

Digital Logic and Design

Chapter 1

Digital Systems and Computer Systems

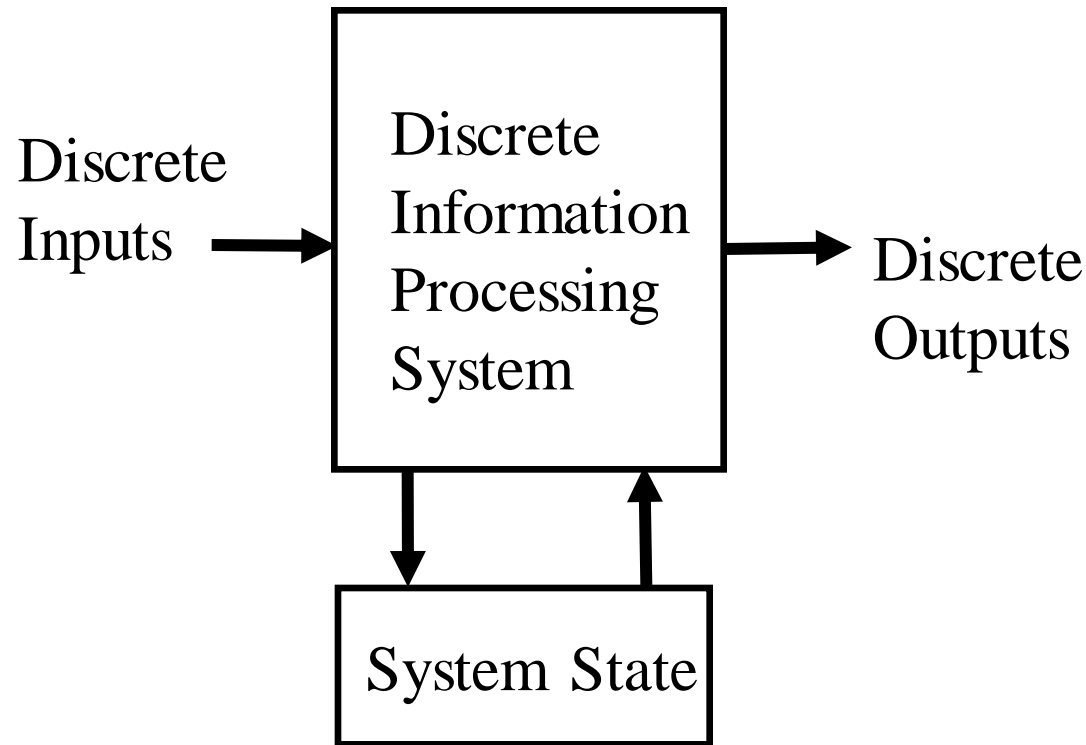
2022/2023

Overview

- **Digital Systems and Computer Systems**
- **Information Representation**
- **Number Systems** [binary, octal and hexadecimal]
- **Arithmetic Operations**
- **Base Conversion**
- **Decimal Codes**
- **Alphanumeric Codes**

Digital System

- Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.

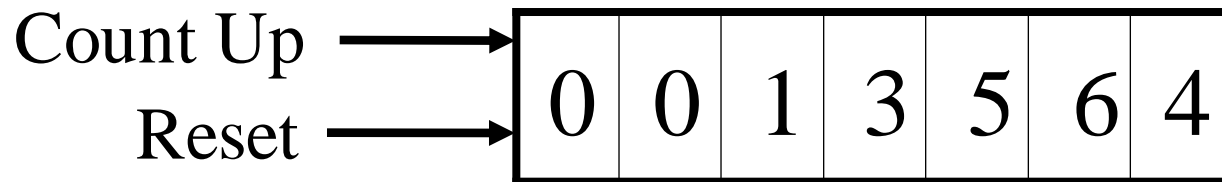


Types of Digital Systems

- **No state present**
 - **Combinational Logic System**
 - **Output = Function(Input)**
- **State present**
 - **State updated at discrete times**
=> Synchronous Sequential System
 - **State updated at any time**
=> Asynchronous Sequential System
 - **State = Function (State, Input)**
 - **Output = Function (State)**
or Function (State, Input)

Digital System Example:

A Digital Counter (e. g., odometer):



