Digital System Design

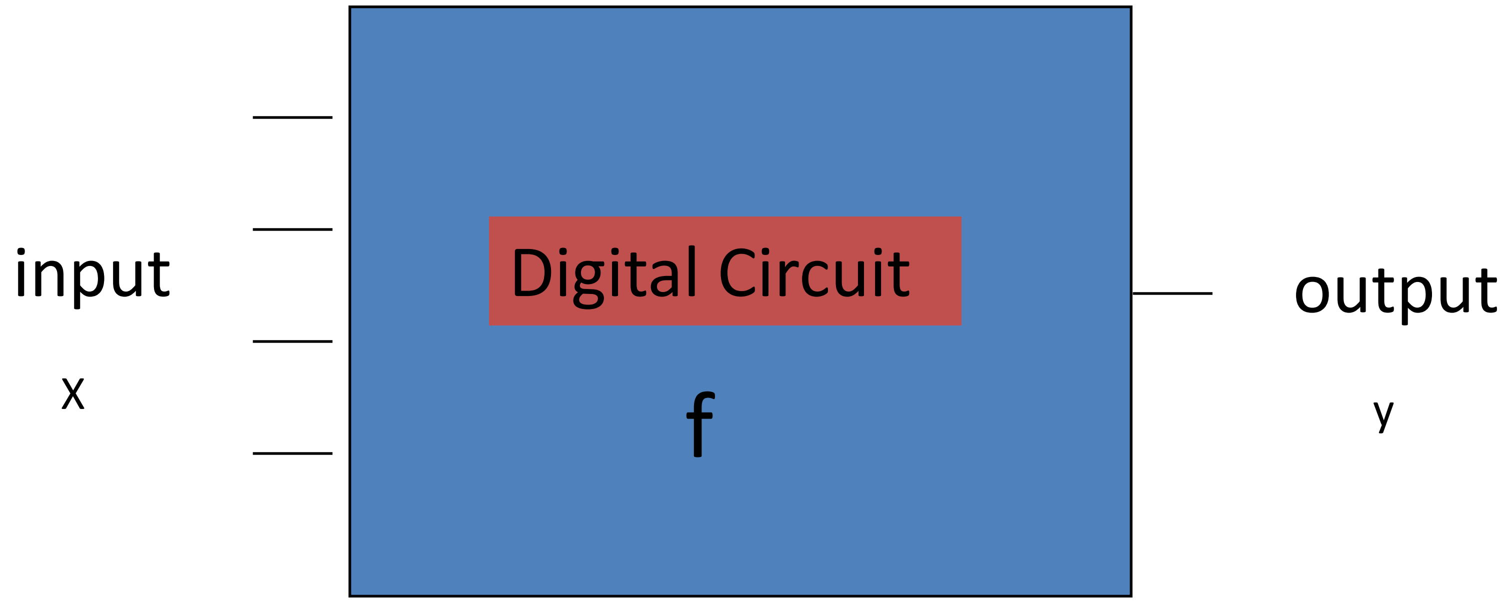
Textbook:

- Digital Design. M. Morris Mano and Michael Ciletti. Pearson Education.
- *Digital Design Principles and Practices – 4th Ed.,*John F. Wakerly, Prentice Hall
- *Logic and Computer Design Fundamentals*, M. Morris Mano and Charles R. Kime (4th edition, 2008). Prentice Hall.

Digital Systems Design

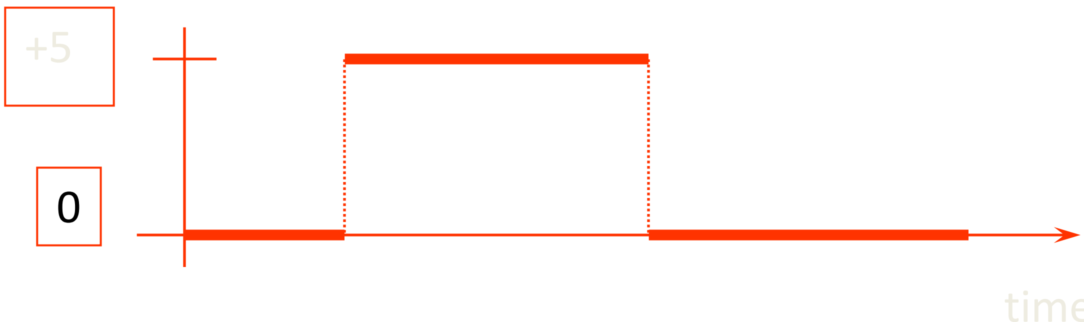
- **Digital Systems**
  transform signals that can be abstracted as discrete in range and domain
- **Design**
  process of coming up with a solution to a problem

