# Digital Logic Design

Instructors:
Prof . Hala Zayed
Ass. Prof Mazen M. Selim
Dr Mona A. S. Ali

# Text Book

- Course Notes (obtained online from the e-learning portal of the national e-learning center)
- Essential Books
  - Logic and Computer Design Fundamentals, 2nd Edition, by M. M. Mano and C. R. Kime, published by Prentice Hall, 2003.
- Recommended Books
  - Digital Fundamentals ,eighth Edition by Thomas L. Floyd, published by Prentice Hall, 2005.

# Course Outline & Schedual

| W # | Topics | Textbook Sections |
|---|---|---|
| 1 | Number systems | Ch 1 sec1-5 (Dr Mona) |
| 2 | Digital codes | Ch 1 reminder (Dr Mona) |
| 3 | Logic Gates | Ch 2 all (Dr Mona) |
| 4 | Boolen Algebra | Ch 3 (Dr Mona) |
| 5 | Switching functions and canonical forms & **Quiz 1** | Ch 3 (Dr Mona) |
| 6 | Simplification using Karnaugh maps | Ch 4 (Dr Mona) |
| 7 | Digital combinational logic (decoders, encoders, multiplexers, demultiplexers) | Ch5 (Ass. Prof Mazen) |
| 8 | Digital combinational logic (adders and subtractors) | Ch5 (Ass. Prof Mazen) |
| 9 | Digital combinational logic (comparators, multipliers, dividers) | Ch 5(Ass. Prof Mazen) |
| 10 | Sequential logic and flip flop (part I) **Quiz 2** | Ch 6 (Prof Hala) |
| 11 | Sequential logic and flip flop (part 2) | Ch6 (Prof Hala) |
| 12 | Analysis of sequential circuits | Ch7 (Prof Hala) |
| 13 | Design of sequential circuits | Ch8 (Prof Hala) |
| 14 | Counter circuits | Ch9 (Prof Hala) |

