to the LEFT of the radix point.
  - The format for a significand using the IEEE format is: 1.xxx…
  - For example, $4.5 = .1001 \times 2^3$ in IEEE format is $4.5 = 1.001 \times 2^2$. The 1 is implied, which means it does not need to be listed in the significand (the significand would include only 001).

Floating-Point Representation

- Example: Express -3.75 as a floating point number using IEEE single precision.
- First, let's normalize according to IEEE rules:
  - $-3.75 = -11.11_2 = -1.111 \times 2^1$
  - The bias is 127, so we add 127 + 1 = 128 (this is our exponent)
  - The first 1 in the significand is implied, so we have:



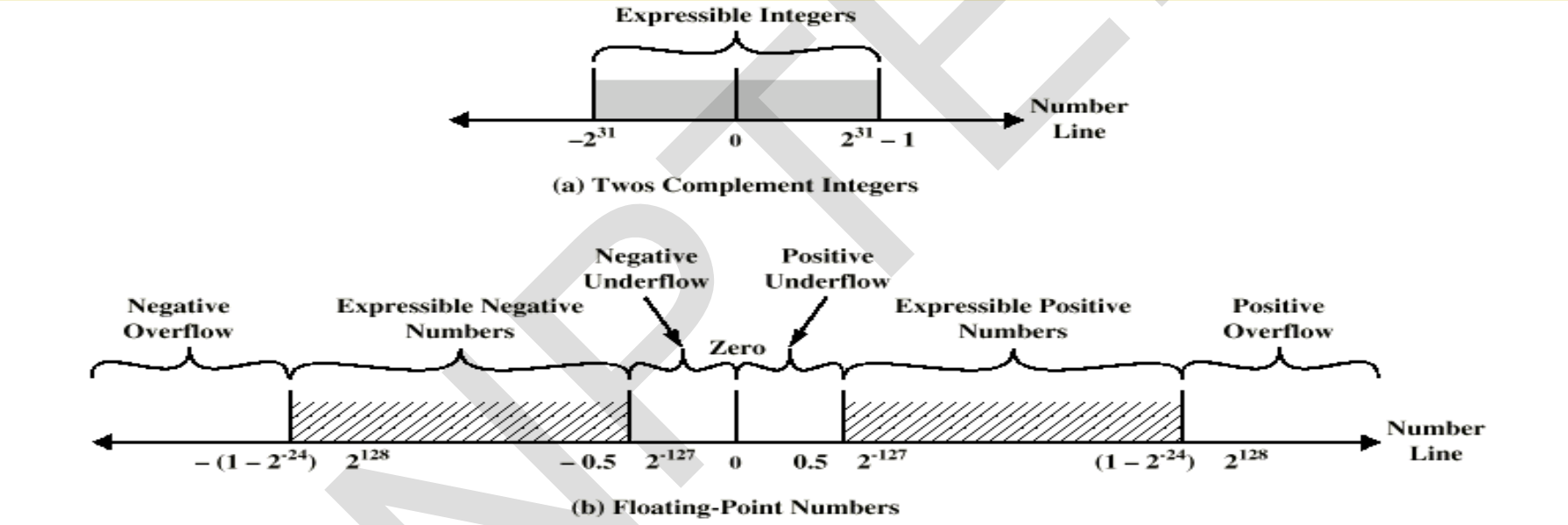
Since we have an implied 1 in the significand, this equates to

$-(1).111_2 \times 2^{(128-127)} = -1.111_2 \times 2^1 = -11.11_2 = -3.75.$

FP Ranges

- For a 32 bit number
  - 8 bit exponent
  - +/- $2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of significand
  - 23 bit significand $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

Expressible Numbers



Floating-Point Representation

- Using the IEEE-754 single precision floating point standard:
  - An exponent of 255 indicates a special value.
    - If the significand is zero, the value is $\pm$ infinity.
    - If the significand is nonzero, the value is NaN, “not a number,” often used to flag an error condition.
- Using the double precision standard:
  - The “special” exponent value for a double precision number is 2047, instead of the 255 used by the single precision standard.