Design: Given a specification / behavior of y, design / build system / circuit f (x)



Digital versus Analog

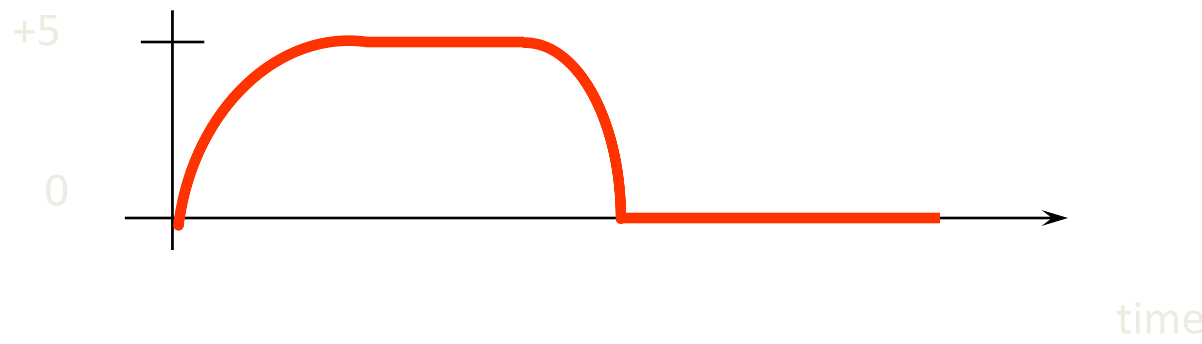
- **Digital systems** have inputs and outputs that are represented by <u>discrete values</u>



Binary digital systems have exactly two possible input / output values, i.e., 0 or +5 V.

Digital versus Analog

- Analog systems have inputs and outputs that take on a continuous range of values



Pros & cons of digital vs analog

- Digital systems have inherent ability to deal with electrical signals that have been **degraded** by transmission through circuits

- The real world operates in an analog fashion- that is continuously;
  - thus digital systems need interface devices (sensor, actuators, converters) to control analog devices

Advantages of Digital Techniques

1. Easy storage of information
2. Accuracy and precision
3. Easier to design
4. Programmability
5. Less affected by noise
6. Easier fabrication processes

Number Systems

Introduction

- A ***bit*** is the most basic unit of information in a computer.
  - It is a state of “on” or “off” in a digital circuit.
  - Sometimes they represent **high** or **low** voltage

  – A ***byte*** is a group of eight bits.. It is the smallest possible *addressable* unit of computer storage.

- A ***word*** is a contiguous group of bytes.
  - Words can be any number of bits or bytes.
  - Word sizes of 16, 32, or 64 bits are most common.

Common Number Systems

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	No

Quantities/Counting

Decimal	Binary	Octal	Hexa-decimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

Decimal	Binary	Octal	Hexa-decimal
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Decimal	Binary	Octal	Hexa-decimal
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17

Decimal Number

- Base (Radix) is 10
- Digits (0,1, . . 9)
- Each position carries a weight.
- If we were to write 1936.25 using a power series expansion and base 10 arithmetic:

$$1\times10^3+9\times10^2+3\times10^1+6\times10^0+2\times10^{-1}+5\times10^{-2}$$

MSD

Weights:	10^3	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-3}

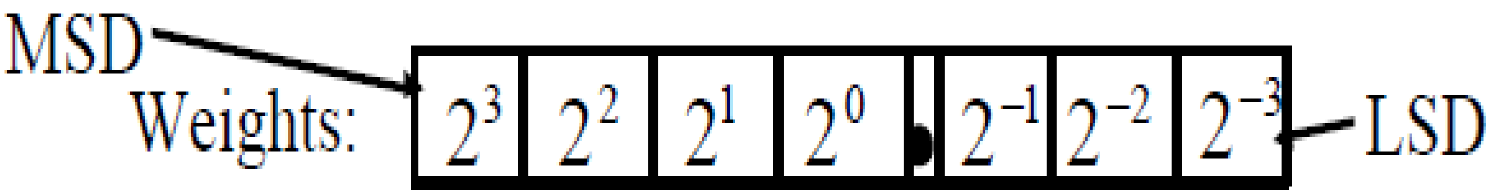
LSD

Binary Numbers

- Strings of binary digits (“bits”)
  - One bit can store a number from the set (0 , 1)
  - *n* bits can store numbers from 0 to 2^n-*1*

