# Computer Science 1000: Part #3

## Binary Numbers

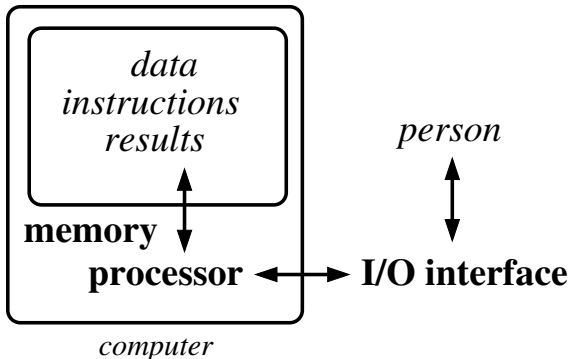COMPUTER ORGANIZATION: AN OVERVIEW

AN HISTORICAL INTERLUDE

REPRESENTING NUMBERS IN BINARY

REPRESENTING TEXT, SOUND, AND
PICTURES IN BINARY

# Computer Organization: An Overview
## The Von Neumann Architecture (1945)



Also known as the stored-program architecture

A computer is a machine that

(1) stores a very, very large number of numbers and

(2) performs very, very long specified sequences of very simple operations on these numbers

(3) very, very fast.

# Computer Organization: An Overview (Cont'd)
## Guiding Principles

There are two main principles of computer organization:

1. **Levels of Abstraction**: A complex system can be
   described as a hierarchy of levels, where collections of
   interacting entities at one level are encapsulated in a single
   entity at a higher level (see Textbook, Figure 5.1, p. 223).

2. **Internal vs. External Representation**: Within a complex
   system, the representations used internally by an entity to
   perform its functions need not be those with which it
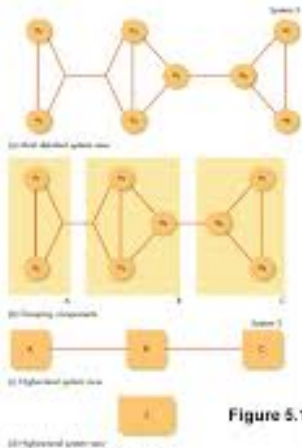   interacts externally with other entities.

**Figure 5.1** The Concept of Abstraction

# A Historical Interlude:
## The Glory of Decimal Numbers

- Decimal numbers (**base-10 positional notation**) were developed around 1200 years ago in the Middle East.
- Rapidly replaced older non-positional systems, e.g.,

MCMXXXVII
$$(1000 + (1000 - 100) + 10 + 10 + 10 + 5 + 1 + 1)$$

vs.

$1937_{10}$
$$((1 \times 10^3) + (9 \times 10^2) + (3 \times 10^1) + (7 \times 10^0))$$

## A Historical Interlude: (Cont'd)
## The Glory of Decimal Numbers (Cont'd)

| Position | Decimal |
|----------|---------|
| 1 | $10^0 \to 1$ |
| 2 | $10^1 \to 10$ |
| 3 | $10^2 \to 100$ |
| 4 | $10^3 \to 1000$ |
| 5 | $10^4 \to 10,000$ |
| 6 | $10^5 \to 100,000$ |
| 7 | $10^6 \to 1,000,00$ |
| 8 | $10^7 \to 10,000,000$ |
| 9 | $10^8 \to 100,000,000$ |
| 10 | $10^9 \to 1,000,000,000$ |

# A Historical Interlude: (Cont'd)
## The Tyranny of Decimal Numbers

Centrality of decimal numbers in mathematics led to their being both the internal and external representations of numbers in the earliest computing machines, e.g., the 1642 addition machine of Blaise Pascal (1623–1662).
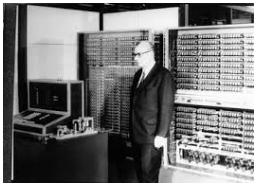
# A Historical Interlude: (Cont'd)
## The Tyranny of Decimal Numbers (Cont'd)

This reliance on decimal internal representations continued into the early days of electromechanical computing.