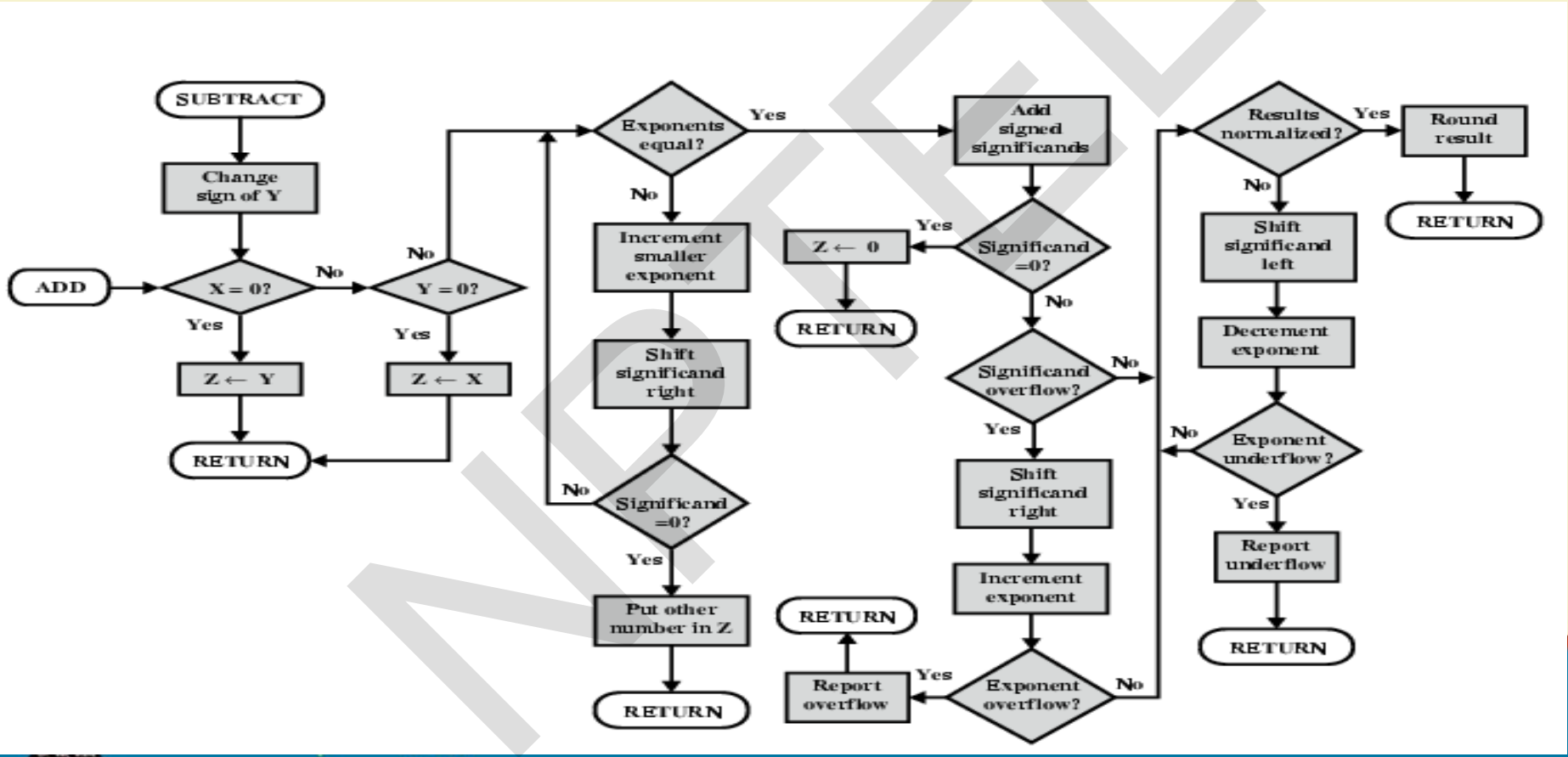
Floating-Point Representation

- Both the 14-bit model that we have presented and the IEEE-754 floating point standard allow two representations for zero.
  - Zero is indicated by all zeros in the exponent and the significand, but the sign bit can be either 0 or 1.
- This is why programmers should avoid testing a floating-point value for equality to zero.
  - Negative zero does not equal positive zero.

