Chapter Three

Data Representation and Computer Arithmetic

JAFAR MUZEYIN

Topics Covered

- Number Systems and Conversion
- Units of Data Representation
- Coding Methods
- Binary Arithmetic
- Complements
- Fixed and Floating points representation
- (Boolean Algebra) and Logic Circuits

Number systems and conversion

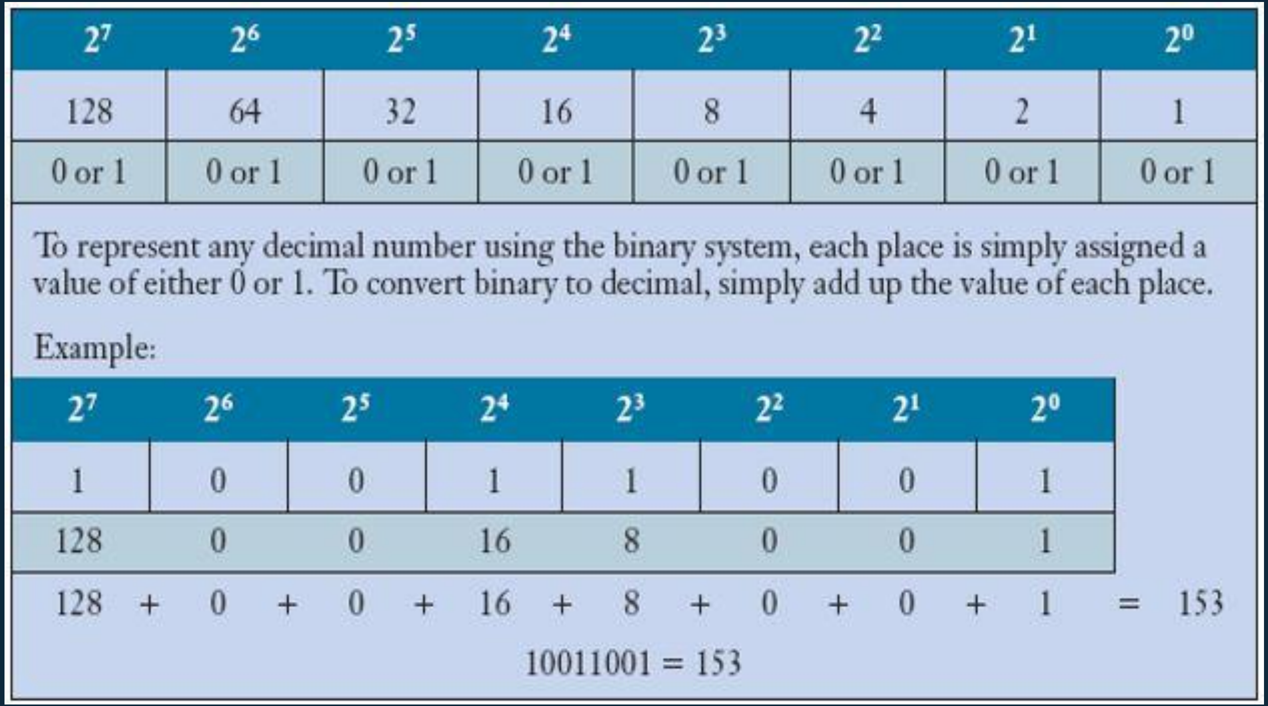
- Number Systems
  - Decimal
  - Binary
  - Octal
  - Hexadecimal
- Conversion

Decimal systems

- **The decimal system**
  - Base 10 with ten distinct digits (0, 1, 2, …, 9)
  - Any number greater than 9 is represented by a combination of these digits
  - The weight of a digit is based on power of 10
- **Example:**
  - The number 81924 is actually the sum of: $(8X10^4)+(1X10^3)+(9X10^2)+(2X10^1)+(4X10^0)$

Binary systems

- Computers use the binary system to store and compute numbers.



2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1

To represent any decimal number using the binary system, each place is simply assigned a value of either 0 or 1. To convert binary to decimal, simply add up the value of each place.

Example:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	0	0	1
128	0	0	16	8	0	0	1

$128 + 0 + 0 + 16 + 8 + 0 + 0 + 1 = 153$

10011001 = 153

Binary systems

- **The binary system (0 & 1)**
  - The two digits representation is called a binary system
  - Two electrical states – **on (1) & off (0)**
  - The position weights are based on the **power of 2**
  - The various combination of the two digits representation gives us the final value

Examples :

I. 1011011 in binary = 91 in decimal

II. 1101.01 in binary = 13.25 in decimal

Binary Fractions

Binary fractions can also be represented:

Position Value:	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	etc.
Fractions:	1/2	1/4	1/8	1/16	1/32	
Decimal:	.5	.25	.125	.0625	.03125	

Binary Fractions

5th 4th 3rd 2nd 1st 0th

$1\ 0\ 1\ 1\ 1\ 1_2 = (1X2^5)+(0X2^4)+(1X2^3)+(1X2^2)+(1X2^1)+(1X2^0)$

$= 32+8+4+2+1$

$= 47_{10}$

$1011001_2 = 89_{10}$

Exercise :

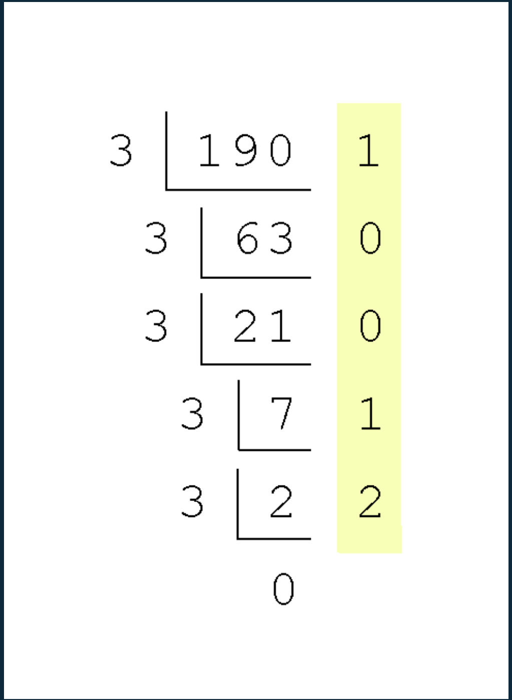
Convert the following binary numbers into their decimal equivalent

- $1110100_2 = (?)_{10}$
- $101101.1101_2 = (?)_{10}$

Converting Between Bases

- **Converting 190 to base 3...**
  - Continue in this way until the quotient is zero.
  - In the final calculation, we note that 3 divides 2 zero times with a remainder of 2.
  - Our result, reading from bottom to top is:

$$190_{10} = 21001_3$$



Conversion of Decimal to Binary

Divide by 2 (remainder division) till the dividend is zero and read remainders in reverse order.
The right column shows result of integer division

Read answer in this direction, write the answer left to right

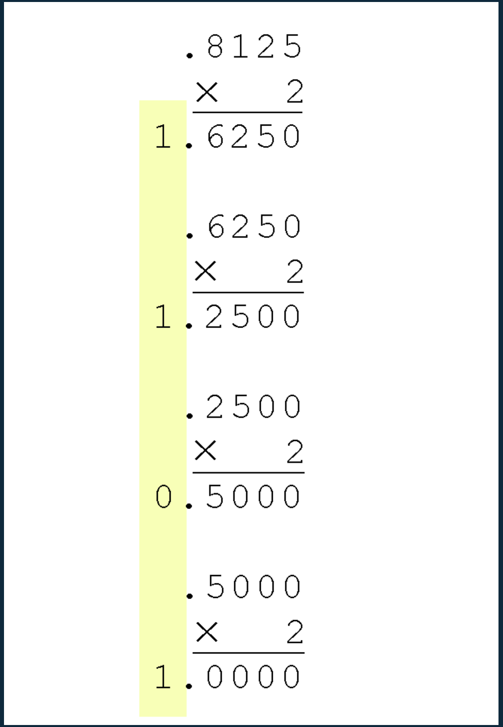
mod 2	637/2
1	318
0	159
1	79
1	39
1	19
1	9
1	4
0	2
0	1
1	0

$637_{10} = 1001111101_2$

❖ Convert 789_{10} to base 2

Converting Between Bases

- **Converting 0.8125 to binary . . .**
  - You are finished when the product is zero, or until you have reached the desired number of binary places.
  - Our result, reading from top to bottom is:
    $0.8125_{10} = 0.1101_2$
  - This method also works with any base. Just use the target radix as the multiplier.