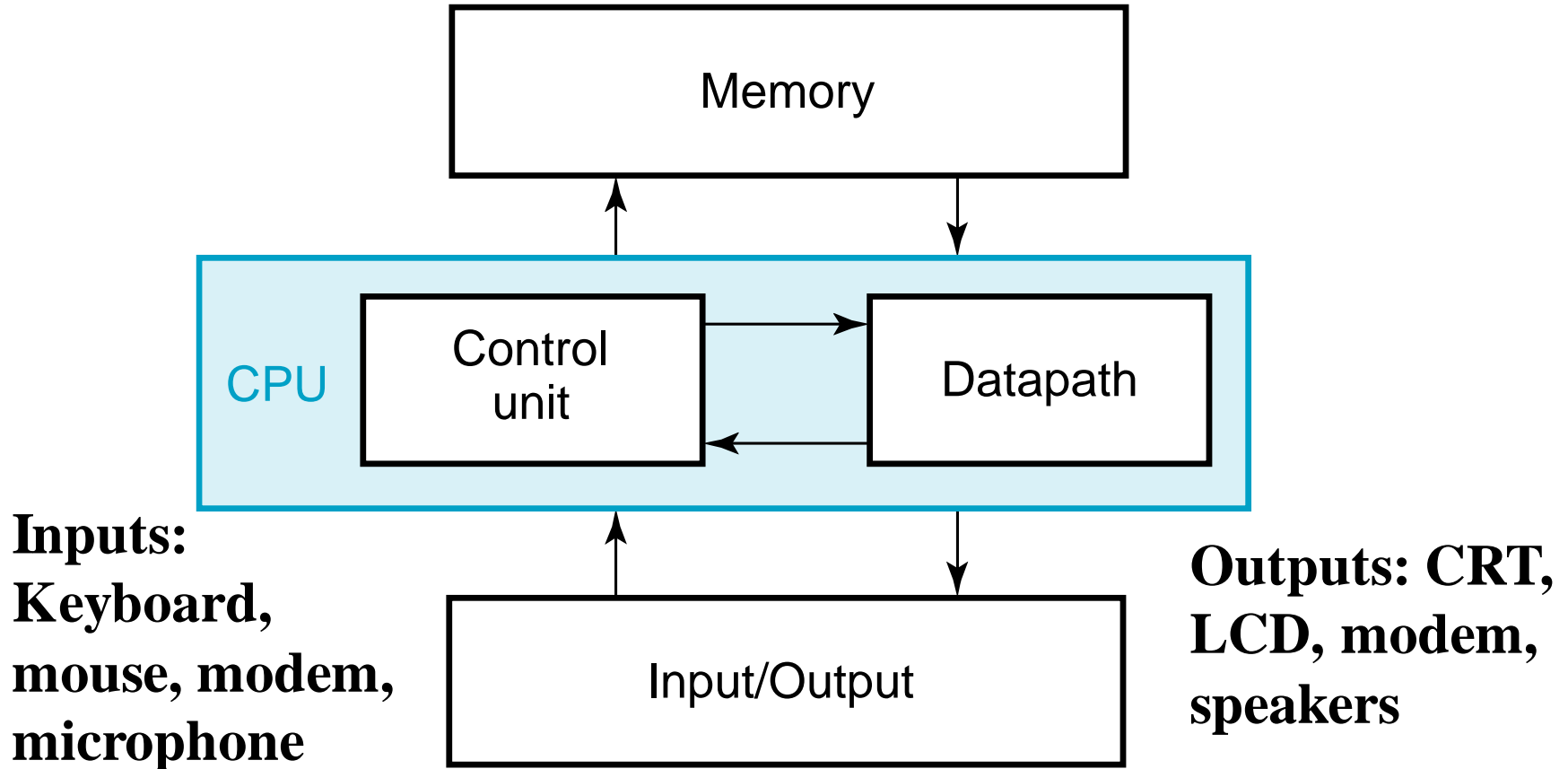
Inputs: Count Up, Reset

Outputs: Visual Display

State: "Value" of stored digits

Synchronous or Asynchronous?