Floating-Point Representation

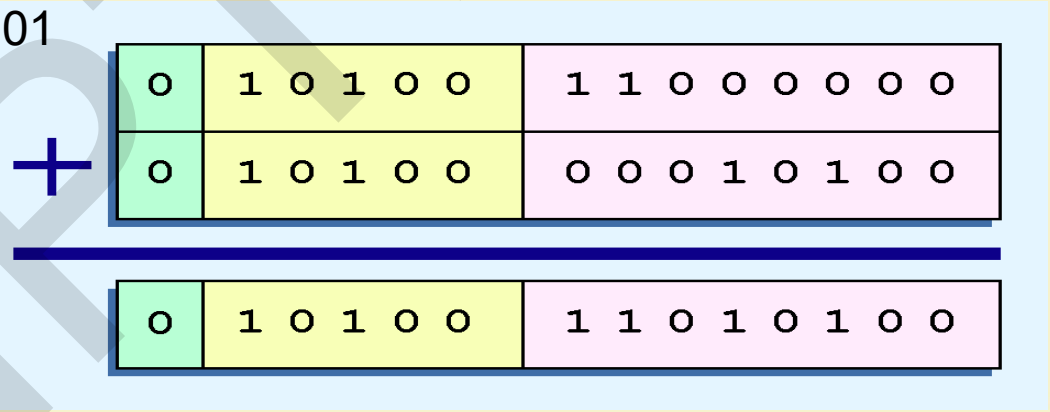
- Floating-point addition and subtraction are done using methods analogous to how we perform calculations using pencil and paper.
- The first thing that we do is express both operands in the same exponential power, then add the numbers, preserving the exponent in the sum.
- If the exponent requires adjustment, we do so at the end of the calculation.

FP Addition & Subtraction Flowchart



Floating-Point addition example

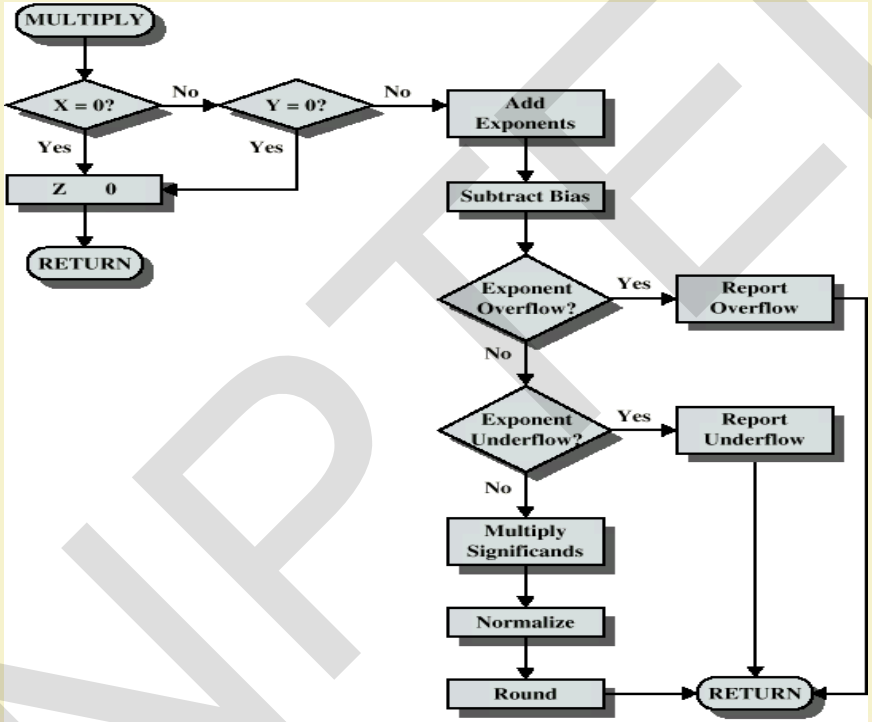
- Example:
  - Find the sum of 12_{10} and 1.25_{10} using the 14-bit "simple" floating-point model.
- We find $12_{10} = 0.1100 \times 2^4$. And $1.25_{10} = 0.101 \times 2^1 = 0.000101 \times 2^4$.
- Thus, our sum is 0.110101×2^4.



Floating-Point Multiplication

- Floating-point multiplication is also carried out in a manner akin to how we perform multiplication using pencil and paper.
- We multiply the two operands and add their exponents.
- If the exponent requires adjustment, we do so at the end of the calculation.

Floating Point Multiplication flowchart



Rounding and Errors

- No matter how many bits we use in a floating-point representation, our model must be finite.
- The real number system is, of course, infinite, so our models can give nothing more than an approximation of a real value.
- At some point, every model breaks down, introducing errors into our calculations.
- By using a greater number of bits in our model, we can reduce these errors, but we can never totally eliminate them.