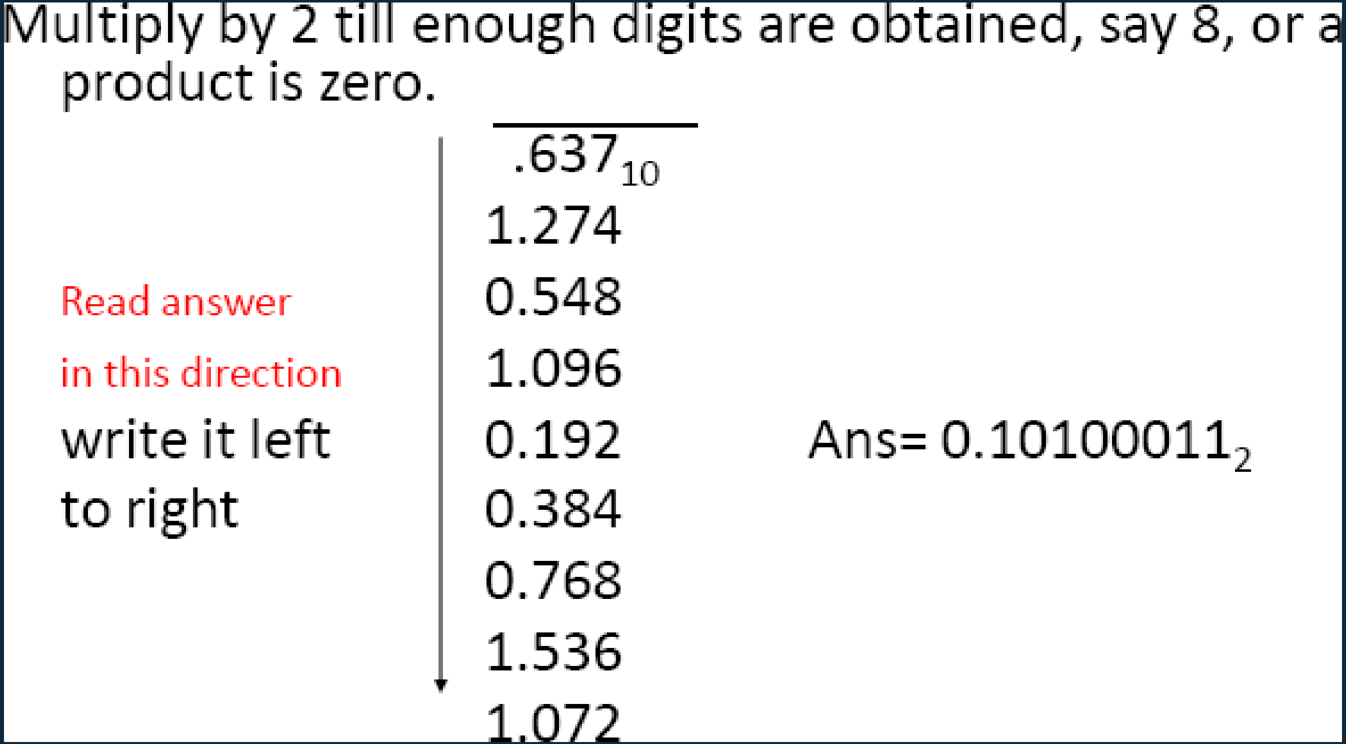
```
  .8125
  ×   2
 1.6250

  .6250
  ×   2
 1.2500

  .2500
  ×   2
 0.5000

  .5000
  ×   2
 1.0000
```

To Binary Fractions - Conversions



Multiply by 2 till enough digits are obtained, say 8, or a
product is zero.

$\overline{.637_{10}}$

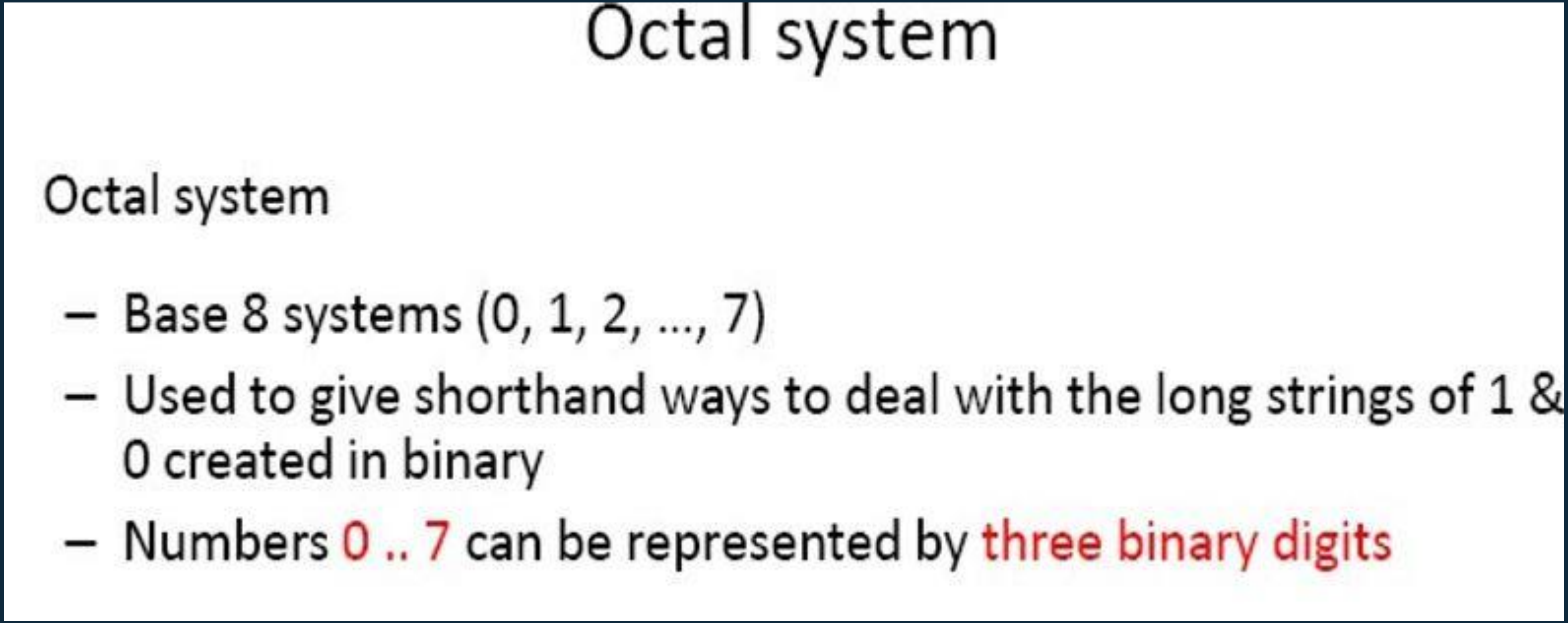
Read answer
in this direction
write it left
to right