Binary – Powers of 2

- Positional representation
- Each digit represents a power of 2



- If we write 10111.01 using a decimal power series we convert from binary to decimal:

$$1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} =$$

$$= 1\times 16 + 0\times 8 + 1\times 4 + 1\times 2 + 1\times 1 + 0\times 0.5 + 1\times 0.25 = 23.25$$

Hexadecimal

- Strings of 0s and 1s too hard to write
- Use base-16 or <u>hexadecimal</u> – 4 bits
- The first 10 digits are borrowed from the decimal system and the letters A, B, C, D, E, F are used for the digits 10, 11, 12, 13, 14, 15

Dec	Bin	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Dec	Bin	Hex
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Octal System

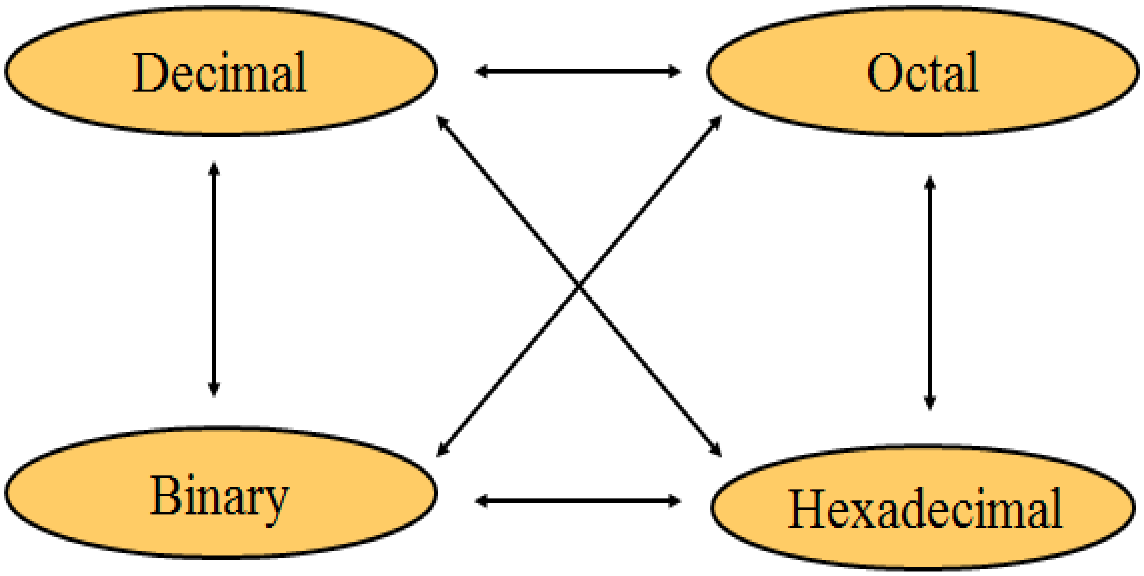
- Its base is 8
- Eight digits 0, 1, 2, 3, 4, 5, 6, 7

$$(236.4)_8 = (\ ?\ )_{10}$$

$$2\times 8^2 + 3\times 8^1 + 6\times 8^0 + 4\times 8^{-1} = 158.5$$

Conversion Among Bases

- The possibilities:



Example

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

Base

Decimal to Decimal

Weight

$$
\begin{aligned}
125_{10} \Rightarrow \quad & 5 \times 10^{0} &= \quad 5 \\
& 2 \times 10^{1} &= \quad 20 \\
& 1 \times 10^{2} &= \underline{\;100\;} \\
& & 125
\end{aligned}
$$

Base

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

Bit “0”

$101011_2 \Rightarrow$

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

- What is 10011100 in decimal?