# CHAPTER 1

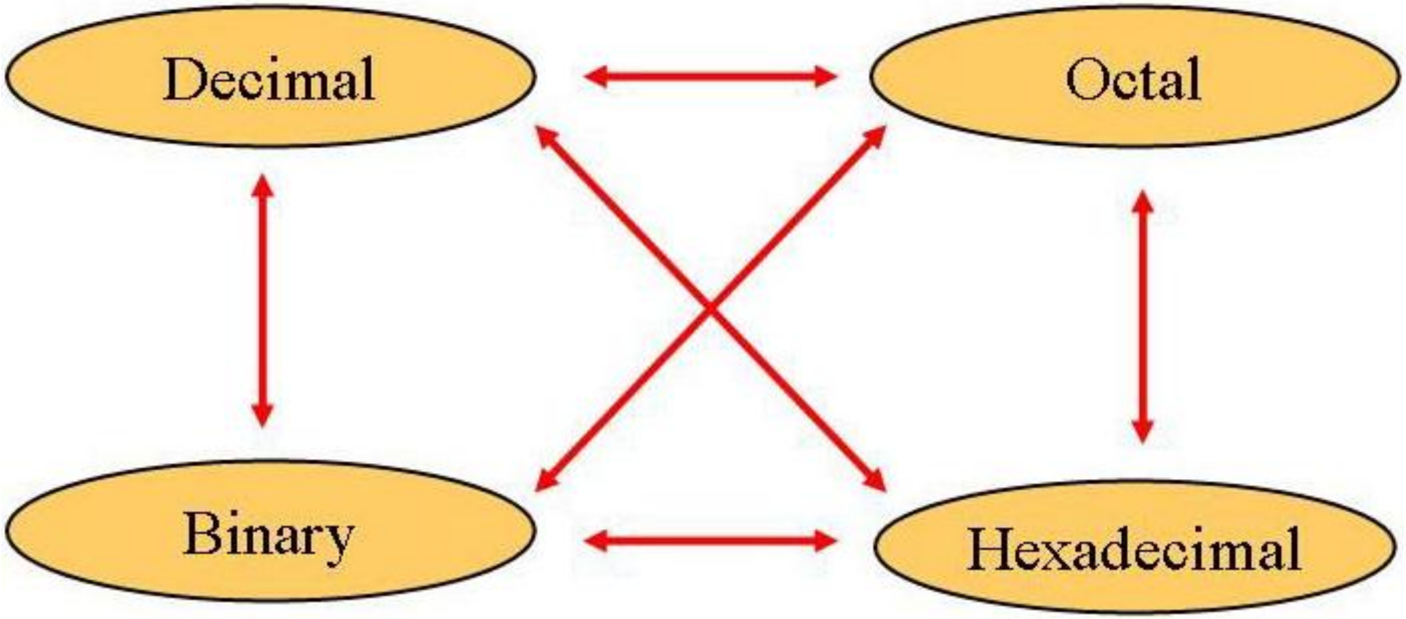
## NUMBER SYSTEMS AND CODES

# Contents

# Common Number System

| System | Base | Symbols | Used by humans? | Used in computers? |
|---|---|---|---|---|
| Decimal | 10 | 0, 1, … 9 | Yes | No |
| Binary | 2 | 0, 1 | No | Yes |
| Octal | 8 | 0, 1, … 7 | No | No |
| Hexa-decimal | 16 | 0, 1, … 9, A, B, … F | No | Yes |

# Conversion among Bases



# Quick example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base

# Decimal to Decimal (just for fun)

Weight

$$
\begin{array}{lrcr}
125_{10} \Rightarrow & 5 \times 10^{0} & = & 5 \\
 & 2 \times 10^{1} & = & 20 \\
 & 1 \times 10^{2} & = & 100 \\
\hline
 & & & 125
\end{array}
$$