$\downarrow$

1.274
0.548
1.096
0.192
0.384
0.768
1.536
1.072

Ans= 0.10100011_2

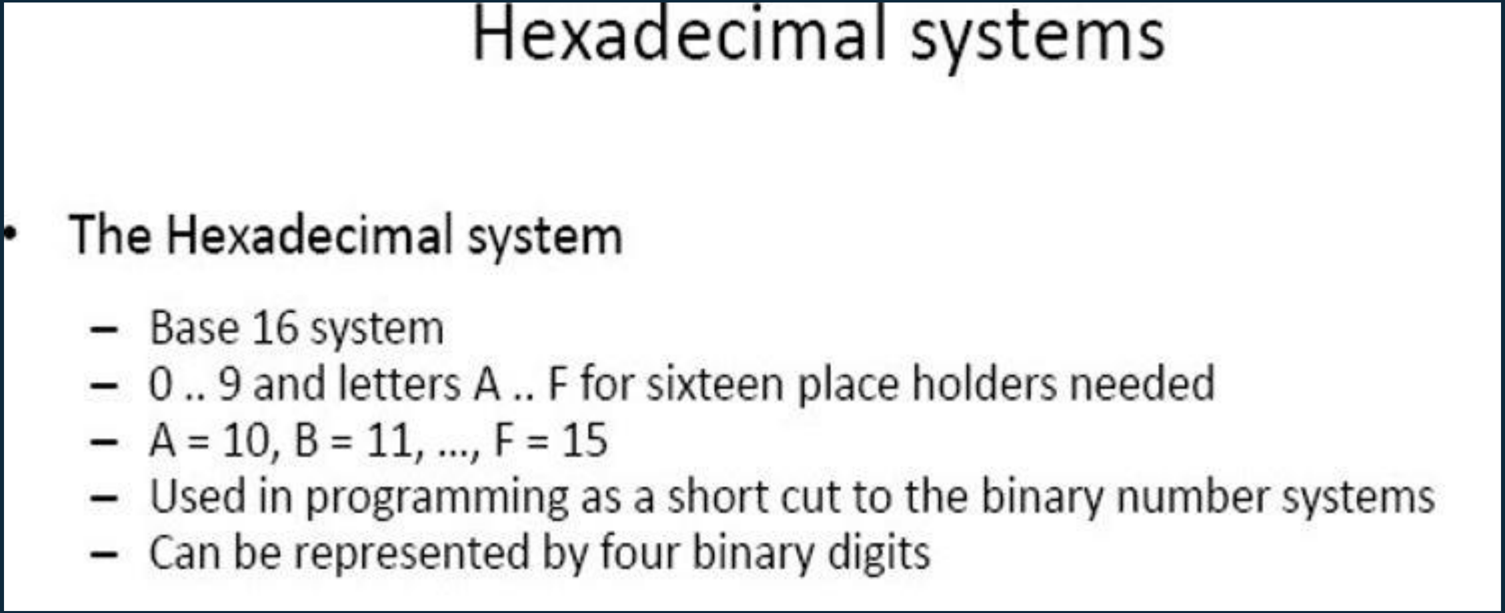
- Convert 0.325_{10} to base 2



Octal system

Octal system

- Base 8 systems (0, 1, 2, ..., 7)
- Used to give shorthand ways to deal with the long strings of 1 & 0 created in binary
- Numbers 0 .. 7 can be represented by three binary digits



Hexadecimal systems

- The Hexadecimal system
  - Base 16 system
  - 0 .. 9 and letters A .. F for sixteen place holders needed
  - A = 10, B = 11, ..., F = 15
  - Used in programming as a short cut to the binary number systems
  - Can be represented by four binary digits

Exercise – Convert …

Decimal	Binary	Octal	Hexa-decimal
29.8			
	101.1101		
		3.07	
			DF.E8

Exercise – Convert …

Answer

Decimal	Binary	Octal	Hexa-decimal
29.8	11101.110011…	35.63…	1D.CC…
5.8125	101.1101	5.64	5.D
3.109375	11.000111	3.07	3.1C
12.5078125	1100.10000010	14.404	C.82

Decimal → Octal Conversion

- A decimal number can be converted to an octal number by successively dividing the number by 8 as follows:

  266 ÷ 8 = 33 remainder 2 LSD (right-most digit)

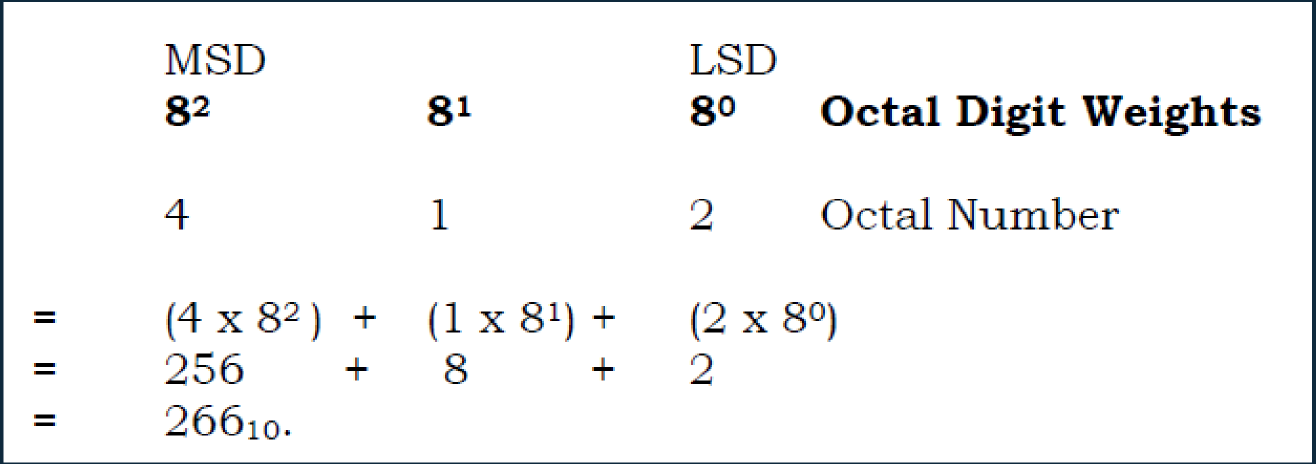
  33 ÷ 8 = 4 remainder 1

  4 ÷ 8 = 0 remainder 4 MSB (left-most digit).