1	0	0	1	1	1	0	0		

$$128 + 0 + 0 + 16 + 8 + 4 + 0 + 0 = 156$$

Decimal to Binary

- Technique I

1. Find largest power-of-two smaller than decimal number
2. Make the appropriate binary digit a ‘1’
3. Subtract the “power of 2” from decimal
4. Do the same thing again

Example

- Convert 28 decimal to binary

32 is too large, so use 16

Binary $\rightarrow$ 10000 Decimal $\rightarrow$ 28 – 16 = 12

Next is 8

Binary $\rightarrow$ 11000 Decimal $\rightarrow$ 12 – 8 = 4

Next is 4

Binary $\rightarrow$ 11100 Decimal $\rightarrow$ 4 – 4 = 0

- Technique II
  - Divide by two, keep track of the remainder
  - First remainder is bit 0 (LSB, least-significant bit)
  - Second remainder is bit 1

Example

$125_{10} = ?_2$

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

$125_{10} = 1111101_2$

- Conversion from decimal fraction to binary:
  - same method used for integers except multiplication is used instead of division.

Convert $(0.8542)_{10}$ to binary (give answer to 6 digits).

0.8542 x 2 =	1	+	0.7084	$a_{-1} = 1$	**MSB**
0.7084 x 2 =	1	+	0.4168	$a_{-2} = 1$	
0.4168 x 2 =	0	+	0.8336	$a_{-3} = 0$	
0.8336 x 2 =	1	+	0.6672	$a_{-4} = 1$	
0.6675 x 2 =	1	+	0.3344	$a_{-5} = 1$	
0.3344 x 2 =	0	+	0.6688	$a_{-6} = 0$	**LSB**

$$(0.8542)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6})_2 = (0.110110)_2$$

Hexadecimal to Binary

- Technique
  - Convert each hexadecimal digit to a 4-bit equivalent binary representation

Example

$10AF_{16} = ?_2$

1	0	A	F
↓	↓	↓	↓
0001	0000	1010	1111

$10AF_{16} = 0001000010101111_2$

Binary to Hexadecimal

- Convert groups of 4 bits

0101 | 0011 | 0111 | 1011

5 3 7 B

$101001101111011 = (537B)_{16}$

Hexadecimal to Binary

- Technique
  - Convert each hexadecimal digit to a 4-bit equivalent binary representation

Example

$10AF_{16} = ?_2$

1	0	A	F
↓	↓	↓	↓
0001	0000	1010	1111

$10AF_{16} = 0001000010101111_2$

Octal to Binary

- Technique
  - Convert each octal digit to a 3-bit equivalent binary representation

Example

$705_8 = ?_2$

$$\begin{array}{ccc} 7 & 0 & 5 \\ \downarrow & \downarrow & \downarrow \\ 111 & 000 & 101 \end{array}$$

$705_8 = 111000101_2$

Decimal to Octal

- Technique
  - Divide by 8
  - Keep track of the remainder

Example

$1234_{10} = ?_8$

$$
\begin{array}{r|r l}
8 & 1234 & \\ \hline
8 & 154 & 2 \\ \hline
8 & 19 & 2 \\ \hline
8 & 2 & 3 \\ \hline
 & 0 & 2
\end{array}
$$

$1234_{10} = 2322_8$

Decimal to Hexadecimal

- Technique
  - Divide by 16
  - Keep track of the remainder

Example

$1234_{10} = ?_{16}$

$$
\begin{array}{r|l l}
16 & 1234 & \\
\hline
16 & \quad 77 & 2 \\
\hline
16 & \quad\ 4 & 13 = D \\
\hline
 & \quad\ 0 & 4
\end{array}
$$

$1234_{10} = 4D2_{16}$

Common Powers

- Base 10

Power	Preface	Symbol	Value
10^{-12}	pico	p	.000000000001
10^{-9}	nano	n	.000000001
10^{-6}	micro	µ	.000001
10^{-3}	milli	m	.001
10^{3}	kilo	k	1000
10^{6}	mega	M	1000000
10^{9}	giga	G	1000000000
10^{12}	tera	T	1000000000000

- Base 2

Power	Preface	Symbol	Value
2^{10}	kilo	k	1024
2^{20}	mega	M	1048576
2^{30}	Giga	G	1073741824

Binary Number Systems

Binary Number Systems

- Unsigned Binary Code
- Signed Binary Codes
- Floating-Point System

Unsigned Binary Code

- The Unsigned Binary Code is used to represent positive integer numbers.
- What is the range of values that can be represented with n bits in the Unsigned Binary Code?

$$[0,\ 2^n-1]$$

Unsigned Binary Code

- Example 1: Represent $(26)_{10}$ in Unsigned Binary Code

$$26_{10} = \quad 11010$$

- Example 2: Represent $(26)_{10}$ in Unsigned Binary Code using 8 bits.

$$26_{10} = \; 00011010$$

- Example 3: Represent $(26)_{10}$ in Unsigned Binary Code using 4 bits.