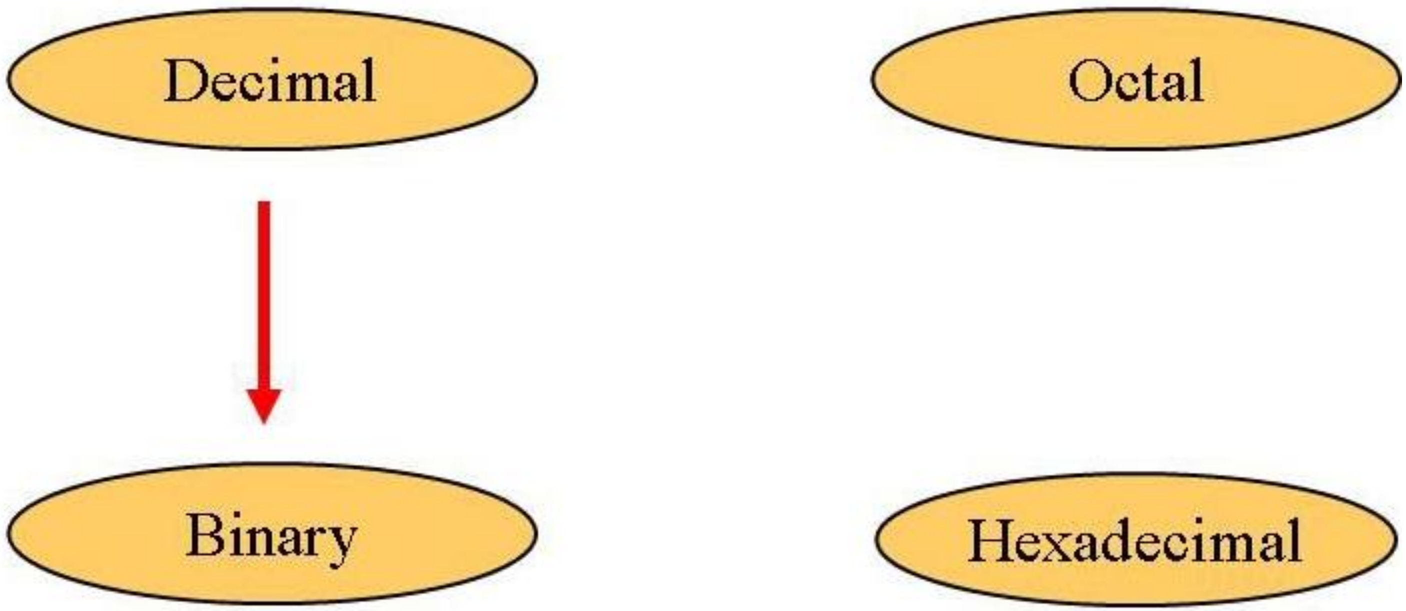
Base

# *BINARY NUMBERS*

- In the decimal numbering system, each position can represent 10 different digits from 0 to 9. each position has a weighting factor of powers of 10.
- $5621 = 1x10^0 + 2x10^1 + 6x10^2 + 5x10^3$
- In binary numbers, we can only use the digits 0 and 1 and the weights are powers of 2.

| $2^{10}$ | $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# Decimal to Binary



# Binary to Decimal Conversion

- To convert a binary number into decimal, we multiply each bit (binary digit) by the weight of its position and sum up the results.

$(11011011)_2 = 1x\ 2^0 + 1x\ 2^1 + 1x\ 2^3 + 1x\ 2^4 + 1x\ 2^6 + 1x2^7$

$= 1 + 2 + 8 + 16 + 64 + 128 = 219$

# Decimal to Binary Conversion

- There are two ways to make this conversion:
  - the repeated division-by-2-method (which you have studied before)
  - the sum of weights method

# Decimal to Binary

$125_{10} = ?_2$

| | | |
|---|---|---|
| 2 | 125 | |
| 2 | 62 | 1 |
| 2 | 31 | 0 |
| 2 | 15 | 1 |
| 2 | 7 | 1 |
| 2 | 3 | 1 |
| | 1 | 1 |

$125_{10} = 1111101_2$

# Sum of weights method

- To find a binary number that is equivalent to a decimal number, we can determine the set of binary weights whose sum is equal to the decimal number.

# Sum of weights method (contd.)

- **Example:**
- Convert the following decimal numbers to binary form: 13, 100, 65, and 189. Put your answer as eight bit numbers.
- **Answer:**

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 13 = | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 100 = | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 65 = | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 189 = | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

MSB LSB

# Binary To Decimal

Technique

- Multiply each bit by $2^n$, where $n$ is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

# Decimal to Binary

Bit “0”

$$
\begin{aligned}
101011_2 \Rightarrow \quad & 1 \times 2^0 = 1 \\
& 1 \times 2^1 = 2 \\
& 0 \times 2^2 = 0 \\
& 1 \times 2^3 = 8 \\
& 0 \times 2^4 = 0 \\
& 1 \times 2^5 = 32 \\
\hline
& \quad 43_{10}
\end{aligned}
$$

# Range of binary numbers

- Total combinations = $2^n$ different numbers in the range 0 to $(2^n - 1)$
- For example a 4-bit number can hold up to $2^4=16$ different values in the range 0 to 15 (0 to 1111).
- An 8-bit number can hold up to $2^8=256$ different values in the range 0 to 255 (0 to 11111111).