- Therefore $266_{10} = 412_8$

Octal ➔ Decimal Conversion

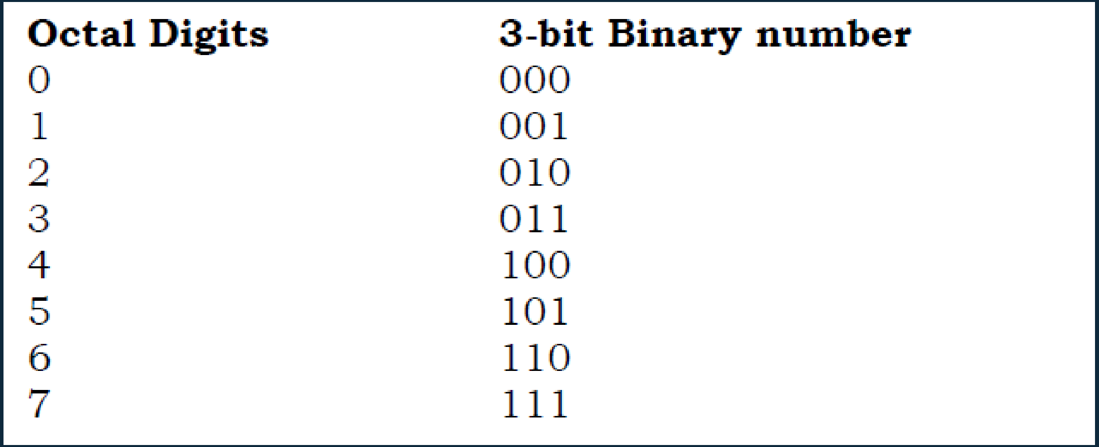
- To convert an octal number to a decimal number, multiply each octal value by the weight of the digit and sum the results. For example, $412_8 = 266_{10}$.



	MSD		LSD	
	8^2	**8^1**	**8^0**	**Octal Digit Weights**
	4	1	2	Octal Number
=	$(4 \text{ x } 8^2)$ +	$(1 \text{ x } 8^1)$ +	$(2 \text{ x } 8^0)$	
=	256 +	8 +	2	
=	266_{10}.			

Octal ➔ Binary Representation

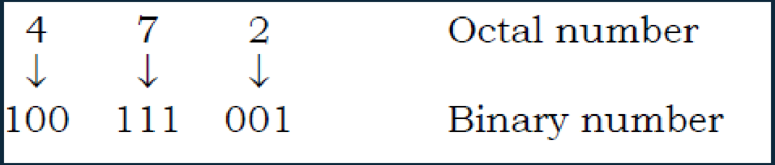
- Each octal digit can be represented by a 3-bit binary number as shown below:



Octal Digits	3-bit Binary number
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

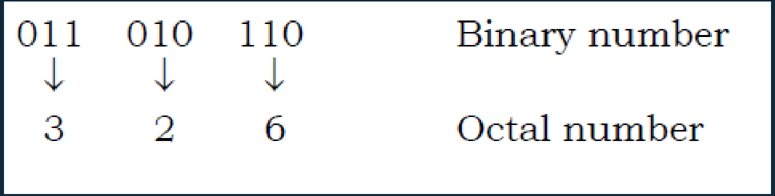
Octal ⟷ Binary Conversion

- Conversion from octal to binary is very straightforward. Each octal digit is replaced by 3-bit binary number. For example, 472_8 = 100 111 010_2.



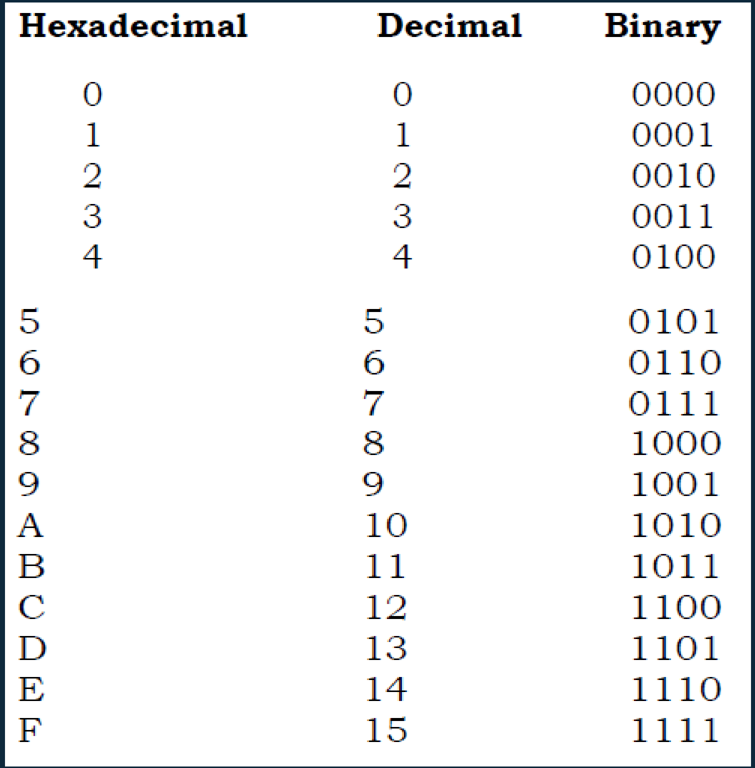
4	7	2	Octal number
↓	↓	↓	
100	111	001	Binary number

- A binary number is converted into an octal number by taking groups of 3 bits, starting from LSB, and replacing them with an octal digit. For example, 11 010 110_2 = 326_8.



011	010	110	Binary number
↓	↓	↓	
3	2	6	Octal number

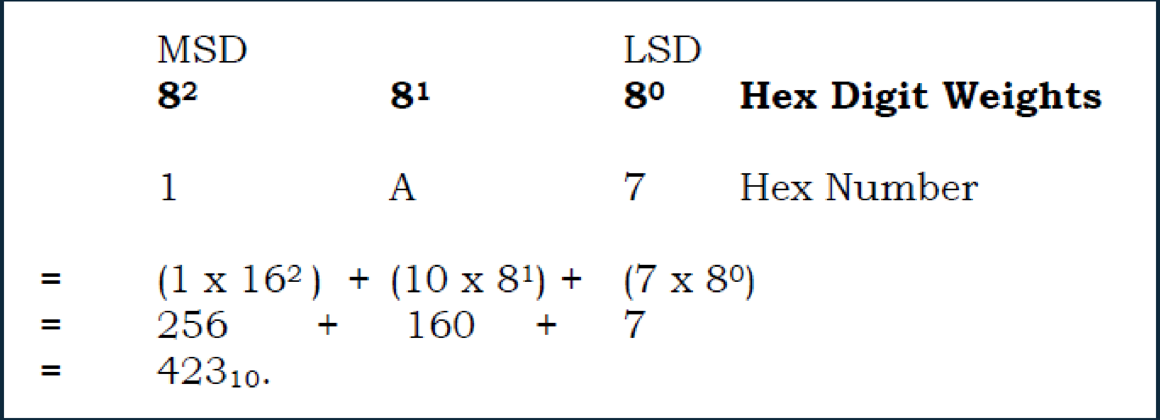
Hexadecimal Number



Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadecimal ➔ Decimal Conversion

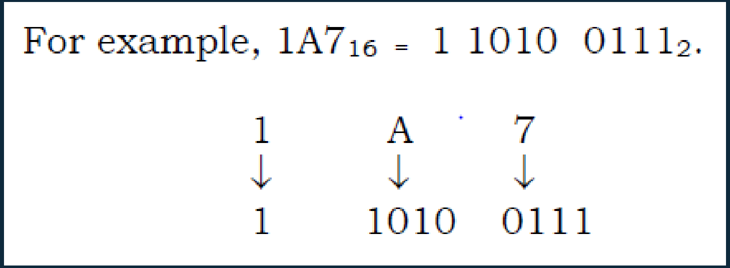
- To convert a hex number to a decimal number, multiply each hex value by the weight of the digit and sum the results. For example, $1A7_{16} = 423_{10}$.



	MSD		LSD	
	8^2	**8^1**	**8^0**	**Hex Digit Weights**
	1	A	7	Hex Number
=	(1×16^2) +	(10×8^1) +	(7×8^0)	
=	256 +	160 +	7	
=	423_{10}.			