A Digital Computer Example

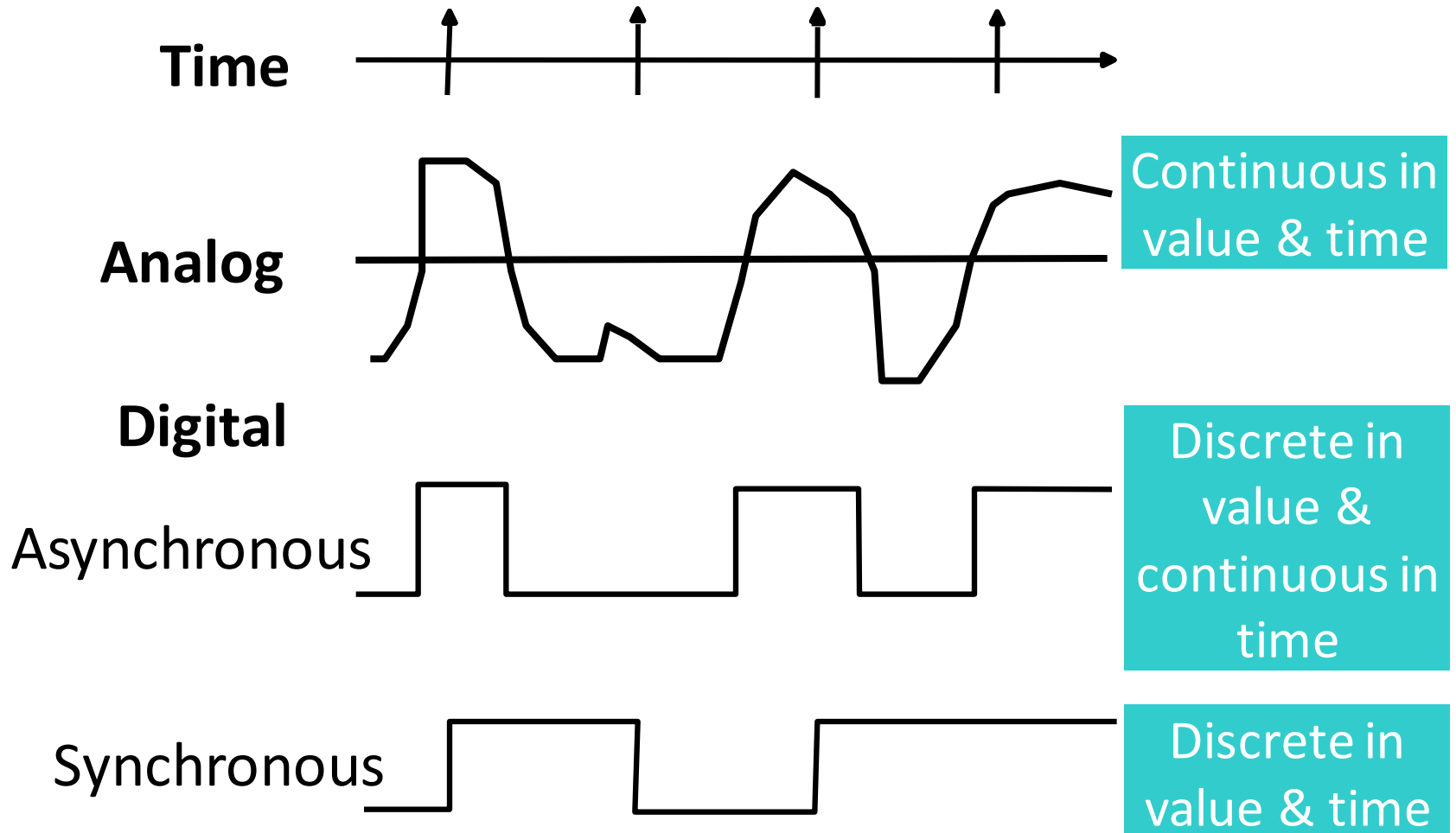


**Synchronous or
Asynchronous?**

Signal

- **An information variable represented by physical quantity.**
- **For digital systems, the variable takes on discrete values.**
- **Two level, or binary values are the most prevalent values in digital systems.**
- **Binary values are represented abstractly by:**
 - **digits 0 and 1**
 - **words (symbols) False (F) and True (T)**
 - **words (symbols) Low (L) and High (H)**
 - **and words On and Off.**
- **Binary values are represented by values or ranges of values of physical quantities**

Signal Examples Over Time



Chapter 1 : Number System

- 2.1 Decimal, Binary, Octal and Hexadecimal Numbers
- 2.2 Relation between binary number system with other number system
- 2.3 Representation of integer, character and floating point numbers in binary
- 2.4 Binary Arithmetic
- 2.5 Arithmetic Operations for One's Complement, Two's Complement, magnitude and sign and floating point number

Decimal, Binary, Octal and Hexadecimal Numbers

Most numbering system use positional notation :

$$N = a_n r^n + a_{n-1} r^{n-1} + \dots + a_1 r^1 + a_0 r^0$$

Where:

N: an integer with n+1 digits

r: base

$$a_i \in \{0, 1, 2, \dots, r-1\}$$

Examples:

a) $N = 278$

$r = 10$ (base 10) \Rightarrow decimal numbers

symbol: 0, 1, 2, 3, 4, 5, 6,
7, 8, 9 (10 different symbols)

$N = 278 \Rightarrow n = 2;$

$a_2 = 2; a_1 = 7; a_0 = 8$

$$N = a_n r^n + a_{n-1} r^{n-1} + \dots + a_1 r^1 + a_0 r^0$$

$$278 = \underbrace{(2 \times 10^2)}_{\text{Hundreds}} + \underbrace{(7 \times 10^1)}_{\text{Tens}} + \underbrace{(8 \times 10^0)}_{\text{Ones}}$$

$$N = a_n r^n + a_{n-1} r^{n-1} + \dots + a_1 r^1 + a_0 r^0$$

b) $N = 1001_2$

$r = 2$ (base-2) \Rightarrow binary numbers

symbol: 0, 1 (2 different symbols)

$N = 1001_2 \Rightarrow n = 3;$

$a_3 = 1; a_2 = 0; a_1 = 0; a_0 = 1$

$1001_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

c) $N = 263_8$

$r = 8$ (base-8) \Rightarrow Octal numbers

symbol : 0, 1, 2, 3, 4, 5, 6, 7.