# Example

- What is the range of values (in decimal) that can be represented by a binary number of the following number of bits: 10, 20 and 24.
- Solution
- N=10 range = 0 to $2^{10} - 1$ = 0 to 1023 i.e. 1024 (1K)numbers
- N=20 range =0 to $2^{20} - 1$ =0 to 1048575 i.e. 1048576 (1M)numbers
- N=24 range =0 to $2^{24} - 1$ =0 to 16777215 i.e. 16777216 (16M)numbers

# OCTAL NUMBERS

- The eight allowable digits are 0,1,2,3,4,5,6 and 7 and the weights are powers of 8.

| Decimal | Binary | Octal |
| --- | --- | --- |
| 0 | 0 0 0 | 0 |
| 1 | 0 0 1 | 1 |
| 2 | 0 1 0 | 2 |
| 3 | 0 1 1 | 3 |
| 4 | 1 0 0 | 4 |
| 5 | 1 0 1 | 5 |
| 6 | 1 1 0 | 6 |
| 7 | 1 1 1 | 7 |
| 8 | 1 0 0 0 | 10 |
| 9 | 1 0 0 1 | 1 1 |
| 10 | 1 0 1 0 | 1 2 |
| 11 | 1 0 1 1 | 1 3 |

# Octal Conversions: binary to octal

- group the binary positions in groups of three
- Convert the following binary numbers into octal: a) 10110111 b) 01101100
- Solution
  - 10110111 = 010 110 111 = 267
  - 01101100 = 001 101 100 = 154

# Octal Conversions: octal to binary

- replace each octal number with three equivalent binary numbers even if the number can be represented by less than three bits
- Convert the following octal number into binary: a) 327 b)601
- Solution
  - a) 327 = 011 010 111 = 11010111
  - b) 601 = 110 000 001 = 110000001

# Octal Conversions: octal to decimal

- To convert from octal to decimal, (multiply by weighting factors).
- Convert $(713)_8$ to decimal.
- Solution
  - $713 = 7 \times 8^2 + 1 \times 8^1 + 3 \times 8^0 = 459$

# Octal Conversions: decimal to octal

- To convert from *decimal to octal,* the successive-division procedure or the sum of weights procedure can be used

# Octal Conversions (contd.)

- Convert the following decimal numbers to octal: a) $(596)_{10}$ b) $(100)_{10}$

| | $8^3$ | $8^2$ | $8^1$ | $8^0$ |
|---|---|---|---|---|
| | 512 | 64 | 8 | 1 |
| 596 = | 1 | 1 | 2 | 4 |
| 1000 = | 1 | 7 | 5 | 0 |

- Solution

LSB

a) $596 \div 8 = 74$ remainder 4

$74 \div 8 = 9$ remainder 2 1124

$9 \div 8 = 1$ remainder 1

$1 \div 8 = 0$ remainder 1

MSB

b) $1000 \div 8 = 125$ remainder 0

$125 \div 8 = 15$ remainder 5 1750

$15 \div 8 = 1$ remainder 7

$1 \div 8 = 0$ remainder 1

# HEXADECIMAL NUMBERS

- The 16 allowable digits are 0,1,2,3,4,5,6,7,8,9,A, B,C,D,E and F
- the weights are powers of 16.

| Decimal | Binary | Hexadecimal |
|---|---|---|
| 0 | 0000 0000 | 0 0 |
| 1 | 0000 0001 | 0 1 |
| 2 | 0000 0010 | 0 2 |
| 3 | 0000 0011 | 0 3 |
| 4 | 0000 0100 | 0 4 |
| 5 | 0000 0101 | 0 5 |
| 6 | 0000 0110 | 0 6 |
| 7 | 0000 0111 | 0 7 |
| 8 | 0000 1000 | 0 8 |
| 9 | 0000 1001 | 0 9 |
| 10 | 0000 1010 | 0 A |
| 11 | 0000 1011 | 0 B |
| 12 | 0000 1100 | 0 C |
| 13 | 0000 1101 | 0 D |
| 14 | 0000 1110 | 0 E |
| 15 | 0000 1111 | 0 F |
| 16 | 0001 0000 | 1 0 |
| 17 | 0001 0001 | 1 1 |
| 18 | 0001 0010 | 1 2 |
| 19 | 0001 0011 | 1 3 |
| 20 | 0001 0100 | 1 4 |