Hexadecimal $\Longleftrightarrow$ Binary Conversion

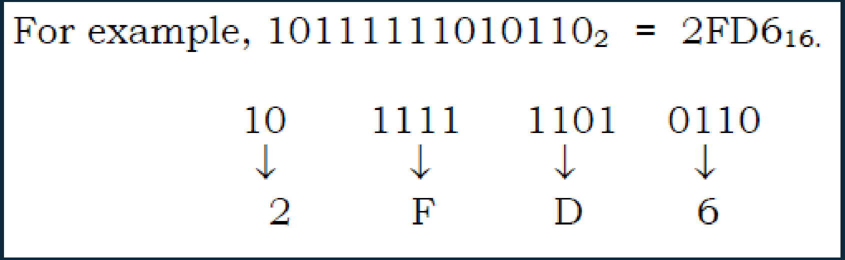
- Each hex digit can be represented by a 4-bit binary number as shown above. Conversion from hex to binary is very straightforward. Each hex digit is replaced by 4-bit binary number.



For example, $1A7_{16}$ = 1 1010 0111_2.

$$\begin{array}{ccc} 1 & A & 7 \\ \downarrow & \downarrow & \downarrow \\ 1 & 1010 & 0111 \end{array}$$

- A binary number is converted into an octal number by taking groups of 4 bits, starting from LSB, and replacing them with a hex digit. For example, $11010110_2 = 326_8$.



For example, 10111111010110_2 = $2FD6_{16}$.

$$\begin{array}{cccc} 10 & 1111 & 1101 & 0110 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & F & D & 6 \end{array}$$

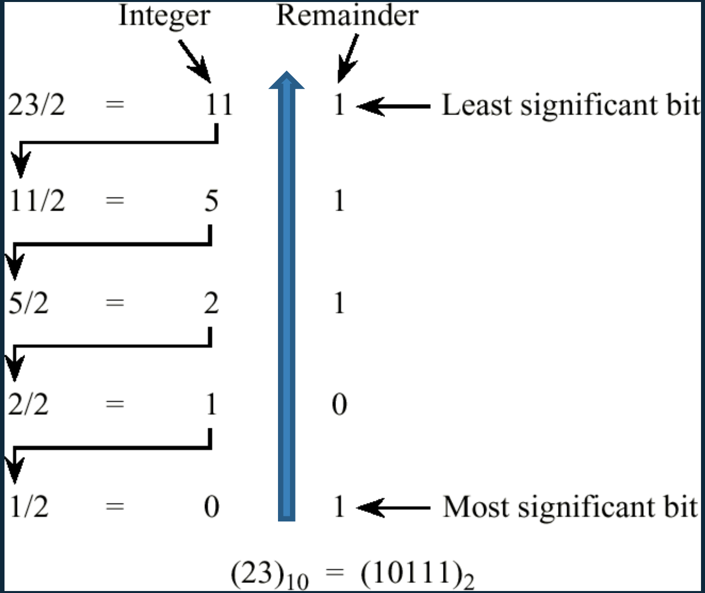
Base Conversion for Floating Points with the Remainder Method

Decimal ➔ Binary

Eg. Convert 23.375_{10} to base 2.

Technique:

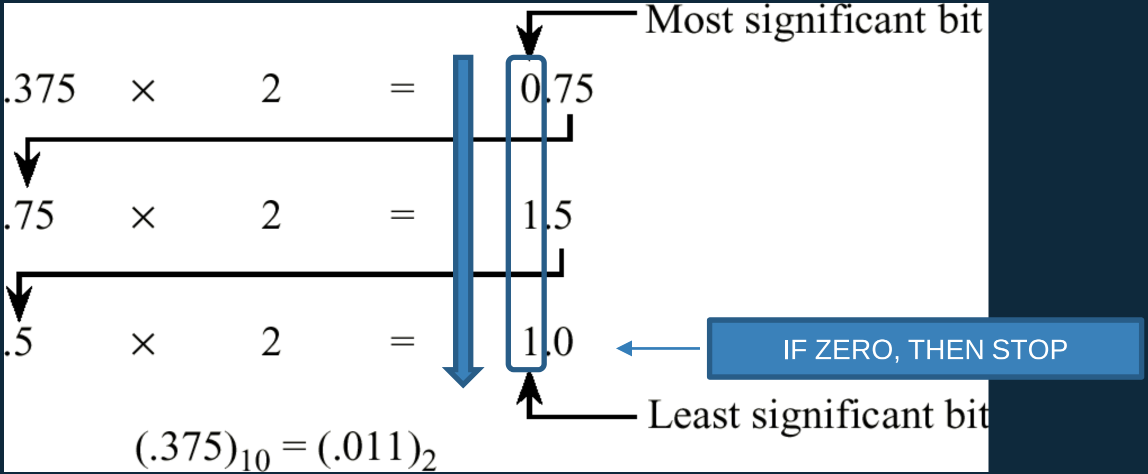
1. Start by converting the integer portion:



Floating Points Conversion using Remainder Method

Decimal → Binary

2. Then, convert the fraction by multiply it with the based we want to convert:



Base Conversion for Floating Points with the Remainder Method

Binary → Decimal

Eg. 1010.01_2 = ( ) $_{10}$

- Technique:
  - Multiply each binary number by 2^{-n}, where $-n$ is the weight of the bit for fraction starting from left to right. .
  - Then, sum the results.

$$1010.01_2$$

$$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \;.\; 0 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 10 + 0.25$$

$$= 10.25_{10}$$

Therefore, 1010.01_2 = **10.25_{10}**

Base Conversion for Floating Points with the Remainder Method

Octal – Decimal

Technique:

- Multiply each octal number by 8^{-n}, where *-n* is the weight of the bit for fraction starting from left to right. .
- Then, sum the results.

Eg. 46.3_8 = $\_\_\_\_\__{10}$

$$46.3_8 = 4 \times 8^1 + 6 \times 8^0 + 3 \times 8^{-1}$$
$$= 38 + 0.375$$
$$= 38.375_{10}$$

Therefore, 46.3_8 = **38.375_{10}**

Base Conversion for Floating Points with the Remainder Method

Hexadecimal - Decimal

Technique:

- Multiply each hexadecimal number by 16^{-n}, where *-n* is the weight of the bit for fraction starting from left to right.
- Then, sum the results.

Eg. $A7.0F_{16}$ = ____________ $_{10}$

$A7.0F_{16} = 10 \times 16^{1} + 7 \times 16^{0} + 0 \times 16^{-1} + 15 \times 16^{-2}$

$= 167 + 0.059$

$= 167.059_{10}$

Therefore, $A7.0F_{16}$ = **167.059_{10}**

Exercises Part 1

3. Convert hexadecimal $ABF2_{16}$ to
    a. decimal
    b. binary
    c. octal
    d

4. Convert1001010010110001 to
    a. decimal
    b. octal
    c. hexadecimal

5. Convert number octal 526_8 to
    a. decimal
    c. hexadecimal
    d. binary

Exercises Part 2

1) Convert the following number to the indicated base/code.

   a) 11101.11_2 to decimal.
   b) $FED.47_{16}$ to octal.
   c) 01101001_{BCD} to binary.
   d) 754_8 to BCD.
   e) 152.25_{10} to hexadecimal.

Storage Memory Bits

- How many bits does a computer use to store an integer?
  - Some models = 32 bits
  - Other models = 64 bits
- What if we try to compute or store a larger integer?
  - If we try to compute a value larger than the computer can store, we get an arithmetic overflow error.