Harvard Mark I
(1944)



Zuse Z3
(1941)



Colossus
(1944)

## A Historical Interlude: (Cont'd)
## The Glory of Binary Numbers

The emergence of fully electronic computing in the late 1940s led to binary (**base-2 positional notation**) internal representations of numbers.

$$167_{10}$$
$$((1 \times 10^2) + (6 \times 10^1) + (7 \times 10^0))$$

vs.

$$10100111_2$$
$$((1 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) +$$
$$(0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0))$$

# A Historical Interlude: (Cont'd)
## The Glory of Binary Numbers (Cont'd)

| Position | Decimal | Binary |
|----------|---------|--------|
| 1  | $10^0 \rightarrow 1$ | $2^0 \rightarrow 1$ |
| 2  | $10^1 \rightarrow 10$ | $2^1 \rightarrow 2$ |
| 3  | $10^2 \rightarrow 100$ | $2^2 \rightarrow 4$ |
| 4  | $10^3 \rightarrow 1000$ | $2^3 \rightarrow 8$ |
| 5  | $10^4 \rightarrow 10,000$ | $2^4 \rightarrow 16$ |
| 6  | $10^5 \rightarrow 100,000$ | $2^5 \rightarrow 32$ |
| 7  | $10^6 \rightarrow 1,000,00$ | $2^6 \rightarrow 64$ |
| 8  | $10^7 \rightarrow 10,000,000$ | $2^7 \rightarrow 128$ |
| 9  | $10^8 \rightarrow 100,000,000$ | $2^8 \rightarrow 256$ |
| 10 | $10^9 \rightarrow 1,000,000,000$ | $2^9 \rightarrow 512$ |

In binary, need 10 digits to represent a factor of $\approx 1000$.

# A Historical Interlude: (Cont'd)
## The Glory of Binary Numbers (Cont'd)

Done for reliability, e.g., distinguishing between and maintaining two voltage levels is **much** easier to do than distinguishing between and maintaining ten voltage levels.

| Decimal | | Binary |
|---------|-----|--------|
| (0) +0 | | +0 (0) |
| (1) +5 | | |
| (2) +10 | | |
| (3) +15 | | |
| (4) +20 | vs. | |
| (5) +25 | | |
| (6) +30 | | |
| (7) +35 | | |
| (8) +40 | | |
| (9) +45 | | +45 (1) |

To convert a binary number to its decimal equivalent, add up the powers of two corresponding to the 1's in the number, e.g.,

$$
\begin{aligned}
110101_2 &= 2^5 + 2^4 + 2^2 + 2^0 \\
&= 32 + 16 + 4 + 1 \\
&= 53_{10}
\end{aligned}
$$

$$
\begin{aligned}
10101100_2 &= 2^7 + 2^5 + 2^3 + 2^2 \\
&= 128 + 32 + 8 + 4 \\
&= 172_{10}
\end{aligned}
$$

To convert a decimal number to its binary equivalent, repeatedly divide by two and read the remainder digits in reverse (from last to first), e.g.,

|  | Quotient | Remainder |  |
|---|---|---|---|
| $53/2 \Rightarrow$ | 26 | 1 | |
| $26/2 \Rightarrow$ | 13 | 0 | |
| $13/2 \Rightarrow$ | 6 | 1 | $\Longrightarrow 110101_2$ |
| $6/2 \Rightarrow$ | 3 | 0 | |
| $3/2 \Rightarrow$ | 1 | 1 | |
| $1/2 \Rightarrow$ | 0 | 1 | |

# Representing Numbers in Binary

Basic base-2 positional notation (**unsigned binary**) can handle the representation and addition of positive integers , e.g.,

$$
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & \\
\end{array}
\qquad \qquad 1 \qquad \leftarrow \text{carry}
$$

$$
\Longleftrightarrow
$$

$$
\begin{array}{r}
1\ 1\ 0\ 1\ 1_2 \\
+\qquad 1\ 1\ 0\ 1_2 \\
\hline
1\ 0\ 1\ 0\ 0\ 0_2 \\
\end{array}
\qquad
\begin{array}{r}
2\ 7_{10} \\
+\ 1\ 3_{10} \\
\hline
4\ 0_{10} \\
\end{array}
$$

. . . But what about negative integers? . . .
. . . Or fractional numbers? . . .

# Representing Numbers in Binary: (Cont'd) Sign/Magnitude Representation

- Use leftmost bit to encode sign (+ve as 0, -ve as 1), e.g.,

| **Negative** | | **Positive** | |
|---|---|---|---|
| $-0_{10}$ | $100_2$ | $000_2$ | $+0_{10}$ |
| $-1_{10}$ | $101_2$ | $001_2$ | $+1_{10}$ |
| $-2_{10}$ | $110_2$ | $010_2$ | $+2_{10}$ |
| $-3_{10}$ | $111_2$ | $011_2$ | $+3_{10}$ |

- Has two representations of zero, and both can arise during arithmetic; this causes problems for computer designers.