# Hexadecimal Conversion: binary to hexadecimal

- grouping the binary positions in groups of four
- Convert the following binary numbers into hexadecimal: a) 10101111   b) 01101100
- **Solution:**
  - 10110111 = 1011 0111 = $(B\ 7)_{16}$
  - 01101100 = 0110 1100 = $(6\ C)_{16}$

# Hexadecimal Conversion: hex to binary

- replace each hexadecimal number with four equivalent binary numbers even if the number can be represented by less than four bits
- Convert the following hexadecimal number into binary: a) A2E b)60F
- **Solution:**

14

  - a) $(A2E)_{16}$ = 1010 0010 1110

14

    = $(101000101110)_2$
  - b) $(60F)_{16}$ = 0110 0000 1111

    = $(011000001111)_2$

# Hexadecimal Conversion: hex to decimal

- To convert from hexadecimal to decimal, (multiply by weighting factors).
- Convert (7AD)16 to decimal.
- **Solution:**
  - $(7AD)_{16} = 7 \times 16^2 + 10 \times 16^1 + 13 \times 16^0$
    $= (1965)_{10}$

# Hexadecimal Conversion: decimal to hex

- To convert from *decimal to hexadecimal,* the successive-division procedure or the sum of weights procedure can be used.
- Convert the following decimal numbers to hexadecimal: a) $(596)_{10}$ b) $(100)_{10}$
- **Solution:**
  - 596 ÷ 16 = 37 remainder 4
  - 37 ÷ 16 = 2 remainder 5 254
  - 2 ÷ 16 = 0 remainder 2

  - 1000 ÷ 16 = 62 remainder 8
  - 62 ÷ 16 = 3 remainder 14 3E8
  - 3 ÷ 16 = 0 remainder 3

# Exercise

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 33 | | | |
| | 1110101 | | |
| | | 703 | |
| | | | 1AF |

# Exercise

**Answer**

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 33 | 100001 | 41 | 21 |
| 117 | 1110101 | 165 | 75 |
| 451 | 111000011 | 703 | 1C3 |
| 431 | 110101111 | 657 | 1AF |

# Binary Addition

Two 1-bit values

| A | B | A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

“two”

# Binary Addition

Two *n*-bit values

- Add individual bits
- Propagate carries
- E.g.,

$$
\begin{array}{rcccccc}
 & 1 & & & & 1 & \\
 & & 1 & 0 & 1 & 0 & 1 \\
+ & & 1 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 1 & 1 & 0
\end{array}
\qquad
\begin{array}{rr}
 & 21 \\
+ & 25 \\
\hline
 & 46
\end{array}
$$

# Binary Arithmetic

- Binary Addition

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | **1** | **1** | **1** | **1** | | | |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1** | | | **1** | **1** | **1** | **1** | |
| | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| **1** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

- 11101101 + 01000011 = 100110000 This example shows that the result could not fit in 8-bits ( 237 + 67 = 304) and the maximum capacity of 8-bits is 255. That is what we call **overflow**.

# Binary Subtraction

- The four cases for subtracting binary digits (A - B) are as follows

| A | B | D | B |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

- D is the difference and B is the borrow

# Example

- Subtract the following binary numbers and put the result in 8-bits. Verify your answer by converting into decimal:
- a) 10111111 - 01111100
  - 10111111 - 01111100 = 01000011 (191 - 124 = 67)
- b) 11101101 – 01000011
  - 11101101 - 01000011 = 10101010 (237 – 67 = 170)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **10** | | | | | | |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | **0** | **10** | |
| 1 | 1 | 1 | 0 | 1 | ~~1~~ | ~~0~~ | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

# Multiplication

Decimal (just for fun)

```
   35
x 105
-----
  175
 000
 35
-----
 3675
```

# Multiplication

Binary, two 1-bit values

| A | B | $A \times B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Multiplication

Binary, two *n*-bit values

- As with decimal values
- E.g.,

```
     1110
   x 1011
 --------
     1110
    1110
   0000
  1110
 --------
 10011010
```