Sign-and-magnitude

Sign-and-magnitude

- Also called, "sign-and-magnitude representation"
- A number consists of a magnitude and a symbol representing the sign

How Do we write Negative Binary Numbers

- Historically: 3 approaches
  - Sign-and-magnitude
  - Ones-complement
  - Twos-complement
- For all 3, the most-significant bit (MSB) is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative
- Twos-complement is the important one
  - Simplifies arithmetic
  - Used almost universally

Sign-and-magnitude

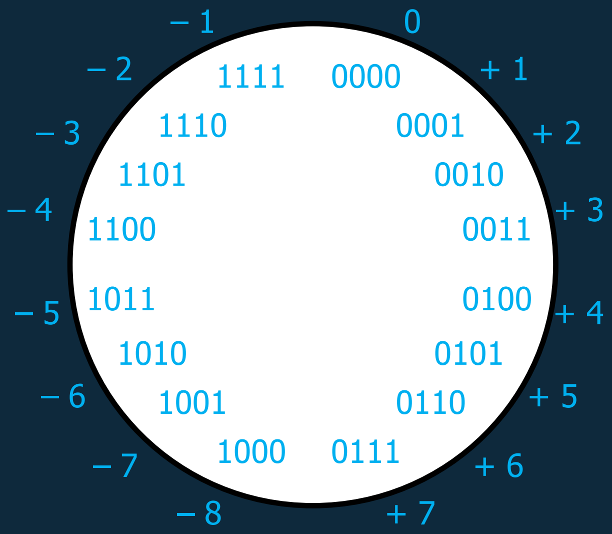
- The most-significant bit (MSB) is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative
- The remaining bits are the number's magnitude

Ones-complement

- Negative number: Bitwise complement positive number
  - $0011 \equiv 3_{10}$
  - $1100 \equiv -3_{10}$

Twos-complement

- Negative number: Bitwise complement plus one
  - $0011 \equiv 3_{10}$
  - $1101 \equiv -3_{10}$
- Number wheel
  - MSB is the sign digit
  - $0 \equiv$ positive
  - $1 \equiv$ negative



Twos-complement (con't)

- Complementing a complement to the original number
- Arithmetic is easy
  - Subtraction = negation and addition
    - Easy to implement in hardware

Representing Unsigned Integers

- How does a 16-bit computer represent the value 14?

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

- What is the largest 16-bit integer?

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$= 1x2^{15} + 1x2^{14} + \ldots + 1x2^{1} + 1x2^{0} = 65{,}535$$

If we just add 1 to this number, for example, overflow will occur. We will get a wrong result.

Representing Signed Integers

- How does a 16 bit computer represent the value -14?

1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Sign bit.

- What is the largest 16-bit signed integer?

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$= 1x2^{14} + 1x2^{13} + \ldots + 1x2^{1} + 1x2^{0} = 32{,}767$$

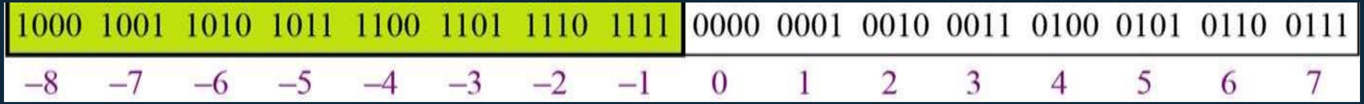
- Most computers use a different representation, called two’s complement.

How to Storing an integer in two's complement format?

- Convert the integer to an n-bit binary.
- If it is **positive** or **zero**, it is stored as it is. If it is
  - **negative**, take the two's complement and then store it.

- **Retrieving an integer in two's complement format**:

- If the **leftmost bit** is 1, the computer applies the two's complement operation to the n-bit binary. If the leftmost bit is 0, no operation is applied.
- The computer changes the binary to decimal (integer) and corresponding sign is added.



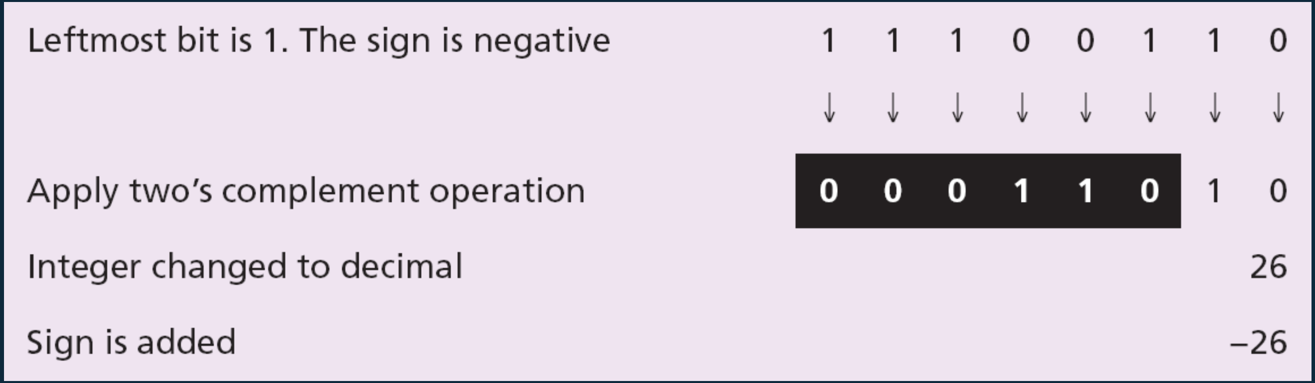
1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
–8	–7	–6	–5	–4	–3	–2	–1	0	1	2	3	4	5	6	7

Example:

Retrieve the integer that is stored as 11100110 in memory using two's complement format.

Solution:

The leftmost bit is 1, so the integer is negative. The integer needs to be two's complemented before changing to decimal.



Leftmost bit is 1. The sign is negative	1	1	1	0	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
Apply two’s complement operation	0	0	0	1	1	0	1	0
Integer changed to decimal								26
Sign is added								−26

Reading Assignment

How to store real numbers?

How to represent Floating-point?

IEEE Standard

Binary Codes

- Computers also use binary numbers (integers) to represent non-numeric information, such as text or graphics (images).
- Binary representations of text, (letters, textual numbers, textual symbols, etc.) are called codes.
- In a binary code, the binary number is a symbol and does not represent an actual number.
- A code normally cannot be "operated on" in the usual fashion – mathematical, logical, etc. That is, one can not usually add up, for example, two binary codes. It would be like attempting to add text and graphics!

Character representation- ASCII

- ASCII (American Standard Code for Information
  - Interchange) - **Binary Codes**
- It is the scheme used to represent characters.
- Each character is represented using 7-bit binary code.
- If 8-bits are used, the first bit is always set to 0

Numeric and Alphabetic Codes

- **ASCII code**
  - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
  - an **alphanumeric code**
  - each character represented by a 7-bit code
    - gives 128 possible characters
    - codes defined for upper and lower-case alphabetic characters, digits 0 – 9, punctuation marks and various non-printing control characters (such as carriage-return and backspace)

ASCII – examples

Symbol	decimal	Binary
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011

Binary Arithmetic

Binary Addition

- The binary addition operation works similarly to the base 10 decimal system, except that it is a base 2 system.
- The binary system consists of only two digits, 1 and 0.

Binary Addition

- The four rules of binary addition are:

  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 =10

Binary Addition

Example :

```
1 0 1 0
   1 0 1
________________
1 1 11
```

Example :

```
10001
11101
________________
 1 0 1 1 1 0
```

Binary Subtraction

- Binary subtraction is one of the four binary operations, where we perform the subtraction method for two binary numbers (comprising only two digits, 0 and 1).