(8 different symbols)

$N = 263_8 \Rightarrow n = 2; a_2 = 2; a_1 = 6; a_0 = 3$

$263_8 = (2 \times 8^2) + (6 \times 8^1) + (3 \times 8^0)$

d) $N = 263_{16}$

$r = 16$ (base-16) \Rightarrow Hexadecimal
numbers

symbol : 0, 1, 2, 3, 4, 5, 6, 7, 8,
9, A, B, C, D, E, F
(16 different symbols)

$$N = 263_{16} \Rightarrow n = 2;$$

$$a_2 = 2; a_1 = 6; a_0 = 3$$

$$263_{16} = (2 \times 16^2) + (6 \times 16^1) + (3 \times 16^0)$$

Decimal	Binary	Octal	Hexadecimal
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
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
16	10000	20	10

There are also non-positional numbering systems.

Example: Roman Number System

1987 = MCMLXXXVII

Relation between binary number system and others

Binary and Decimal

- Converting a decimal number into binary (**decimal** → **binary**)
 - ✓ **Divide** the decimal number by 2 and take its remainder
 - ✓ The process is repeated until it produces the result of 0
 - ✓ The binary number is obtained by taking the remainder **from the bottom to the top**

Example: Decimal \rightarrow Binary

$53_{10} \Rightarrow$

53	/	2	=	26	remainder	1
26	/	2	=	13	remainder	0
13	/	2	=	6	remainder	1
6	/	2	=	3	remainder	0
3	/	2	=	1	remainder	1
1	/	2	=	0	remainder	1

↑
Read from
the bottom
to the top

= 110101_2 (6 *bits*)

= 00110101_2 (8 *bits*)

(note: ***bit*** = ***binary digit***)

$0.81_{10} \rightarrow \text{binary}???$

0.81_{10}	\Rightarrow	$0.81 \times 2 = 1.62$	$\xrightarrow{\quad}$	0.110011_2
		$0.62 \times 2 = 1.24$	$\xrightarrow{\quad}$	
		$0.24 \times 2 = 0.48$	$\xrightarrow{\quad}$	
		$0.48 \times 2 = 0.96$	$\xrightarrow{\quad}$	
		$0.96 \times 2 = 1.92$	$\xrightarrow{\quad}$	
		$0.92 \times 2 = 1.84$	$\xrightarrow{\quad}$	

$= 0.110011_2 \text{ (approximately)}$

Converting a binary number into decimal

(**binary** → **decimal**)

Multiply each bit in the binary number
with the weight (or position)

Add up all the results of the
multiplication performed

The desired decimal number is the
total of the multiplication results
performed

Example: Binary \rightarrow Decimal

a) 111001_2 (6 bits)

$$\Rightarrow (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

$$= 32 + 16 + 8 + 0 + 0 + 1$$

$$= 57_{10}$$

b) 00011010_2 (8 bits)

$$= 2^4 + 2^3 + 2^1$$

$$= 16 + 8 + 2$$

$$= 26_{10}$$

Binary and Octal

Theorem

If base R_1 is the integer power of other base, R_2 , i.e.

$$R_1 = R_2^d$$

e.g., $8 = 2^3$

Every group of **d** digits in R_2
(e.g., 3 digits) is equivalent to **1**
digit in the R_1 base

(Note: This theorem is used to convert
binary numbers to octal and hexadecimal
or the other way round)

- From the theorem, assume that
$$R_1 = 8 \text{ (base-8) octal}$$
$$R_2 = 2 \text{ (base-2) binary}$$
- From the theorem above,

$$R_1 = R_2^d$$
$$8 = 2^3$$

So, **3 digits in base-2 (binary) is equivalent to 1 digit in base-8 (octal)**

- From the stated theorem, the following is a binary-octal conversion table.

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

In a computer system, the conversion from binary to octal or otherwise is based on the conversion table above.

3 digits in base-2 (binary) is equivalent to 1 digit in base-8 (octal)

Example: Binary \rightarrow Octal

Convert these binary numbers into octal numbers:

(a) 00101111_2 (8 bits)

(b) 11110100_2 (8 bits)

Refer to the binary-octal conversion table

$\underbrace{000}_0 \underbrace{101}_5 \underbrace{111}_7$

$= 57_8$

Refer to the binary-octal conversion table

$\underbrace{011}_3 \underbrace{110}_6 \underbrace{100}_4$

$= 364_8$

Binary and Hexadecimal

- The same method employed in binary-octal conversion is used once again.
- Assume that:
$$R_1 = 16 \text{ (hexadecimal)}$$
$$R_2 = 2 \text{ (binary)}$$
- From the theorem: $16 = 2^4$

Hence, **4 digits in a binary number** is equivalent to **1 digit in the hexadecimal** number system (and otherwise)
- The following is the binary-hexadecimal conversion table

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

Example:

1. Convert the following binary numbers into hexadecimal numbers:

(a) 00101111_2

Refer to the binary-hexadecimal conversion table above

$$\begin{array}{ccc}
 0010 & 1111_2 & = 2F_{16} \\
 2 & F &
 \end{array}$$

Example: Octal \rightarrow Hexadecimal

Convert the following octal numbers into hexadecimal numbers (16 bits)