# Binary Multiplication

- a) 11100 x 101 = 10001100
  - (16+8+4) x (4+1) = (128+8+4)
  - 28 x 5 = 140
- b) 11011 x 1101 = 101011111
  - (16+8+2+1) x (8+4+1) = (256+64+16+8+4+2+1)
  - 27 x 13 = 351

$$
\begin{array}{cccccccc}
 & & & 1 & 1 & 1 & 0 & 0 \\
 & & & & & 1 & 0 & 1 \\
\hline
 & & & 1 & 1 & 1 & 0 & 0 \\
 & & 0 & 0 & 0 & 0 & 0 & \\
 & 1 & 1 & 1 & 0 & 0 & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
$$

$$
\begin{array}{c|cccccccc}
 & & & & & 1 & 1 & 0 & 1 & 1 \\
 & & & & & & 1 & 1 & 0 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 & 1 \\
 & & & & 0 & 0 & 0 & 0 & 0 & \\
 & & & 1 & 1 & 0 & 1 & 1 & & \\
 & & 1 & 1 & 0 & 1 & 1 & & & \\
\hline
\mathbf{1} & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

# Binary Division

- 11001 ÷ 101 = 101
- (16+8+1) ÷ (4+1) = (4+1)
- 25 ÷ 5 = 5

$$
\begin{array}{r|rrrrr}
 & & & 1 & 0 & 1 \\
\cline{2-6}
1\ 0\ 1 & 1 & 1 & 0 & 0 & 1 \\
\cline{1-1}
 & 1 & 0 & 1 & & \\
\cline{2-6}
 & & & 1 & 0 & 1 \\
 & & & 1 & 0 & 1 \\
\cline{4-6}
 & & & 0 & 0 & 0
\end{array}
$$

# *1’s and 2’s COMPLEMENTS*

- 1’s and 2’s complement allow the representation of negative numbers in binary.
- ***The 1's complement of a binary number is found by simply changing all 1s to 0s and all 0s to 1s.***
- <u>Examples</u>
- The 1’s complement of 10001111 = 01110000 .
- The 1’s complement of 01101100 = 10010011 .
- The 1’s complement of 00110011 = 11001100 .

# 2's complement

- *The 2's complement of a binary number is found by adding 1 to the LSB of the 1 's complement.*
- *Another way of obtaining the 2’s complement of a binary number is to start with the LSB (the rightmost bit) and leave the bits unchanged until you find the first 1. Leave the first 1 unchanged and complement the rest of the bits (change 0 to 1 and 1 to 0).*

# *2's complement*

- Example
  - The 2’s complement of 10001111
    = 01110000 +1 = 01110001
  - The 2’s complement of 01101100
    = 10010011 + 1 =10010100
  - The 2’s complement of 00110011
    = 11001100 + 1 = 11001101

# REPRESENTATION OF SIGNED NUMBERS

- There are three basic ways to represent signed numbers:
  - sign-magnitude
  - 1’s complement
  - 2’s complement.

# Sign-Magnitude

- The number consists of two parts:
  - the MSB (most significant bit) represents the sign
  - the other bits represent the magnitude of the number.
- If the sign bit is 1 the number is negative and if it is 0 the number is positive.

# Examples: decimal to sign-magnitude

- -30 = 1 0011110 (The leftmost 1 indicates that the number is negative. The remaining 7-bits carry the magnitude of 30)
- 30 = 0 0011110 (The only difference between –30 and +30 is the sign bit because the magnitude bits are similar in both numbers.)
- -121 = 1 1111001
- 99 = 0 1100011

# Examples: sign-magnitude to decimal

- 10111001 = -57 (The leftmost 1 indicates that the number is negative. The remaining 7-bits carry the magnitude of 57)
- 11111111 = -127 (The minimum number that can be represented in an 8-bit register using sign-magnitude representation)
- 01111111 = +127 (The maximum number that can be represented in an 8-bit register using sign-magnitude representation)