Binary Subtraction Rules

0 – 0 = 0

0 – 1 = 1 (with a borrow of 1)

1 – 0 = 1

1 – 1 = 0

Binary Subtraction

Example :

```
1 1 0 0
 0 0 1 1
________
1 0 0 1
```

Example :

```
101101
 100111
________
 110
```

Binary Multiplication

Four major steps in binary digit multiplication are:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Binary Multiplication

Solve 110 × 11

```
      1   1   0
×         1   1
---------------
  ①   1   1   0
  1   1   0   x
---------------
1 0   0   1   0
```

1011.1 × 10.1

```
        1   0   1   1  .1
            ×   1   0  .1
-------------------------
        ①   ①
    ①   1   0   1   1   1
    0   0   0   0   0   x
1   0   1   1   1   x   x
-------------------------
1   1   1   0   0  .1   1
```

Boolean Algebra & Digital Logic

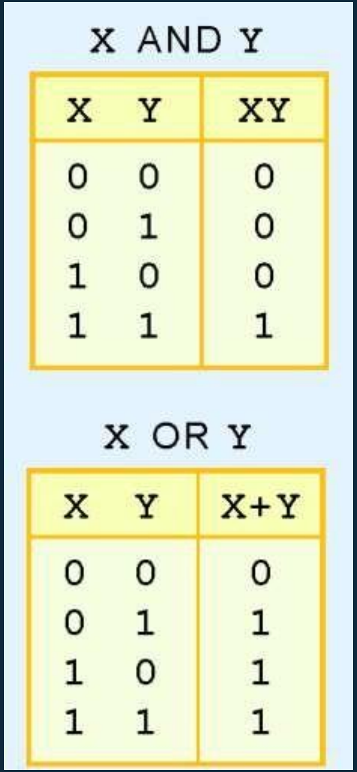
Boolean Algebra & Digital Logic

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are "true" and "false."
  - In digital systems, these values are "on" and "off,"
    - 1 and 0, or "high" and "low."
- Boolean expressions are created by performing operations on Boolean variables.
  - (Common Boolean operators include **AND**, **OR**,
    - and **NOT**.)

one unary operator "not" (symbolized by an over bar), two binary operators "+" and "."

Boolean Algebra

- A Boolean operation can be completely described using a truth table.
- The truth table for the Boolean operators AND and OR are shown at the right.
- The AND operation is also known as a Boolean product. The OR operation is the Boolean sum.



X AND Y

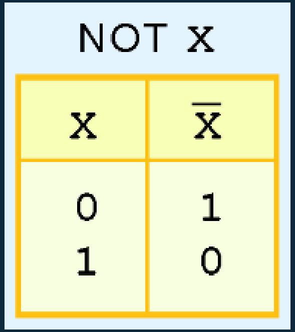
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

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

Boolean Algebra

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by an overbar



NOT x

x	$\overline{x}$
0	1
1	0

Boolean Algebra

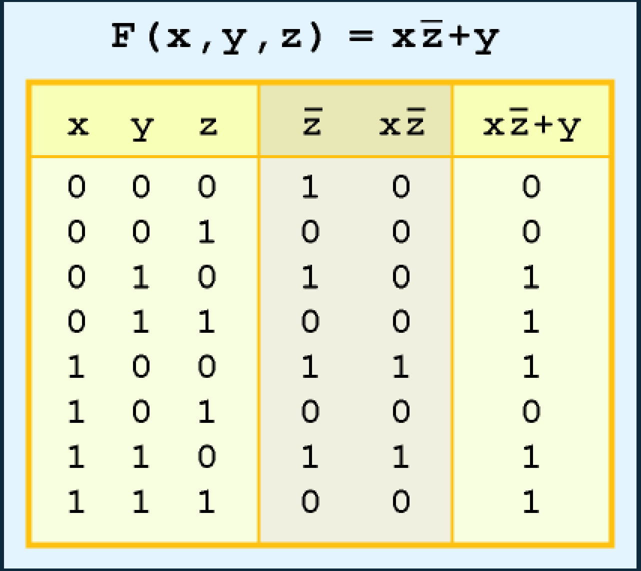
- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

Boolean Algebra

- The truth table for the Boolean function:

  $$F(x,y,z) = x\bar{z}+y$$

  - is shown at the right.
- To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.



$F(x,y,z) = x\bar{z}+y$

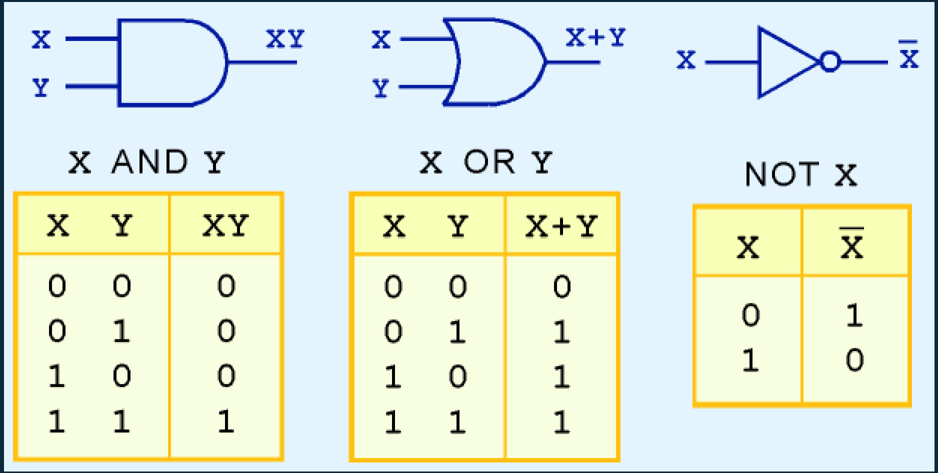
x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Logic Gates

- We have looked at Boolean functions in abstract terms.
- In this section, we see that Boolean functions are implemented in digital computer circuits called gates.
- A gate is an electronic device that produces a result based on two or more input values.
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a
    - particular purpose.

Logic Gates

❖ The three simplest gates are the AND, OR, and NOT gates.



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

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

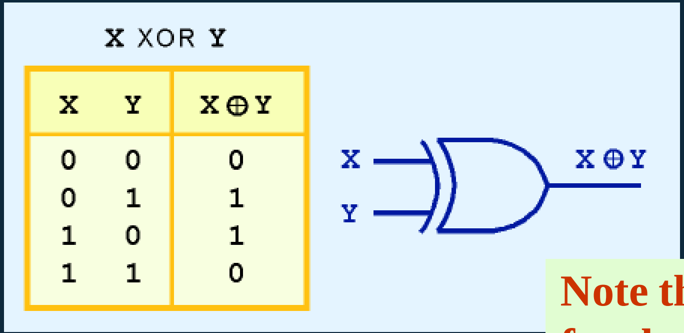
NOT X

X	$\overline{X}$
0	1
1	0

❖ They correspond directly to their respective Boolean operations, as you can see by their truth tables.

Logic Gates

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

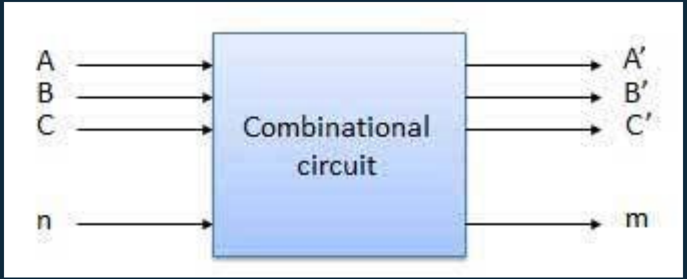

X XOR Y

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Note the special symbol □ for the XOR operation.