Unsigned Binary Code (4 bits)

Unsigned	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Unsigned Binary Code: Arithmetic & Logic Operations

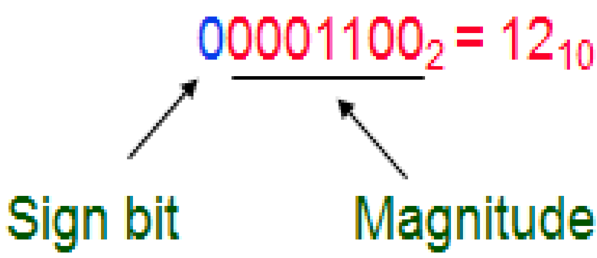
- Arithmetic Operations:
  - **Addition**
  - **Subtraction**
  - **Multiplication**
  - **Division**
- Logic Operations
  - **AND** **CONJUNCTION**
  - **OR** **DISJUNCTION**
  - **NOT** **NEGATION**
  - **XOR** **EXCLUSIVE OR**

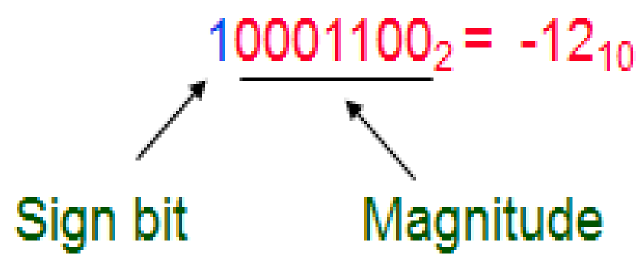
Signed Binary Codes

- These are codes used to represent positive and negative numbers.
- Three types of signed binary number representations:
  - signed magnitude
  - 1's complement
  - 2's complement

How To Represent Signed Numbers

- In each case: left-most bit indicates sign: positive (0) or negative (1).
- The remaining bits represent the magnitude of the number





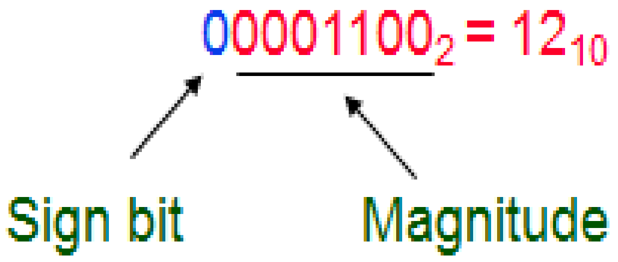
Example:

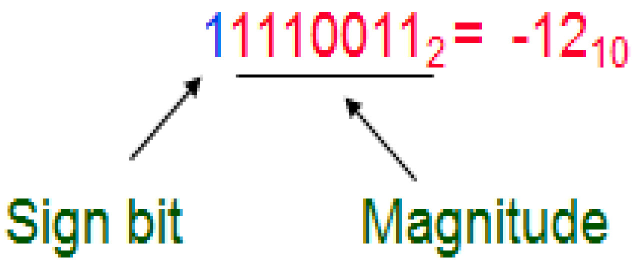
Sign & Mag. Code	Decimal
01101	+13
11101	-13
00101	+5
10101	-5

One’s Complement Representation

- The one’s complement of a binary number involves inverting all bits.
  - 1’s comp of 00110011 is 11001100
  - 1’s comp of 10101010 is 01010101
- For an n bit number N, the 1’s complement is $(2^n-1) – N$.

- To find negative of 1's complement number take the 1's complement.





Two’s Complement Representation

- The two’s complement of a binary number involves inverting all bits and adding 1.
  - 2’s comp of 00110011 is 11001101
  - 2’s comp of 10101010 is 01010110
- For an n bit number N the 2’s complement is $(2^n-1) – N + 1$.