Rounding and Errors

- Our job becomes one of reducing error, or at least being aware of the possible magnitude of error in our calculations.
- We must also be aware that errors can compound through repetitive arithmetic operations.
- For example, our 14-bit model cannot exactly represent the decimal value 128.5. In binary, it is 9 bits wide:

$$10000000.1_2 = 128.5_{10}$$

Rounding and Errors

- When we try to express 128.5_{10} in our 14-bit model, we lose the low-order bit, giving a relative error of:

$$\frac{128.5 - 128}{128.5} \approx 0.39\%$$

- If we had a procedure that repetitively added 0.5 to 128.5, we would have an error of nearly 2% after only four iterations.

Rounding and Errors

- Floating-point errors can be reduced when we use operands that are similar in magnitude.
- If we were repetitively adding 0.5 to 128.5, it would have been better to iteratively add 0.5 to itself and then add 128.5 to this sum.
- In this example, the error was caused by loss of the low-order bit.
- Loss of the high-order bit is more problematic.

Rounding and Errors

- Floating-point overflow and underflow can cause programs to crash.
- Overflow occurs when there is no room to store the high-order bits resulting from a calculation.
- Underflow occurs when a value is too small to store, possibly resulting in division by zero.

Experienced programmers know that it's better for a program to crash than to have it produce incorrect, but plausible, results.

EVEN BETTER, CAUGHT THE ERROR and DEAL WITH IT!

Rounding and Errors

- When discussing floating-point numbers, it is important to understand the terms *range, precision,* and *accuracy*.
- The range of a numeric integer format is the difference between the largest and smallest values that can be expressed.
- Accuracy refers to how closely a numeric representation approximates a true value.
- The precision of a number indicates how much information we have about a value

Rounding and Errors

- Most of the time, greater precision leads to better accuracy, but this is not always true.
  - For example, 3.1333 is a value of pi that is accurate to two digits, but has 5 digits of precision.
- There are other problems with floating point numbers.
- Because of truncated bits, you cannot always assume that a particular floating point operation is commutative or distributive.

Rounding and Errors

- This means that we cannot assume:

  (a + b) + c = a + (b + c) or

  a*(b + c) = ab + ac

- Moreover, to test a floating point value for equality to some other number, it is best to declare a "nearness to x" epsilon value. For example, instead of checking to see if floating point x is equal to 2 as follows:

  if x == 2 then …

  it is better to use:

  if (abs(x - 2) < epsilon) then ...

  (assuming we have epsilon defined correctly!)

Some Exercises

- If (75)x = (45)y, then possible values of x and y are
- Possible values of *x* and *y* satisfying $(2x)_5 = (3y)_6$ are
- If $x_6 = (32)_9 * (24)_5$ then x is equal to
- The number 267_8 in base-11 number system is

Boolean Algebra

Santanu Chattopadhyay
Electronics & Electrical Communication Engineering

Introduction

- Developed by English Mathematician *George Boole* in between 1815 - 1864.
- It is described as *an algebra of logic or an algebra of two values i.e True or False.*
- The term *logic* means a statement having binary decisions i.e *True/Yes or False/No*.

Application of Boolean algebra

- It is used to perform the logical operations in digital computer.
- In digital computer True represent by '1' (high voltage) and False represent by '0' (low voltage)
- Logical operations are performed by logical operators. The fundamental logical operators are:

1. AND (conjunction)
2. OR (disjunction)
3. NOT (negation/complement)

AND operator

- It performs logical multiplication and denoted by (.) dot.

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

OR operator

- It performs logical addition and denoted by (+) plus.

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT operator

- It performs logical negation and denoted by (-) bar. It operates on single variable.

X	$\overline{X}$	(means complement of x)
0	1	
1	0	

Truth Table

- Truth table is a table that contains all possible values of logical variables/statements in a Boolean expression.

No. of possible combinations = 2^n, where n=number of variables used in a Boolean expression.

Truth Table

- The truth table for XY + Z is as follows:

Dec	X	Y	Z	X.Y	X.Y+Z
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	0
3	0	1	1	0	1
4	1	0	0	0	0
5	1	0	1	0	1
6	1	1	0	1	1
7	1	1	1	1	1

Tautology & Fallacy

- If the output of Booean expression is always True or 1 is called Tautology.
- If the output of Boolean expression is always False or 0 is called Fallacy.

