Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr
naydin@ieee.org
http://www.yildiz.edu.tr/~naydin

1

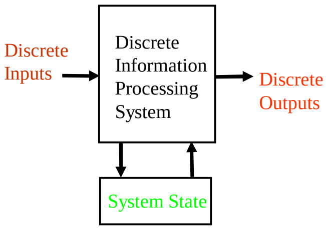
Information Systems:

Fundamentals

2

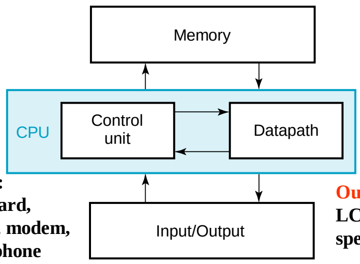
Digital System

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



3

A Digital Computer Example



Inputs: Keyboard, mouse, modem, microphone

Outputs: CRT, LCD, modem, speakers

Synchronous or Asynchronous?

4

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**

5

Measures of capacity and speed

Special Powers of 10 and 2 :

- Kilo- (K) = 1 thousand = 10^3 and 2^{10}
- Mega- (M) = 1 million = 10^6 and 2^{20}
- Giga- (G) = 1 billion = 10^9 and 2^{30}
- Tera- (T) = 1 trillion = 10^{12} and 2^{40}
- Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}

Whether a metric refers to a power of ten or a power of two typically depends upon what is being measured.

6

Example

- Hertz = clock cycles per second (frequency)
  - 1MHz = 1,000,000Hz
  - Processor speeds are measured in MHz or GHz.
- Byte = a unit of storage
  - 1KB = 2^{10} = 1024 Bytes
  - 1MB = 2^{20} = 1,048,576 Bytes
  - Main memory (RAM) is measured in MB
  - Disk storage is measured in GB for small systems, TB for large systems.

7

Measures of time and space

- Milli- (m) = 1 thousandth = 10^{-3}
- Micro- (μ) = 1 millionth = 10^{-6}
- Nano- (n) = 1 billionth = 10^{-9}
- Pico- (p) = 1 trillionth = 10^{-12}
- Femto- (f) = 1 quadrillionth = 10^{-15}

8

Data types

- Our first requirement is to find a way to represent information (data) in a form that is mutually comprehensible by human and machine.
  - Ultimately, we will have to develop schemes for representing all conceivable types of information - language, images, actions, etc.
  - We will start by examining different ways of representing integers, and look for a form that suits the computer.
  - Specifically, the devices that make up a computer are switches that can be on or off, i.e. at high or low voltage.
  - Thus they naturally provide us with two symbols to work with:
    - we can call them on and off, or 0 and 1.

9

What kinds of data do we need to represent?

Numbers
signed, unsigned, integers, floating point, complex, rational, irrational, …
Text
characters, strings, …
Images
pixels, colors, shapes, …
Sound
Logical
true, false
Instructions
…
Data type:
- representation and operations within the computer

10

Number Systems – Representation

- Positive radix, positional number systems
- A number with *radix* r is represented by a string of digits:

$$A_{n-1}A_{n-2}\ldots A_1A_0\,.\,A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$$

in which $0 \leq A_i < r$ and $.$ is the *radix point*.
- The string of digits represents the power series:

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j \cdot r^j\right)$$

(Integer Portion) + (Fraction Portion)

11

Decimal Numbers

- "decimal" means that we have ten digits to use in our representation
  - the symbols 0 through 9
- What is 3546?
  - it is *three* thousands plus *five* hundreds plus *four* tens plus *six* ones.
  - i.e. $3546 = 3\times10^3 + 5\times10^2 + 4\times10^1 + 6\times10^0$
- How about negative numbers?
  - we use two more symbols to distinguish positive and negative:
    + and -

12

Decimal Numbers

- "decimal" means that we have ten digits to use in our representation (the symbols 0 through 9)
- What is 3546?
  - it is *three* thousands plus *five* hundreds plus *four* tens plus *six* ones.
  - i.e. $3546 = 3.10^3 + 5.10^2 + 4.10^1 + 6.10^0$
- How about negative numbers?
  - we use two more symbols to distinguish positive and negative:

    + and -

13

Unsigned Binary Integers

$Y = \text{"abc"} = a.2^2 + b.2^1 + c.2^0$

(where the digits a, b, c can each take on the values of 0 or 1 only)

N = number of bits

Range is: $0 \leq i < 2^N - 1$

	3-bits	5-bits	8-bits
0	000	00000	00000000
1	001	00001	00000001
2	010	00010	00000010
3	011	00011	00000011
4	100	00100	00000100

Problem:
- How do we represent *negative* numbers?

14

Two's Complement

- Transformation
  - To transform a into -a, invert all bits in a and add 1 to the result

Range is: $-2^{N-1} < i < 2^{N-1} - 1$

Advantages:
- Operations need not check the sign
- Only one representation for zero
- Efficient use of all the bits

-16	10000
…	…
-3	11101
-2	11110
-1	11111
0	00000
+1	00001
+2	00010
+3	00011
…	…
+15	01111

15

Limitations of integer representations

- Most numbers are not integer!
  - Even with integers, there are two other considerations:
- Range:
  - The magnitude of the numbers we can represent is determined by how many bits we use:
    - e.g. with 32 bits the largest number we can represent is about +/- 2 billion, far too small for many purposes.
- Precision:
  - The exactness with which we can specify a number:
    - e.g. a 32 bit number gives us 31 bits of precision, or roughly 9 figure precision in decimal repesentation.
- We need another data type!