(a) 65_8

(b) 123_8

Refer to the binary-octal conversion table

6_8	5_8
110	101
0000	0000
0011	0101
0	3
0	5

$= 35_{16}$

Refer to the binary-octal conversion table

1_8	2_8	3_8
001	010	011
0000	0000	0101
0	0	5
		3

$= 53_{16}$

octal \rightarrow binary \rightarrow hexadecimal

Example: Hexadecimal \rightarrow Binary

Convert the following hexadecimal numbers into binary numbers

(a) $12B_{16}$

(b) $ABCD_{16}$

Refer to the binary-hexadecimal conversion table

$$\begin{array}{ccccc} 1 & & 2 & & B_{16} \\ \vdots & & \vdots & & \\ 0001 & 0010 & 1011 & _2 & (12 \text{ bits}) \\ = 000100101011_2 \end{array}$$

Refer to the binary-hexadecimal conversion table

$$\begin{array}{ccccccc} A & & B & & C & & D_{16} \\ \vdots & & \vdots & & \vdots & & \vdots \\ 1010 & 1011 & 1101 & 1110 & _2 \\ = 1010101111011110_2 \end{array}$$

Quze1 :

(5 marks)

Q1: difference between Ram and Rom is:

1. Two is writable
2. Two are Erasable
3. Both 1,2
4. No one

Q2 convert:

- Binary \rightarrow decimal
 - 110.011
- Decimal \rightarrow binary
 - 3.81
- Oct to hex of 5655₈
- Q3 find in 8 bits
 - 53 using sign-mang and one's compl.
 - And then 2's compl.

Solution 1

- Binary \rightarrow decimal

- $001100 = 13 \leftarrow 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 4 = 12$

- $11100.011 = 28.375$

- Decimal \rightarrow binary

- $145 = 10010001 \leftarrow 145/2 = 72 \text{ (remainder 1)}; 72/2=36(r=0); 36/2=18(r=0);$

- $34.75 = 100010.11 \leftarrow 18/2=9(r=0); 9/2=4(r=1); 4/2=2(r=0); 2/2=1(r=0); 1/2=0(r=1)$

- Octal \rightarrow hexadecimal

octal \rightarrow binary \rightarrow decimal \rightarrow hexadecimal


- $5655_8 = \text{BAD} \leftarrow \begin{array}{l} \text{Octal} \rightarrow \text{binary} \\ 101:110:101:101 \\ \text{Binary} \rightarrow \text{hexadecimal} \\ 1011:1010:1101 \\ \text{B} \quad \text{A} \quad \text{D} \end{array}$

Solution 1

- Binary \rightarrow decimal

- 001100 = 12


- 11100.011 = 28.375


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

- Decimal \rightarrow binary

- 145 = 10010001

- 34.75 = 100010.11


$$\begin{array}{l} 0.75 \times 2 = 1.5 \downarrow \\ 0.5 \times 2 = 1.0 \end{array}$$

- Octal \rightarrow hexadecimal

octal \rightarrow binary \rightarrow hexadecimal

- $5655_8 = \text{BAD}$

Exercise 2

- Binary \rightarrow decimal
 - 110011.10011 \leftarrow 51.59375
- Decimal \rightarrow binary
 - 25.25 \leftarrow 11001.01
- Octal \rightarrow hexadecimal
 - 12_8 \leftarrow B

- End

Representation of integer, character and floating point numbers in binary

Introduction

Machine instructions operate on data. The most important general categories of data are:

1. **Addresses** – unsigned integer
2. **Numbers** – integer or fixed point, floating point numbers and decimal (eg, BCD (*Binary Coded Decimal*))
3. **Characters** – IRA (International Reference Alphabet), EBCDIC (Extended Binary Coded Decimal Interchange Code), ASCII (American Standard Code for Information Interchange)
4. **Logical Data**
 - Those commonly used by computer users/programmers: signed integer, floating point numbers and characters

Integer Representation

- $-1101.0101_2 = -13.3125_{10}$
- Computer storage & processing → do not have benefit of minus signs (-) and periods.
 - ↳ Need to represent the integer →

Signed Integer Representation

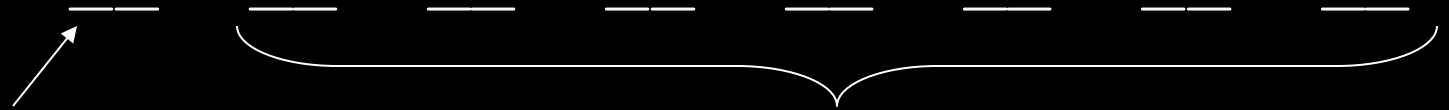
- Signed integers are usually used by programmers
- Unsigned integers are used for addressing purposes in the computer (especially for assembly language programmers)
- Three representations of signed integers:
 1. Sign-and-Magnitude
 2. Ones Complement
 3. Twos Complement

Sign-and-Magnitude

- The easiest representation
- The leftmost bit in the binary number represents the sign of the number. **0** if positive and **1** if negative
- The balance bits represent the magnitude of the number.

