Lecture 7.2

Internal Data Representation
Why binary representation makes sense?

- Inside computers, all data is represented as 0’s and 1’s
- Why binary representation makes sense?
- Because …
  - The RAM is electric,
  - The disk is magnetic,
  - The communications is optical,
  - The CPU logic is true/false
- All bi-state artifacts!
- The binary system is a perfect match to the way computers work.
Basic Program Elements: lecture outline

- **Number systems**
  - Binary
  - Octal
  - Hexadecimal
  - Decimal

- **Internal representations of**
  - Positive integers
  - Negative integers
  - Fractional numbers

- **Internal representations of**
  - Characters and strings
  - Images
  - Music
  - Other media types.
# Number systems (well, some of them)

<table>
<thead>
<tr>
<th>unary</th>
<th>decimal</th>
<th>binary</th>
<th>3-bit register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>1</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>2</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>***</td>
<td>3</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>****</td>
<td>4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>*****</td>
<td>5</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>******</td>
<td>6</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>*******</td>
<td>7</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>*******</td>
<td>8</td>
<td>1000</td>
<td>overflow</td>
</tr>
<tr>
<td>*******</td>
<td>9</td>
<td>1001</td>
<td>overflow</td>
</tr>
<tr>
<td>*******</td>
<td>10</td>
<td>1010</td>
<td>overflow</td>
</tr>
</tbody>
</table>
How many things can you represent with two signs?

<table>
<thead>
<tr>
<th>1 bit</th>
<th>2 bit</th>
<th>3 bit</th>
<th>4 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>010</td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>011</td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>0101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>110</td>
<td>0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

- A binary system based on \( n \) bits can represent \( 2^n \) different things.
- Each time we add a bit to the representation, we can represent twice as many things.
- We can use these bit combinations to create agreed-upon codes of...
  - Numbers
  - Characters
  - Images
  - Etc.
The binary system

Let’s use cards to represent numbers:

Using the cards to represent numbers

Binary representation of 9

- What is the smallest number that we can represent?
- The largest?
- Can we represent any number within this range?
- Is the representation unique?
- How can you tell that a number is odd / even?

- How to convert binary to decimal?
- How to convert a decimal to binary?
- How to add 1 to a given number?
The binary system

- **Computing the decimal value of a binary number:**

\[
(10011)_{two} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19
\]

- **The general scheme:**

\[
(x_n x_{n-1} \ldots x_0)_b = \sum_{i=0}^{n} x_i \cdot b^i
\]

- **This scheme works with any given number system with base b:**

\[
(9038)_{ten} = 9 \cdot 10^3 + 0 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0 = 9038
\]
Basic Program Elements: lecture outline

- Number systems
  - Binary
  - Octal
  - Hexadecimal
  - Decimal

- Internal representations of
  - Positive integers
  - Negative integers
  - Fractional numbers

- Internal representations of
  - Characters and strings
  - Images
  - Music
  - Other media types.
### Binary, octal, hexadecimal

<table>
<thead>
<tr>
<th>Binary</th>
<th>Octal</th>
<th>Hexa</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- Binary-octal-hexa conversions are easy
- It is customary to display internal representation of data inside computers using octal or hexa.
Examples where octal and hexa are used in computing

### Unix file permissions

\[(r = \text{read}, \ w = \text{write}, \ x = \text{execute})\]

<table>
<thead>
<tr>
<th>Unix internal representation of file permissions (octal)</th>
<th>(Owner, Group, Others)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>---  no permission</td>
</tr>
<tr>
<td>1</td>
<td>--x  execute</td>
</tr>
<tr>
<td>2</td>
<td>-w-  write</td>
</tr>
<tr>
<td>3</td>
<td>-wx  write and execute</td>
</tr>
<tr>
<td>4</td>
<td>r--  read</td>
</tr>
<tr>
<td>5</td>
<td>r-x  read and execute</td>
</tr>
<tr>
<td>6</td>
<td>rw-  read and write</td>
</tr>
<tr>
<td>7</td>
<td>rwx  read, write and execute</td>
</tr>
</tbody>
</table>

The hexa value `%20` is 32 in decimal, representing “space” in ASCII

URLs

http://www.example.com/name%20with%20spaces

(Two unrelated examples)
Examples where octal and hexa are used in computing

```java
class Foo {
}
```

Memory dump = a snapshot of a memory segment

Sometimes used for hardcore debugging, or hacking

Memory addresses and contents are usually shown in hexa.