P	P’	output (P + P’)	output (P . P’)
0	1	1	0
1	0	1	0

P+P’ is Tautology and P.P’ is Fallacy

Exercise

1. Evaluate the following Boolean expression using Truth Table.

(a) X'Y'+X'Y

(b) X'YZ'+XY'

(c) XY'(Z+YZ')+Z'

2. Verify that P+(PQ)' is a Tautology.

3. Verify that (X+Y)'=X'Y'

Implementation

- Boolean Algebra applied in computers electronic circuits. These circuits perform Boolean operations and these are called logic circuits or logic gates.

Logic Gate

- A gate is an digital circuit which operates on one or more signals and produce single output.
- Gates are digital circuits because the input and output signals are denoted by either 1(high voltage) or 0(low voltage).
- Three type of gates are as under:

1. AND gate
2. OR gate
3. NOT gate

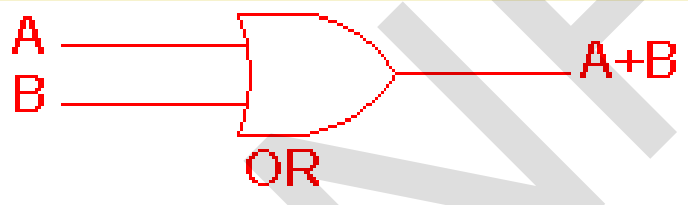
AND gate

- The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high.
- AND gate takes two or more input signals and produce only one output signal.

Input A	*Input B*	*Output AB*
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

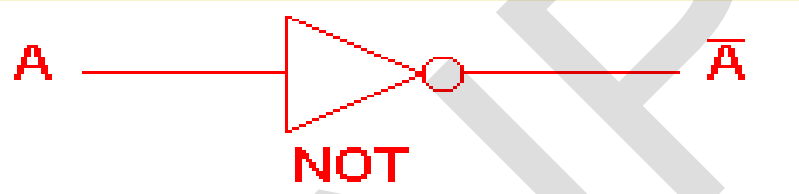
- The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high.
- OR gate also takes two or more input signals and produce only one output signal.



Input A	*Input B*	*Output A+B*
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate

- The NOT gate is an electronic circuit that gives a high output (1) if its input is low .
- NOT gate takes only one input signal and produce only one output signal.
- The output of NOT gate is complement of its input.
- It is also called inverter.



Input A	*Output* $\overline{A}$
0	1
1	0

Principal of Duality

- In Boolean algebras the duality Principle is obtained by interchanging AND and OR operators and replacing 0's by 1's and 1's by 0's. Compare the identities on the left side with the identities on the right.

Example

A+1 = 1 then A.0 = 0

Basic Theorem of Boolean Algebra

T1 : Properties of 0

(a) $0 + A = A$

(b) $0\,A = 0$

T2 : Properties of 1

(a) $1 + A = 1$

(b) $1\,A = A$

Basic Theorem of Boolean Algebra

T3 : Commutative Law

(a) $A + B = B + A$

(b) $A\,B = B\,A$

T4 : Associate Law

(a) $(A + B) + C = A + (B + C)$

(b) $(A\,B)\,C = A\,(B\,C)$

T5 : Distributive Law

(a) $A\,(B + C) = A\,B + A\,C$

(b) $A + (B\,C) = (A + B)\,(A + C)$

(c) $A+A'B = A+B$

Basic Theorem of Boolean Algebra

T6 : Indempotence (Identity) Law

(a) $A + A = A$

(b) $A\,A = A$

T7 : Absorption (Redundance) Law

(a) $A + A\,B = A$

(b) $A\,(A + B) = A$

Basic Theorem of Boolean Algebra

T8 : Complementary Law

(a) X+X'=1

(b) X.X'=0

T9 : Involution

(a) x'' = x

T10 : De Morgan's Theorem

(a) (X+Y)'=X'.Y'

(b) (X.Y)'=X'+Y'

Exercise

Q 1. State & Verify De Morgan's Law by using truth table and algebraically.

Q 2. State and verify distributive law.

Q 3. Draw a logic diagram for the following expression:

(a) ab+b’c+c’a’

(b) (a+b).(a+b’).c

Representation of Boolean expression

Boolean expression can be represented by either

(i) Sum of Product(SOP) form or

(ii) Product of Sum (POS form)

e.g.

AB+AC → SOP
(A+B)(A+C) → POS

In above examples both are in SOP and POS respectively but they are not in Standard SOP and POS.