## Representing Numbers in Binary: (Cont'd) Excess (Bias) Notation

- Represent negative number by adding bias-value equal to positive equivalent of smallest negative binary number (that is, make all numbers positive), e.g.,

| **Negative** | | **Positive** | |
|---|---|---|---|
| — | | $011_2$ | $+0_{10}$ |
| $-1_{10}$ | $010_2$ | $100_2$ | $+1_{10}$ |
| $-2_{10}$ | $001_2$ | $101_2$ | $+2_{10}$ |
| $-3_{10}$ | $000_2$ | $110_2$ | $+3_{10}$ |
| — | | $111_2$ | $+4_{10}$ |

- Has one zero but more positive than negative numbers.

## Representing Numbers in Binary: (Cont'd) Two's Complement Representation

- Represent negative number by taking unsigned binary positive version and starting after rightmost 1, complement every bit to the left (see also Textbook, p. 160), e.g.,
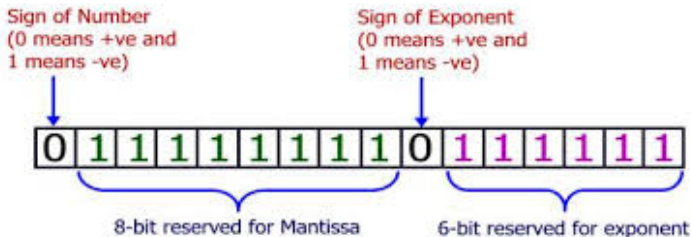
| **Negative** | | **Positive** | |
|---|---|---|---|
| — | | $000_2$ | $+0_{10}$ |
| $-1_{10}$ | $111_2$ | $001_2$ | $+1_{10}$ |
| $-2_{10}$ | $110_2$ | $010_2$ | $+2_{10}$ |
| $-3_{10}$ | $101_2$ | $011_2$ | $+3_{10}$ |
| $-4_{10}$ | $100_2$ | — | |

- Has one zero but more negative than positive numbers.

# Representing Numbers in Binary: (Cont'd) Fractional Numbers

- Store in scientific notation ($M \times B^E$) where base $B = 2$, mantissas ($M$) and exponents ($E$) are stored in signed binary notation, and mantissas are normalized, e.g.,

## Representing Numbers in Binary: (Cont'd) Fractional Numbers (Cont'd)

Example: Representing $5.75_{10}$ as a fractional number

- $5_{10} = 4 + 1 = 2^2 + 2^0 = 101_2$ and
  $.75_{10} = 1/2 + 1/4 = 2^{-1} + 2^{-2} = .11_2$;
  Therefore, $5.75_{10} = 101.11_2$

- Normalize $101.11_2$, e.g.,

$$
\begin{aligned}
101.11_2 &= 101.11_2 \times 2^0 \\
&= 10.111_2 \times 2^1 \\
&= 1.0111_2 \times 2^2 \\
&= .10111_2 \times 2^3
\end{aligned}
$$

- Encode $M = +.10111_2$ and $E = +3_{10} = +11_2$, i.e.,

$$0 + 00010111 + 0 + 000011$$

## Representing Numbers in Binary: (Cont'd) More Compact Representations

Long binary numbers represented more compactly in octal (8 digit; $0, 1, \ldots, 7$) or hexadecimal (16 digit; $0, 1, \ldots, 9, A, B, C, D, E, F$), e.g.,

$$
\begin{array}{ccc|ccc|ccc|ccc}
 & 7 & & & 7 & & & 1 & & & 1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1_2
\end{array} \Rightarrow 7711_8
$$

$$
\begin{array}{cccc|cccc|cccc}
 & F & & & & C & & & & 9 & & \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1_2
\end{array} \Rightarrow FC9_{16}
$$

To convert an octal (hexadecimal) number to its decimal equivalent, add up the powers of eight (sixteen) multiplied by non-zero digits, e.g.,

$$302_8 = (3 \times 8^2) + (0 \times 8^1) + (2 \times 8^0)$$
$$= (3 \times 64) + 2$$
$$= 194_{10}$$

$$A9_{16} = (10 \times 16^1) + (9 \times 16^0)$$
$$= 160 + 9$$
$$= 169_{10}$$

To convert a decimal number to its octal (hexadecimal) equivalent, repeatedly divide by eight (sixteen) and read the remainder digits in reverse (from last to first), e.g.,

|                      | Quotient | Remainder |                     |
|----------------------|----------|-----------|---------------------|
| $194/8 \Rightarrow$  | 24       | 2         |                     |
| $24/8 \Rightarrow$   | 3        | 0         | $\Longrightarrow 302_8$ |
| $3/8 \Rightarrow$    | 0        | 3         |                     |
| $169/16 \Rightarrow$ | 10       | 9         |                     |
| $10/16 \Rightarrow$  | 0        | 10        | $\Longrightarrow A9_{16}$ |

What does

010000010101100011001111

mean?