- To find negative of 2’s complement number, take the 2’s complement.

$$0\underline{0001100}_2 = 12_{10}$$

Sign bit Magnitude

$$1\underline{1110100}_2 = -12_{10}$$

Sign bit Magnitude

Two’s Complement Shortcuts

- Algorithm 1 – Simply complement each bit and then add 1 to the result.
  - Finding the 2’s complement of $(01100101)_2$ and of its 2’s complement

N = 01100101 [N] = 10011011

```
  10011010          01100100
+        1      +          1
-------------   -------------
  10011011          01100101
```

Two's Complement Shortcuts

- Algorithm 2 – Starting with the least significant bit, copy all of the bits up to and including the first 1 bit and then complementing the remaining bits.

  N = 0 1 1 0 0 1 0 1

  [N] = 1 0 0 1 1 0 1 1

- Machines that use 2’s complement arithmetic can represent integers in the range

$$-2^{n-1} <= N <= 2^{n-1}-1$$

where n is the number of bits available for representing N.

- For 2’s complement, more negative numbers than positive.
- For 1’s complement, two representations for zero.
- 2’s complement most important (only 1 representation for zero).

1’s Complement Addition

- Using 1’s complement numbers, adding numbers is easy.
  - Step 1: Add binary numbers
  - Step 2: Add carry to low-order bit

- For example, suppose we wish to add $(12)_{10} + (1)_{10}$.

  $(12)_{10} = +(1100)_2 = 01100_2$ in 1's comp.

  $(1)_{10} = +(0001)_2 = 00001_2$ in 1's comp.

			0	1	1	0	0
Add	+		0	0	0	0	1
		0	0	1	1	0	1
Add carry							0
Final Result			0	1	1	0	1

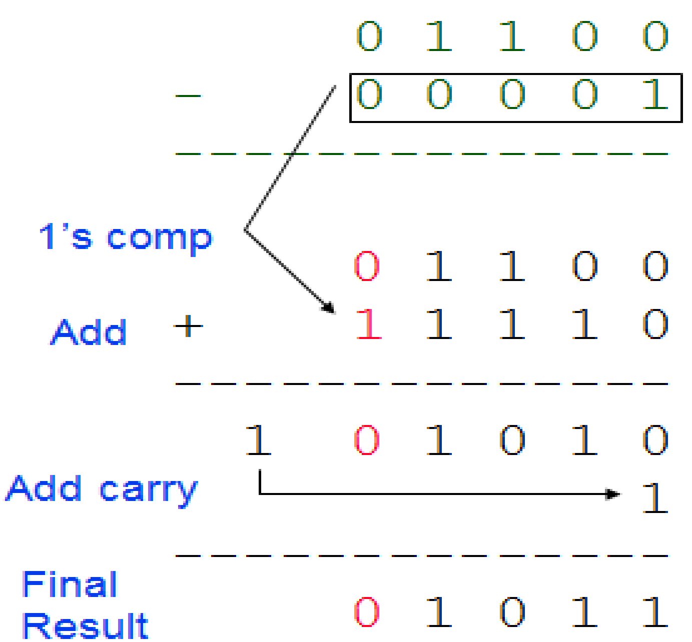
1's Complement Subtraction

- Using 1's complement numbers, subtracting numbers is also easy.
  - Step 1: Take 1's complement of 2^{nd} operand
  - Step 2: Add binary numbers
  - Step 3: Add carry to low order bit

- For example, Let's compute $(12)_{10}$ - $(1)_{10}$.

  $(12)_{10}$ = $+(1100)_2$ = 01100_2 in 1's comp.

  $(-1)_{10}$ = $-(0001)_2$ = 11110_2 in 1's comp.



2’s Complement Addition

- Using 2’s complement numbers, adding numbers is easy.
  - Step 1: Add binary numbers
  - Step 2: Ignore carry bit

2's Complement Addition

- Let's compute $(12)_{10} + (1)_{10}$.

  $(12)_{10} = +(1100)_2 = 01100_2$ in 2's comp.

  $(1)_{10} = +(0001)_2 = 00001_2$ in 2's comp.