# Range of numbers in Sign-Magnitude Representation

- for an n-bit number, the range of values that could be represented using sign-magnitude notation is from
  - $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$.
- For example if n=8 the range is from –127 to 127

# Representation of negative numbers in 1's Complement

- Negative numbers are represented in 1's complement format
- positive numbers are represented as the positive sign-magnitude numbers

# Examples: decimal to 1's complement

- 30 = 00011110
- -30 = 11100001
  - the number equals the 1's complement of 30
- 121 = 01111001
- -121 = 10000110
- 99 = 01100011

# Examples: 1's complement to decimal

- 10111001 = -01000110 = -70
  - The leftmost 1 indicates that the number is negative. Take the 1's complement of the number to get the magnitude of 70
- 11111111 = -00000000 = -0
  - there are two representations of zero
- 01111111 = +127
  - The maximum +ve number
- 10000000 = -01111111 = -127
  - The maximum –ve number

# Range of numbers in 1's complement Representation

- $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$.
  - exactly the same as the range of numbers in sign-magnitude

# Representation of negative numbers in 2's Complement

- Negative numbers are represented in 2′s complement format
- Positive numbers are represented exactly the same way as in sign-magnitude and in 1′s complement

# Examples: decimal to 2's complement

- 30 = 00011110
- -30 = 11100010
  - the number equals the 2's complement of 30
- 121 = 01111001
- -121 = 10000111
- 99 = 01100011

# Examples: 2’s complement to decimal

- 10111001 = -01000111 = -71
  - The leftmost 1 indicates that the number is negative.
  - Take the 2’s complement of the number to get the magnitude of 71
- 11111111 = -00000001 = -1
  - No two representations of zero
- 01111111 = +127
  - The maximum +ve number
- 10000000 = -10000000 = -128
  - The minimum –ve number

# Range of numbers in 2's complement Representation

- $-(2^{n-1})$ to $+(2^{n-1}-1)$
- if n=8 the range is from −128 to 127

# 2's Complement Evaluation

- Positive and negative numbers in the 2's complement system are evaluated by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros.
- The weight of the sign bit in a negative number is given a negative value

# EXAMPLE

- 01010110 = 64 + 16 + 4 + 2 = +86
-

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | 2° |
|---|---|---|---|---|---|---|---|
| **0** | **1** | **0** | **1** | **0** | **1** | **1** | **0** |

- 10101010 = -128 + 32 + 8 + 2 = -86

| $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | 2° |
|---|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **0** | **1** | **0** | **1** | **0** |

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS (ADDITION)

- **Both numbers positive:**
  - 00000111 7
  - + 00000100 + 4
  - 00001011 11
- **Positive number with magnitude larger than negative number:**
  - 00001111 15
  - + 11111010 + -6
  - Discard carry *1* 00001001 9

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS (ADDITION)

- **Negative number with magnitude larger than positive number:**
  - 00010000 16
  - + 11101000 + -24
  -
  - 11111000 -8
- **Both numbers negative:**
  - 11111011 —5
  - + 11110111 + -9
  - Discard carry—> 1 11110010 -14

# Overflow Condition

- When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an **overflow** results
- incorrect sign bit
- only when both numbers are positive or both numbers are negative

# Example

| | | |
|---|---|---|
| | 01111101 | 125 |
| + | 00111010 | + 58 |
| | 10110111 | 183 |

- Incorrect sign
- Incorrect magnitude
- What if we have an extra bit?

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS (Subtraction)

- *the subtraction operation changes the sign of the subtrahend and adds it to the minuend.*
- *Example:* 10001000 – 11100010
  - Try in your notebook.

# solution

- 10001000 – 11100010
- -120 - (-30) = -120 + 30 = -90

| | |
|---|---|
| 10001000 | Minuend (-120) |
| + 00011110 | 2's complement of subtrahend (+30) |
| 10100110 | Difference (-90) |