Things that binary computer memory does well:

**Sounds**

01000#00101#01100#01100#01111

**Pictures**



**Text**
*"HELLO"*

**Numbers**

*8, 5, 12, 12, 15*
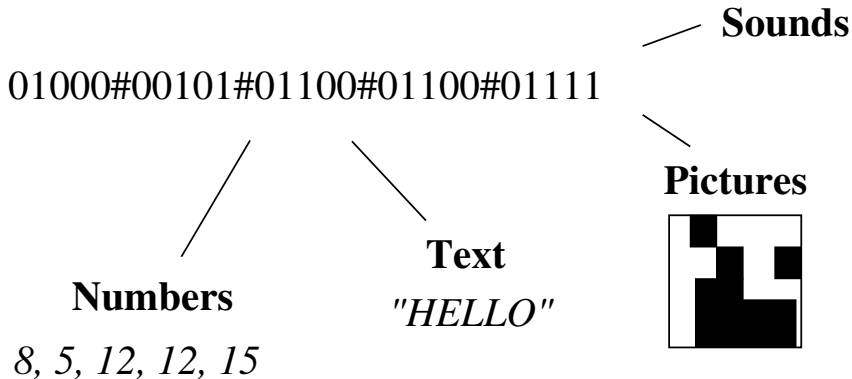
# Representing Text in Binary

- Associate individual symbols with unsigned binary numbers; these symbols may not be printable but rather instructions to I/O interface devices, e.g., control characters.

- Original ASCII and EBCDIC standards used 7 and 8 bits to represent 128 and 256 symbols (see Table 4.3 in textbook for part of ASCII standard).

- Original UNICODE standard used 16 bits to represent $\approx 65{,}000$ symbols; embedded original ASCII standard in lowest 128 codes.

- UNICODE subsequently extended to 32 bits; can accommodate $\approx$ two billion symbols.

# Representing Sound in Binary

- Encode periodic audio signal samples as integers.
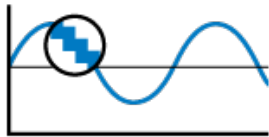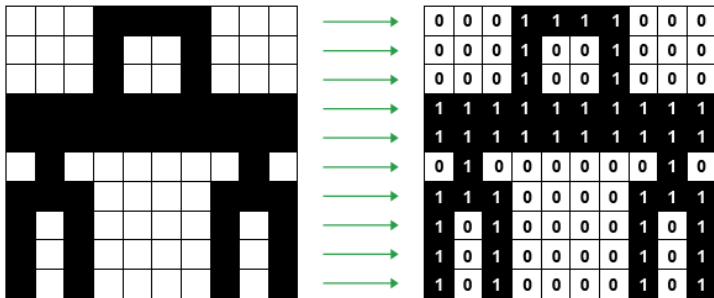- 40,000 samples/sec suffices for human hearing.
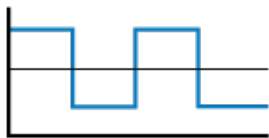


1-bit
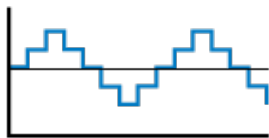
2-bit
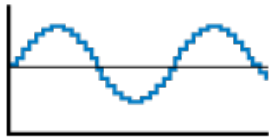
4-bit

16-bit

# Representing Pictures in Binary

- Reduce picture to grid of picture elements (**pixels**).
- Encode pixel values as one or more integers, e.g., single bits (B/W (see below)), 8-value gray scale (B/W), triplets of 256-value red / green / blue intensities (color).

Things that binary computer memory doesn't do so well:

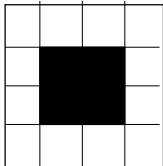Arbitrary
Fractional Numbers

$1/3 \longrightarrow 0.33$

$1/3 \longrightarrow 0.33$

$+ \; 1/3 \longrightarrow 0.33$

$1 \neq 0.99$

Detailed
Pictures

*Earth*

$\downarrow$ ???

# . . . And If You Liked This . . .

- MUN Computer Science courses on this area:

    - COMP 2003: Computer Architecture
    - COMP 3731: Introduction to Scientific Computing
    - COMP 4734: Matrix Computations and Applications

- MUN Computer Science professors teaching courses / doing research in in this area:

    - Sharene Bungay
    - Rod Byrne
    - George Miminis