Examples:

i) 8 bits binary number



Sign bit 7 bits for magnitude (value)

0 => +ve 1 => -ve

$$\begin{aligned} \text{a) } +7 &= \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{1}} \ \underline{\mathbf{1}} \ \underline{\mathbf{1}} \\ (-7 &= \mathbf{1}0000111_2) \end{aligned}$$

$$\begin{aligned} \text{b) } -10 &= \underline{\mathbf{1}} \ \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{0}} \ \underline{\mathbf{1}} \ \underline{\mathbf{0}} \ \underline{\mathbf{1}} \ \underline{\mathbf{0}} \\ (+10 &= \mathbf{0}0001010_2) \end{aligned}$$

ii) 6 bits binary number



$$\begin{aligned} \text{a)} \quad +7 &= \underline{\mathbf{0}} \underline{0} \underline{0} \underline{1} \underline{1} \underline{1} \\ &(-7 = \underline{\mathbf{1}} \underline{0} \underline{0} \underline{1} \underline{1} \underline{1}_2) \end{aligned}$$

$$\begin{aligned} \text{b)} \quad -10 &= \underline{\mathbf{1}} \underline{0} \underline{1} \underline{0} \underline{1} \underline{0} \\ &(+10 = \underline{\mathbf{0}} \underline{0} \underline{1} \underline{0} \underline{1} \underline{0}_2) \end{aligned}$$

Ones Complement

- In the ones complement representation, **positive numbers are same as that of sign-and-magnitude**

Example: $+5 = 00000101$ (8 bit)

⇒ as in sign-and-magnitude representation

- Sign-and-magnitude and ones complement use the same representation above for +5 with 8 bits and all positive numbers.
- For negative numbers, their representation are obtained by changing *bit 0 → 1 and 1 → 0* from their positive numbers

Example:

Convert -5 into ones complement representation (8 bit)

Solution:

- First, obtain $+5$ representation in 8 bits $\Rightarrow 00000101$
- Change every bit in the number from 0 to 1 and vice-versa.
- -5_{10} in ones complement is 11111010_2

Exercise:

Get the representation of ones complement (6 bit) for the following numbers:

$$i) +7_{10}$$

$$ii) -10_{10}$$

Solution:

$$(+7) = 000111_2$$

$$(-7) = 111000$$

Solution:

$$(+10)_{10} = 001010_2$$

So,

$$(-10)_{10} = 110101_2$$

Twos complement

- Similar to ones complement, its **positive number is same as sign-and-magnitude**
- Representation of its **negative number** is obtained by **adding 1** to **the ones complement of the number.**

Example:

Convert -5 into twos complement representation and give the answer in 8 bits.

Solution:

- ✓ First, obtain +5 representation in 8 bits $\Rightarrow 00000101_2$
- ✓ Obtain ones complement for -5
 $\Rightarrow 11111010_2$
- ✓ Add 1 to the ones complement number:
 $\Rightarrow 11111010_2 + 1_2 = 11111011_2$
- ✓ -5 in twos complement is 11111011_2

Exercise:

- Obtain representation of twos complement (6 bit) for the following numbers

i) $+7_{10}$

Solution:

$$(+7) = 000111_2$$

(same as sign-magnitude)

ii) -10_{10}

Solution:

$$(+10)_{10} = 001010_2$$

$$\begin{aligned} (-10)_{10} &= 110101_2 + 1_2 \\ &= 110110_2 \end{aligned}$$

So, twos complement
for -10 is 110110_2

Exercise:

Obtain representation for the following numbers in 8 bits

Decimal	Sign-magnitude	Twos complement
+7		
+6		
-4		
-6		
-7		
+18		
-18		
-13		

Solution:

Obtain representation for the following numbers

Decimal	Sign-magnitude	Twos complement
+7	0111	0111
+6	0110	0110
-4	1100	1100
-6	1110	1010
-7	1111	1001
+18	00010010	00010010
-18	10010010	11101110
-13	11110010	11110011

Character Representation

- For character data type, its representation uses codes such as the **ASCII**, **IRA** or **EBCDIC**.

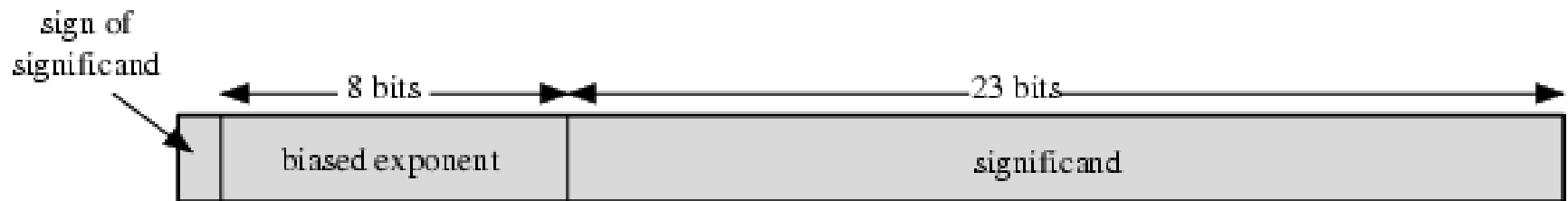
Note: Students are encouraged to obtain the codes

Floating point representation

- In binary, floating point numbers are represented in the form of : $\pm S \times B^{\pm E}$ and the number can be stored in computer words with 3 fields:

- i) Sign (+ve, -ve)
- ii) Significant S
- iii) Exponent E

and B is base is implicit and need not be stored because it is the same for all numbers (base-2) .