16

Real numbers

- Our decimal system handles non-integer *real* numbers by adding yet another symbol - the decimal point (.) to make a *fixed point* notation:
  - e.g. $3456.78 = 3.10^3 + 4.10^2 + 5.10^1 + 6.10^0 + 7.10^{-1} + 8.10^{-2}$
- The *floating point*, or scientific, notation allows us to represent very large and very small numbers (integer or real), with as much or as little precision as needed:
  - Unit of electric charge $e = 1.602\ 176\ 462 \times 10^{-19}$ Coulomb
  - Volume of universe = 1×10^{85} cm^3
    - the two components of these numbers are called the mantissa and the exponent

17

Real numbers in binary

- We mimic the decimal floating point notation to create a "hybrid" binary floating point number:
  - We first use a "binary point" to separate whole numbers from fractional numbers to make a fixed point notation:
    - e.g. $00011001.110 = 1.2^4 + 1.2^3 + 1.2^1 + 1.2^{-1} + 1.2^{-2}$ => 25.75

      ($2^{-1} = 0.5$ and $2^{-2} = 0.25$, etc.)
  - We then "float" the binary point:
    - 00011001.110 => 1.1001110×2^4

      mantissa = 1.1001110, exponent = 4
  - Now we have to express this without the extra symbols (x, 2, .)
    - by convention, we divide the available bits into three fields:

      sign, mantissa, exponent

18

IEEE-754 fp numbers - 1

s	biased exp.	fraction
32 bits: 1	8 bits	23 bits

$N = (-1)^s \times 1.\text{fraction} \times 2^{(\text{biased exp.} - 127)}$

- Sign: 1 bit
- Mantissa: 23 bits
  - We "normalize" the mantissa by dropping the leading 1 and recording only its fractional part (why?)
- Exponent: 8 bits
  - In order to handle both +ve and -ve exponents, we add 127 to the actual exponent to create a "biased exponent":
    - 2^{-127} => biased exponent = 0000 0000 (= 0)
    - 2^{0} => biased exponent = 0111 1111 (= 127)
    - 2^{+127} => biased exponent = 1111 1110 (= 254)

IEEE-754 fp numbers - 2

- Example: Find the corresponding fp representation of 25.75
  - 25.75 => 00011001.110 => 1.1001110 x 2^4
  - sign bit = 0 (+ve)
  - normalized mantissa (fraction) = 100 1110 0000 0000 0000 0000
  - biased exponent = 4 + 127 = 131 => 1000 0011
  - so 25.75 => 0 1000 0011 100 1110 0000 0000 0000 0000 => x41CE0000
- Values represented by convention:
  - Infinity (+ and -): exponent = 255 (1111 1111) and fraction = 0
  - NaN (not a number): exponent = 255 and fraction ≠ 0
  - Zero (0): exponent = 0 and fraction = 0
    - note: exponent = 0 => fraction is *de-normalized*, i.e no hidden 1

IEEE-754 fp numbers - 3

- Double precision (64 bit) floating point

s	biased exp.	fraction
64 bits: 1	11 bits	52 bits

$N = (-1)^s \times 1.\text{fraction} \times 2^{(\text{biased exp.} - 1023)}$

- Range & Precision:
  - 32 bit:
    - mantissa of 23 bits + 1 => approx. 7 digits decimal
    - $2^{+/-127}$ => approx. $10^{+/-38}$
  - 64 bit:
    - mantissa of 52 bits + 1 => approx. 15 digits decimal
    - $2^{+/-1023}$ => approx. $10^{+/-306}$

Binary Numbers and Binary Coding

- **Flexibility of representation**
  - **Within constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.**
- **Information Types**
  - **Numeric**
    - **Must represent range of data needed**
    - **Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted**
    - **Tight relation to binary numbers**
  - **Non-numeric**
    - **Greater flexibility since arithmetic operations not applied.**
    - **Not tied to binary numbers**

Non-numeric Binary Codes

- **Given *n* binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.**
- **Example: A binary code for the seven colors of the rainbow**
- **Code 100 is not used**

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

Number of Bits Required

- **Given M elements to be represented by a binary code, the minimum number of bits, *n*, needed, satisfies the following relationships:**
  - $2^n > M > 2^{(n-1)}$
  - $n = \lceil \log_2 M \rceil$ **where** $\lceil x \rceil$, **called the *ceiling function*, is the integer greater than or equal to *x*.**
- **Example: How many bits are required to represent decimal digits with a binary code?**
  - **4 bits are required** ($n = \lceil \log_2 9 \rceil = 4$)

Number of Elements Represented

- **Given *n* digits in radix *r*, there are r^n distinct elements that can be represented.**
- **But, you can represent *m* elements, $m < r^n$**
- **Examples:**
  - **You can represent 4 elements in radix $r = 2$ with $n = 2$ digits: (00, 01, 10, 11).**
  - **You can represent 4 elements in radix $r = 2$ with $n = 4$ digits: (0001, 0010, 0100, 1000).**

25