			0	1	1	0	0
Add	+		0	0	0	0	1
Final Result		0	0	1	1	0	1
		Ignore					

2's Complement Subtraction

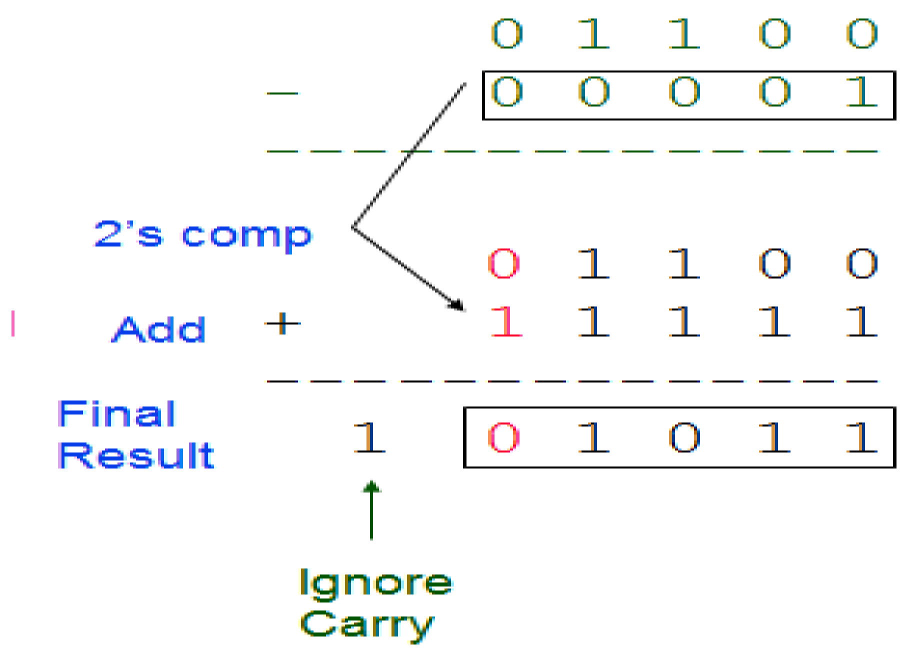
- Step 1: Take 2's complement of 2^{nd} operand
- Step 2: Add binary numbers
- Step 3: Ignore carry bit

2's Complement Subtraction

- Let's compute $(12)_{10}$ - $(1)_{10}$.

  $(12)_{10}$ = $+(1100)_2$ = 01100_2 in 2's comp.

  $(-1)_{10}$ = $-(0001)_2$ = 11111_2 in 2's comp.



- Let’s compute $(13)_{10} - (5)_{10}$.

  $(13)_{10} = +(1101)_2 \qquad = (01101)_2$

  $(-5)_{10} = -(0101)_2 = (11011)_2$

```
                      0 1 1 0 1
carry            +    1 1 0 1 1
                 --------------
                   1  0 1 0 0 0
```

  - Discarding the carry bit, the sign bit is zero, indicating a correct result.

$$(01000)_2 = +(1000)_2 = +(8)_{10}.$$

- Let's compute $(5)_{10} – (12)_{10}$.

  $(-12)_{10} = -(1100)_2 = (10100)_2$

  $(5)_{10} = +(0101)_2 = (00101)_2$

```
        0 0 1 0 1
+       1 0 1 0 0
---------------
        1 1 0 0 1
```

  – Here, there is no carry bit and the sign bit is 1. This indicates a negative result.

    $(11001)_2 = -(7)_{10}$

Binary Codes

Binary Codes

- A binary code is just an assignment of information to bit patterns.
- A binary number is mathematically defined, while a binary code is just an assignment of numeric values to bit patterns.
- Is of 2 types:
  - Weighted code: each bit position is assigned with a weight (BCD, 2421)
  - Non weighted code- no weights associated to the bit positions (Gray Code, excess 3 code)

Binary Coded Decimal (BCD)

- formed by converting each digit of a decimal number individually into binary
- requires more digits than conventional binary
- has advantage of very easy conversion to/from decimal
- used where input and output are in decimal form

Binary Coded Decimal (BCD)

- Also known as the *8421* code.
  - Each Decimal Digit is represented by 4 bits
  - (0 – 9) ⇨ Valid combinations
  - (10 – 15) ⇨ Invalid combinations

Decimal	BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

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

- BCD is not equivalent to binary.