(a) Format

$$\begin{aligned}
 1.1010001 \times 2^{10100} &= 0 \ 10010011 \ 101000100000000000000000 = 1.638125 \times 2^{20} \\
 -1.1010001 \times 2^{10100} &= 1 \ 10010011 \ 101000100000000000000000 = -1.638125 \times 2^{20} \\
 1.1010001 \times 2^{-10100} &= 0 \ 01101011 \ 101000100000000000000000 = 1.638125 \times 2^{-20} \\
 -1.1010001 \times 2^{-10100} &= 1 \ 01101011 \ 101000100000000000000000 = -1.638125 \times 2^{-20}
 \end{aligned}$$

(b) Examples

Binary Arithmetic's

1. Addition (+)

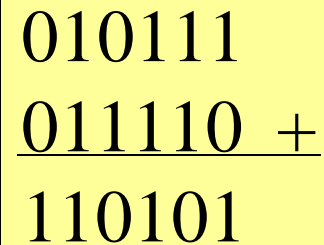
$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

$$1 + 1 + 1 = (1 + 1) + 1 = 10 + 1 = 11_2$$


$$\begin{array}{r} 010111 \\ \underline{011110} + \\ 110101 \end{array}$$

Example:

i. $010111_2 + 011110_2 = 110101_2$

ii. $100011_2 + 011100_2 = 111111_2$

2. **Multiplication** (\times)

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

3. **Subtraction** ($-$)

$$0 - 0 = 0$$

$$0 - 1 = 1 \quad (\text{borrow } 1)$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

4. **Division** (/)

$$0 / 1 = 0$$

$$1 / 1 = 1$$

Example:

$$\text{i. } 010111_2 - 001110_2 = 001001_2$$

$$\text{ii. } 100011_2 - 011100_2 = 000111_2$$

Exercise:

$$\text{i. } 1000100 - 010010$$

$$\text{v. } 110111 + 001101$$

$$\text{ii. } 1010100 + 1100$$

$$\text{vi. } 111000 + 1100110$$

$$\text{iii. } 110100 - 1001$$

$$\text{vii. } 110100 \times 10$$

$$\text{iv. } 11001 \times 11$$

$$\text{viii. } 11001 - 1110$$

Arithmetic Operations for Ones Complement, Twos Complement, sign-and-magnitude and floating point number

Addition and subtraction for signed integers

Reminder: All subtraction operations will be **changed into addition** operations

Example:

$$8 - 5 = 8 + (-5)$$
$$-10 + 2 = (-10) + 2$$
$$6 - (-3) = 6 + 3$$

Sign-and-Magnitude

$$Z = X + Y$$

There are a few possibilities:

i. **If both numbers, X and Y are positive**

- Just perform the addition operation

Example:

$$\begin{aligned} 5_{10} + 3_{10} &= 000101_2 + 000011_2 \\ &= 001000_2 \\ &= 8_{10} \end{aligned}$$

ii. If both numbers are negative

- Add $|X|$ and $|Y|$ and set the sign bit = 1 to the result, Z

$$\begin{aligned}\text{Example: } -3_{10} - 4_{10} &= (-3) + (-4) \\ &= \underline{100011}_2 + \underline{100100}_2\end{aligned}$$

Only add the magnitude, i.e.:

$$00011_2 + 00100_2 = 00111_2$$

Set the sign bit of the result (Z) to 1 (-ve)

$$\begin{aligned}&= \underline{1}00111_2 \\ &= -7_{10}\end{aligned}$$

iii. If signs of **both** number **differ**

- There will be 2 cases:

- a) $| +ve \text{ Number} | > | -ve \text{ Number} |$

Example: $(-2) + (+4)$, $(+5) + (-3)$

- Set the sign bit of the -ve number to 0 (+ve), so that both numbers become +ve.
 - Subtract the number of smaller magnitude from the number with a bigger magnitude

Sample solution:

Change the sign bit of the -ve number to +ve

$$\begin{aligned}(-2) + (+4) &= \underline{100010}_2 + 000100_2 \\ &= 000100_2 - \underline{000010}_2 \\ &= 000010_2 = 2_{10}\end{aligned}$$

b) | -ve Number | > | +ve Number |

- Subtract the +ve number from the -ve number

$$\begin{aligned}\text{Example: } (+3_{10}) + (-5_{10}) & \\ &= 000011_2 + 100101_2 \\ &= \underline{100101}_2 - 000011_2 \\ &= 100010_2 \\ &= -2_{10}\end{aligned}$$