Introduction to Combinational Circuit

- In digital electronics, a combinational circuit is a circuit in which the output depends on the present combination of inputs.
- It can have n number of inputs and m number of output
- Combinational circuits are made up of logic gates. The output of each logic gate is determined by its logic function.

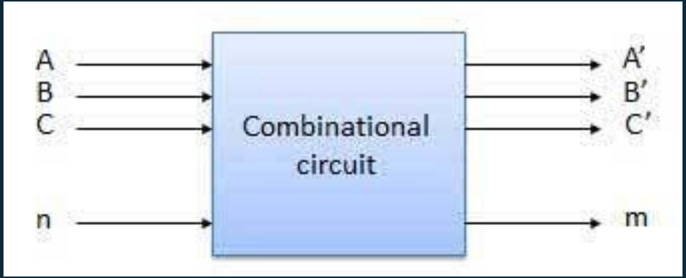


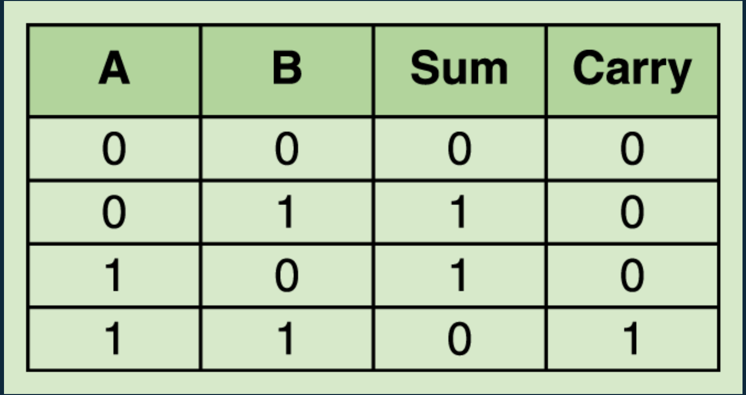
Adders

- At the digital logic level, addition is performed in binary
- Addition operations are carried out
  - by special circuits called **adders**

Half Adders

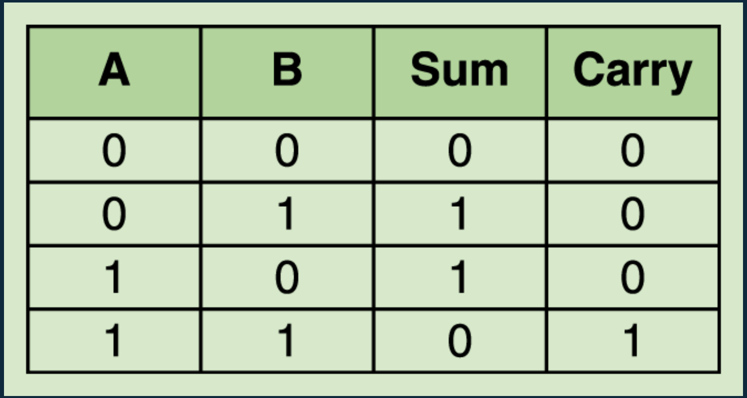
- A circuit that computes the **sum** of two bits and produces the correct carry bit is called a **half adder**
- The result of adding two binary digits could produce a carry value
- Recall that 1 + 1 = 10 in base two





A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

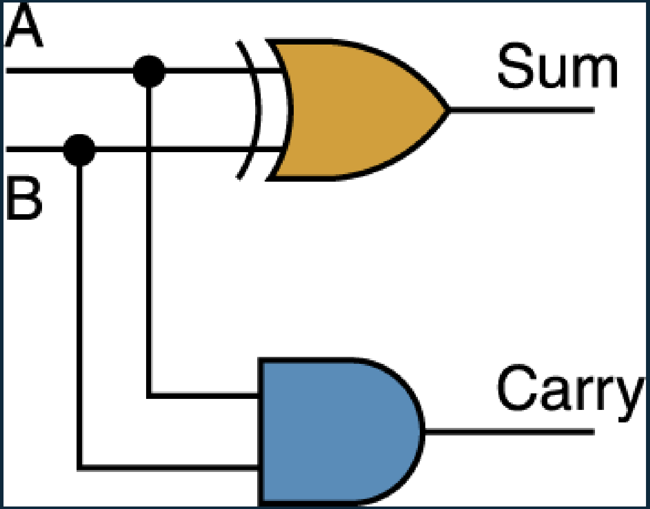
Half Adders



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

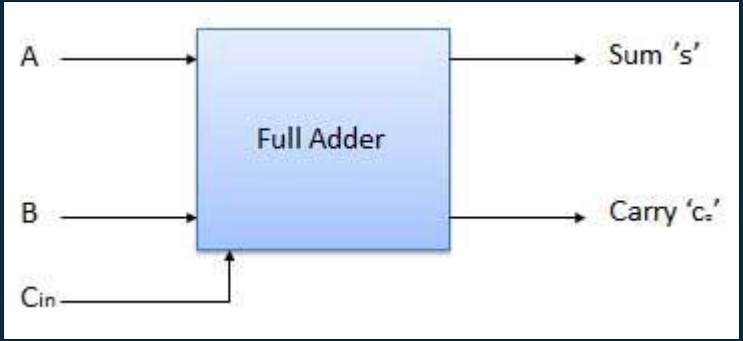
- Circuit diagram representing a **half adder**
- Two Boolean expressions:

sum = A □ B carry = AB



Full Adder

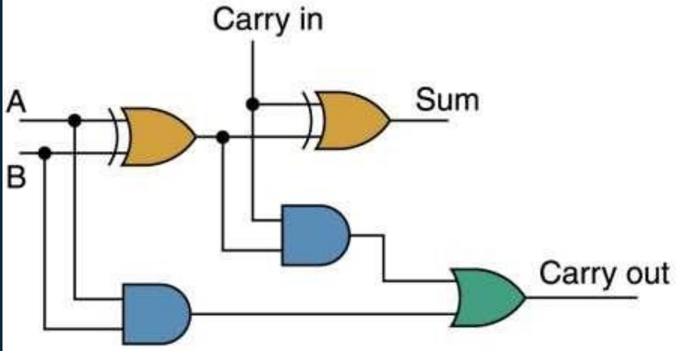
- The full adder is a three input and two output combinational circuit.

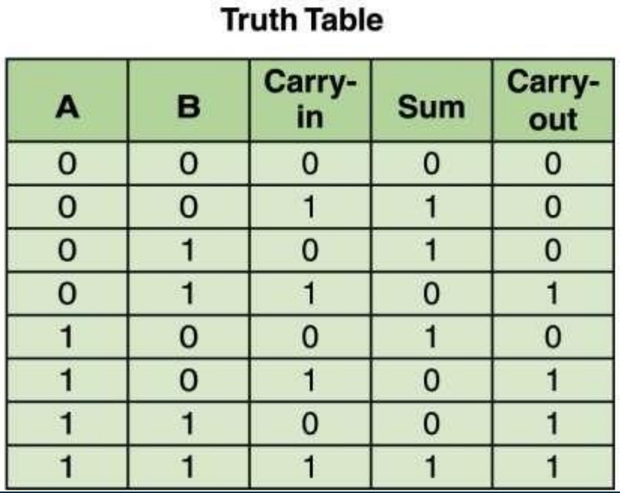


Full Adder

- The full adder is a three input and two output combinational circuit.

Logic Diagram





Truth Table

A	B	Carry-in	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1