BCD Addition

- One decimal digit + one decimal digit
  - If the result is 1 decimal digit (≤ 9), then it is a simple binary addition
  - *Example*:

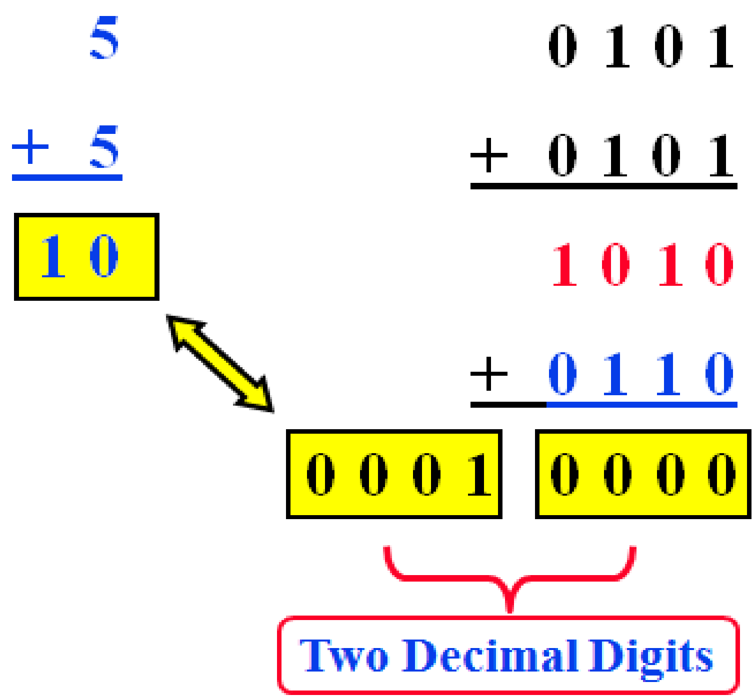
$$
\begin{array}{rcr}
5 & & 0\,1\,0\,1 \\
\underline{+\;\;3} & & \underline{+\;\;0\,0\,1\,1} \\
\boxed{8} & \Longleftrightarrow & \boxed{1\,0\,0\,0}
\end{array}
$$

– If the result is two decimal digits (≥ 10), then binary addition gives invalid combinations

Example:

	5	0 1 0 1
	+ 5	+ 0 1 0 1
$\boxed{0\,0\,0\,1}$ $\boxed{0\,0\,0\,0}$ $\Leftrightarrow$	$\boxed{1\,0}$	1 0 1 0

- If the binary result is greater than 9, correct the result by adding 6



2421 code

- Represents the decimal numbers from 0 to 9.
- Up to 4, it is same as BCD.
- It is a self complementing code

DD	8421	2421
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110
9	1001	1111

Non Weighted code

- Excess-3 code
- Self-complementing code
- Not weighted
- Corresponding BCD code + 0011_2
  - Binary counters

DD	8421	2421	Ex-3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Gray Code

- Unweighted code.
- Every transition from one value to the next value involves only one bit change.
- Also called cyclic/reflected code.
- Good for error detection.

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Alphanumeric Codes

- Apart from numbers, computers also handle textual data.
- Character set frequently used includes:
  - alphabets: 'A' .. 'Z', and 'a' .. 'z'
  - digits: '0' .. '9'
  - special symbols: '$', '.', ',', '@', '*', …
  - non-printable: SOH, NULL, BELL, …
- Two widely used standards:
  - ASCII (American Standard Code for Information Interchange)
  - EBCDIC (Extended BCD Interchange Code)

- In ASCII, each character is represented by a 7-bit code.
- EBCDIC (8 bit) was one of the first widely-used computer codes that supported upper *and* lowercase alphabetic characters, in addition to special characters, such as punctuation and control characters.

Parity

- The method of parity is widely used as a method of error detection.
  - Extar bit known as parity is added to data word
  - The new data word is then transmitted.
- Two systems are used:
  - Even parity: the number of 1's must be even.
  - Odd parity: the number of 1's must be odd.

- Example:

	Even Parity	Odd parity
11001	110011	110010
11110	111100	111101
11000	110000	110001

Overflow / Underflow

- When addition of two numbers cause the result is greater than the largest number of available bits
  - Overflow
- When addition result is smaller than the smallest number the bits can hold.
  - Underflow
- Addition of a positive and a negative number cannot give an overflow or underflow.

Overflow example

$$\begin{array}{r} 011\ (+3)_{10} \\ 011\ (+3)_{10} \\ \hline 110\ \ (+6)_{10} \end{array}$$

- 1's complement computer interprets it as –1
- $(+6)_{10} = (0110)_2$ requires four bits

Underflow examples

- Two's complement addition

$$\begin{array}{rl} & 101\ (-3)_{10} \\ & 101\ (-3)_{10} \\ \hline \text{Carry}\quad 1 & 010\ \ (-6)_{10} \end{array}$$

The computer sees it as +2.

$(-6)_{10} = (1010)_2$ again requires four bits