Ones complement

- In ones complement, it is easier than sign-and-magnitude
- Change the numbers to its representation and perform the addition operation
- However a situation called **Overflow** might occur when addition is performed on the following categories:

1. If both are negative numbers
2. If both are in difference sign and $|+ve\ Number| > |-ve\ Number|$

Overflow => the addition result exceeds the number of bits that was fixed

1. Both are -ve numbers

Example: $-3_{10} - 4_{10} = (-3_{10}) + (-4_{10})$

Solution:

-Convert -3_{10} and -4_{10} into ones complement representation

$$+3_{10} = 00000011_2 \quad (8 \text{ bits})$$

$$-3_{10} = 11111100_2$$

$$+4_{10} = 00000100_2 \quad (8 \text{ bits})$$


$$-4_{10} = 11111011_2$$

- Perform the addition operation

$$\begin{array}{rcl}
 (-3_{10}) & \Rightarrow & 11111100 \text{ (8 bit)} \\
 + (-4_{10}) & \Rightarrow & \underline{11111011 \text{ (8 bit)}} \\
 \hline
 -7_{10} & & \textcolor{red}{1}11110111 \text{ (9 bit)}
 \end{array}$$

Overflow occurs. This value is called EAC and needs to be added to the rightmost bit.

$$\begin{array}{r}
 11110111 \\
 + \quad \quad \quad 1 \\
 \hline
 1\mathbf{1111000}_2
 \end{array}
 = -7_{10}$$


the answer

2. $|+ve\ Number| > |-ve\ Number|$

- This case will also cause an *overflow*

Example: $(-2) + 4 = (-2) + (+4)$

Solution:

- Change both of the numbers above into one's complement representation

$$-2 = 11111101_2 \quad +4 = 00000100_2$$


- Add both of the numbers

$$\begin{array}{rcl} (-2_{10}) & \Rightarrow & 11111101 \quad (8\ bit) \\ + \quad (+4_{10}) & \Rightarrow & 00000100 \quad (8\ bit) \\ \hline & & 100000001 \quad (9\ bit) \\ & & +2_{10} \end{array}$$

There is an EAC

- Add the EAC to the rightmost bit

$$\begin{array}{r}
 00000001 \\
 + \quad \quad \quad 1 \\
 \hline
 00000010_2 = +2_{10}
 \end{array}$$



 the answer

Note:

For cases other than 1 & 2 above, *overflow* does not occur and there will be no EAC and the need to perform addition to the rightmost bit does not arise

Twos Complement

Addition operation in twos complement is same with that of ones complement, i.e. *overflow* occurs if:

1. If both are negative numbers
2. If both are in difference and $|+ve\ Number| > |-ve\ Number|$

Both numbers are -ve

Example: $-3_{10} - 4_{10} = (-3_{10}) + (-4_{10})$

Solution:

- Convert both numbers into twos complement representation

$$+3_{10} = 000011_2 \text{ (6 bit)}$$

$$-3_{10} = 111100_2 \text{ (one's complement)}$$

$$-3_{10} = 111101_2 \text{ (two's complement)}$$

$$-4_{10} = 111011_2 \text{ (one's complement)}$$

$$-4_{10} = 111100_2 \text{ (two's complement)}$$

- Perform addition operation on both the numbers in twos complement representation and ignore the EAC.

$$\begin{array}{r}
 111100 \ (-3_{10}) \\
 \underline{111011 \ (-4_{10})} \\
 1110111
 \end{array}$$

Ignore the
EAC

↑ The answer

$$\begin{aligned}
 &= 111000_2 \text{ (two's complement)} \\
 &= -7_{10}
 \end{aligned}$$

Note:

In two's complement, EAC is ignored (**do not need** to be added to the leftmost bit, like that of one's complement)

2. $|+ve\ Number| > |-ve\ Number|$


Example: $(-2) + 4 = (-2) + (+4)$

Solution:

- Change both of the numbers above into twos complement representation

$$-2 = 111110_2 \qquad +4 = 000100_2$$

- Perform addition operation on both numbers

$$\begin{array}{rcl} & (-2_{10}) & \Rightarrow 111110 \text{ (6 bit)} \\ + & (+4_{10}) & \Rightarrow 000100 \text{ (6 bit)} \\ \hline & +2_{10} & \quad \quad \quad 1000010 \end{array}$$


Ignore the EAC

The answer is $000010_2 = +2_{10}$

Note: For cases other than 1 and 2 above, overflow does not occur.

Exercise:

Perform the following arithmetic operations in ones complement and also twos complement

1. $(+2) + (+3)$ [6 bit]
2. $(-2) + (-3)$ [6 bit]
3. $(-2) + (+3)$ [6 bit]
4. $(+2) + (-3)$ [6 bit]

Compare your answers with the stated theory

- 5
- 101
- 000101 +5
- 111010 -5 (1's)
- 111011 -5(2's)