(The Java program and the memory dump are unrelated)
Basic Program Elements: lecture outline

- Number systems
  - Binary
  - Octal
  - Hexadecimal
  - Decimal

- Internal representations of
  - Positive integers (done)
  - Negative integers
  - Fractional numbers

- Internal representations of
  - Characters and strings
  - Images
  - Music
  - Other media types.
Representing negative numbers

The 2’s complement system:

- Divide all possible codes to two subsets
- Assign the codes that begin with 0 to positive numbers
- Assign the codes that begin with 1 to negative numbers
- To negate a number: leave all trailing 0’s and first 1 intact; flip all the remaining bits

Implications:

We know how to represent byte, short, int, long, and char
Basic Program Elements: lecture outline

- Number systems
  - Binary
  - Octal
  - Hexadecimal
  - Decimal

- Internal representations of
  - Positive integers
  - Negative integers
  - Fractional numbers

- Internal representations of
  - Characters and strings
  - Images
  - Music
  - Other media types.
Representing fractional numbers

| Regular representation:      | 532.81 | 0.00791 |
| Floating point representation: | 53281 * 10^{-2} | 791 * 10^{-5} |
| Normalized:                  | 5.3281 * 10^{-4} | 7.91 * 10^{-3} |

- In normalized form, there is only one digit before the decimal point, and this digit must not be zero. The decimal point “floats” to affect the necessary correction.
- Thus, we can represent fractional numbers as pairs of two numbers: the significand and the exponent.

Floating point arithmetic (example):

\[ 123456.7 + 101.7654 = (1.234567 * 10^5) + (1.017654 * 10^2) \]
\[ = (1.234567 * 10^5) + (0.001017654 * 10^5) \]
\[ = (1.234567 + 0.001017654) * 10^5 \]
\[ = 1.235584654 * 10^5 \]

All arithmetic operations are done using floating point representation.
The numbers are converted to decimals only when we wish to read or write a value.
Floating point in binary

Each bit in the significand represents a value that halves, as follows:

\[
\text{bit 22} = 0.5, \text{ bit 21} = 0.25, \text{ bit 20} = 0.125, \text{ bit 19} = 0.0625 \ldots
\]

This may lead to slight accuracy anomalies.

Implications:

This is how float and double are represented.
Basic Program Elements: lecture outline

- Number systems
  - Binary
  - Octal
  - Hexadecimal
  - Decimal

- Internal representations of
  - Positive integers
  - Negative integers
  - Fractional numbers

- Internal representations of
  - Characters and strings
  - Images
  - Music
  - Other media types.
Representing non-numeric data

The data on the right can be part of a
- Spreadsheet
- Word document
- JPG Image
- Video stream
- Medical record
- Operating system code
- Java code
- ...

So what is it?
- There’s is no way to tell
- The semantics of the data makes sense only in the context in which we use it.

A RAM segment (arbitrary snapshot)

| 10102501 | 00110110001001100011011000110110 |
| 10102502 | 10101010000101010101010110110110 |
| 10102503 | 10110110001101101011011010100110 |
| 10102504 | 10110010001101101011011010100110 |
| 10102505 | 10110010001101101011011010100110 |
| 10102506 | 10110010001101101011011010100110 |
| 10102507 | 10110010001101101011011010100110 |
| 10102508 | 10110010001101101011011010100110 |
| 10102509 | 10110010001101101011011010100110 |
| 10102510 | 10110010001101101011011010100110 |
| 10102511 | 10110010001101101011011010100110 |
| 10102512 | 10110010001101101011011010100110 |
| 10102513 | 10110010001101101011011010100110 |
| 10102514 | 10110010001101101011011010100110 |
| 10102515 | 10110010001101101011011010100110 |
| 10102516 | 10110010001101101011011010100110 |
|   ...   |      ...                      |
Representing characters and strings

```java
public class Charade {
    public static void main(String[] args) {
        System.out.println();
        String s1 = "It\'s just";
        String s2 = "a huge, incredible";
        String s3 = "charade.";
        System.out.println(s1 + " " + s2 + " " + s3);
    }
}
```

There’s a big difference between the way the data is rendered to our senses, and the way the data is stored in memory.

This should not be surprising.
Representing other media types

- Text (e.g. Ascii)
- Sound (e.g. MP3)
- Images (e.g. JPG)
- Programs (e.g. Bytecode)

The conversions are performed only when data is acquired or rendered.

The conversions are typically performed by programs running on device drivers.
Leibniz complained that “All who are occupied with the reading or writing of scientific literature have assuredly very often felt the want of a common scientific language, and regretted the great loss of time and trouble caused by the multiplicity of languages employed in scientific literature”

Leibniz dream: *Charactertia Universalis*: A universal, formal, language of reasoning in which any assertion can be proved or disproved automatically.

The dream’s end: Turing and Goedl in 1930’s.