Canonical form of Boolean Expression (Standard form)

- In standard SOP and POS each term of Boolean expression must contain all the literals (with and without bar) that has been used in Boolean expression.
- If the above condition is satisfied by the Boolean expression, that expression is called Canonical form of Boolean expression.

Canonical form of Boolean Expression (Standard form)

Contd..

- In Boolean expression **AB+AC** the literal C is mission in the 1st term **AB** and B is mission in 2nd term **AC**. That is why AB+AC is not a Canonical SOP.

Canonical form of Boolean Expression (Standard form)

Contd..

Convert AB+AC in Canonical SOP (Standard SOP)

Sol. AB + AC

AB(C+C') + AC(B+B')

ABC+ABC'+ABC+AB'C

ABC+ABC'+AB'C

Distributive law

Canonical form of Boolean Expression (Standard form)

Contd..

Convert (A+B)(A+C) in Canonical SOP (Standard SOP)

Sol. (A+B).(A+C)

((A+B)+(C.C')) .((A+C)+(B.B'))

(A+B+C).(A+B+C').(A+B+C)(A+B'+C)

(A+B+C).(A+B+C')(A+B'+C)

Distributive law

Remove duplicates

Conversion between SOP and POS

- Multiplying out
  - POS → SOP

$F = (A + B')(A + C)(A + D)$

$A + B'$

$\underline{A + C}$

$AA + AB'$

$\underline{\quad AC + B'C}$

$A + AB' + AC + B'C$

$A + B'C$

$\underline{A + D}$

$AA + AB'C + AD + B'CD$

$F = A + B'CD$

- Factoring
  - SOP → POS

$F = A + B'CD$

$F = (A + B')(A + CD)$

$F = (A + B')(A + C)(A + D)$

Canonical form of Boolean Expression (Standard form)

Contd..

Minterm and Maxterm

Individual term of Canonical Sum of Products (SOP) is called Minterm. In otherwords minterm is a product of all the literals (with or without bar) within the Boolean expression.

Individual term of Canonical Products of Sum (POS) is called Maxterm. In otherwords maxterm is a sum of all the literals (with or without bar) within the Boolean expression.

Minterms & Maxterms for 2 variables (Derivation of Boolean function from Truth Table)

x	y	Index	Minterm	Maxterm
0	0	0	$m_0 = x' y'$	$M_0 = x + y$
0	1	1	$m_1 = x' y$	$M_1 = x + y'$
1	0	2	$m_2 = x y'$	$M_2 = x' + y$
1	1	3	$m_3 = x y$	$M_3 = x' + y'$

The minterm m_i should evaluate to 1 for each combination of x and y.

The maxterm is the complement of the minterm

Solved Problem

Prob. Find the minterm designation of XY'Z'

Sol. Subsitute 1's for non barred and 0's for barred letters

Binary equivalent = 100

Decimal equivalent = 4

Thus XY'Z'=m_4

Purpose of the Index

- Minterms and Maxterms are designated with an index
- The index number corresponds to a binary pattern
- The index for the minterm or maxterm, expressed as a binary number, is used to determine whether the variable is shown in the true or complemented form
- For Minterms:
  - '1' means the variable is "Not Complemented" and
  - '0' means the variable is "Complemented".
- For Maxterms:
  - '0' means the variable is "Not Complemented" and
  - '1' means the variable is "Complemented".

Solved Problem

Write SOP form of a Boolean Function F, Which is represented by the following truth table.

Sum of minterms of entries that evaluate to '**1**'

x	*y*	*z*	*F*	Minterm
0	0	0	0	
0	0	1	**1**	$m_1 = x' y' z$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	**1**	$m_6 = x y z'$
1	1	1	**1**	$m_7 = x y z$

Focus on the '**1**' entries

$$F = m_1 + m_6 + m_7 = \sum (1, 6, 7) = \bar{x}\,\bar{y}\,z + x\,y\,\bar{z} + x\,y\,z$$

Exercise

1. Write POS form of a Boolean Function F, Which is represented by the following truth table

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

2. Write equivalent canonical Sum of Product expression for the following Product of Sum Expression: F(X,Y,Z)=Π(1,3,6,7)

Minimization of Boolean Expression

- Canonical SOP (Sum of Minterms) and POS (Product of Maxterm) is the derivation/expansion of Boolean Expression.

- Canonical forms are not usually minimal.

- Minimization of Boolean expression is needed to simplify the Boolean expression and thus reduce the circuitry complexity as it uses less number of gates to produce same output that can by taken by long canonical expression.