Intro to computers (continued),
Intro to algorithms and programming

Jan 16, 2015

Recall: The binary number system

Example:

<table>
<thead>
<tr>
<th>Binary number</th>
<th>Value encoded by the binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0 \times 2^0 = 0$</td>
</tr>
<tr>
<td>1</td>
<td>$1 \times 2^0 = 1$</td>
</tr>
<tr>
<td>10</td>
<td>$1 \times 2^1 + 0 \times 2^0 = 2$</td>
</tr>
<tr>
<td>11</td>
<td>$1 \times 2^1 + 1 \times 2^0 = 3$</td>
</tr>
<tr>
<td>1010</td>
<td>$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 2 = 10$</td>
</tr>
</tbody>
</table>

Recall: Memory cell used by a computer

• One switch can be in one of 2 states
• A row of $n$ switches:

  can be in one of $2^n$ states!

Recall: What does all this have to do with a computer?

• The connection between the computer memory and the binary number system is:

  - The computer system uses the binary number encoding to store the number

Example:

Computer memory

• Structure of a RAM memory:

  - The RAM memory used by a computer consists of a large number of electronic switches
  - The switches are organized in rows
  - For historical reasons, the number of switches in one row is 8

Computer memory jargon:

• Bit = (binary digit) a smallest memory device
  A bit is in fact a switch that can remember 0 or 1
• (The digits 0 and 1 are digits used in the binary number system)

• Byte = 8 bits
  A byte is in fact one row of the RAM memory

• KByte = kilo byte = $1024 (= 2^{10})$ bytes (approximately 1,000 bytes)
• MByte = mega byte = $1048576 (= 2^{20})$ bytes (approximately 1,000,000 bytes)
• GByte = giga byte = $1073741824 (= 2^{30})$ bytes (approximately 1,000,000,000 bytes)
• TByte = tera byte (approximately 1,000,000,000,000 bytes)
Overview of the RAM (Memory)

- Structure of the Memory (RAM)

- The RAM consists of multiple memory cells:

  ![Memory Cell Diagram]

- Each memory cell is uniquely identified by its memory address
- Memory addresses always start at zero (0) and the last memory address depends on the amount of memory installed in the computer system

Why 32 bits?

- Note: the address is also expressed as a binary number
- A computer can have over \(4,000,000,000\) bytes (4 Gigabytes) of memory.
  - Which means we need \(4,000,000,000\) (> \(2^{30}\)) unique addresses.

  And hence, we need a **32 bits** to express the address

Individual memory cells

- Each memory cell has 1 byte (8 bits) and therefore, it can store:
  - \(2^8 = 256\) different patterns
  (These 256 patterns are: 00000000, 00000001, 00000010, 00000011, ..., 11111111)

Storing text (each character fits in one memory cell)

Details:
- In order to store text information in a computer, we need to encode:
  - 26 upper case letters ('A', 'B', and so on)
  - 26 lower case letters ('a', 'b', and so on)
  - 10 digits ('0', '1', and so on)
  - 20 or so special characters ('!', '#', '$', and so on)
  for a total of about **100 different symbols**
- The nearest even power \(2^n\) that is larger than 100 is:
  - \(2^n = 128 > 100\)
- For a reason beyond the scope of this course, an **8th switches is added**

Combining adjacent memory cells

- Each pattern can be encoded exactly **one number**:
  - 00000000 = 0
  - 00000001 = 1
  - 00000010 = 2
  - 00000011 = 3
  - ...
  - 11111111 = 255

Therefore, one byte can store one of **256 possible values**
(You can store the number **34** into a byte, but you **cannot** store the number **456**, the value is **out of range**)

Combining adjacent memory cells (cont.)

- The computer can combine adjacent bytes (memory cells) and use it as a larger memory cell

  Schematically:
  
  ![Byte Diagram]

  A 16 bits memory cell can store one of \(2^{16} = 65536\) different patterns.
  Therefore, it can represent (larger) numbers ranging from: 0 – 65535.
Combining adjacent memory cells (cont.)

- Example: how a computer can use 2 consecutive bytes as a 16 bits memory cell:

  ![Memory Diagram](memory_diagram.png)

  The bytes at address 0 and address 1 can be interpreted as a 16 bits memory cell (with address 0)

Combining adjacent memory cells (cont.)

- The computer can also:
  - combine 4 consecutive bytes and use them as a 32 bits memory cell
    - Such a memory cell can represent numbers ranging from: 0 – (2^16 - 1) or 0 – 4294967295
  - combine 8 consecutive bytes and use them as a 64 bits memory cell
    - Such a memory cell can represent numbers ranging from: 0 – (2^32 - 1) or 0 – 4294967295

  There is no need today to combine 16 consecutive bytes and use them as a 128 bits memory cell
  But this may change in the future...

How a computer executes instructions

For details on how a computer executes instructions see the associated lecture notes from Prof. Cheung available via our course website or at:

http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/01/intro-computer2.html

Fact: computers are dumb machines

- Basic property of a computer (a machine):
  - Computers do what we tell them to do
  - Unfortunately, computer do not necessarily do what we want them to do...

  (Because we can make mistake in telling the computer what we want to do... These mistakes are called "bugs")

Intro to Computer Algorithms
Algorithm

- Definition: **algorithm**

  Dictionary definition:

  - Algorithm = a step-by-step procedure for solving a problem or accomplishing some task, especially by means of a computer

Computer Algorithms

- Computer Algorithm:

  Is an algorithm that can be executed by a computer

Computer Algorithms (cont.)

- Properties of computer algorithms:

  - The steps in an algorithm must be consists of operations that can be executed by a computer
  - The step in an algorithm must be unambiguous
  - (Remember that a dumb machine like a computer will do what it is told to do.

Resolution ambiguity requires some thinking (intelligence) which computers cannot do !)

Algorithm development

- We will now illustrate the process of developing an algorithm
- An algorithm always accomplishes some well-defined task or solves some well-defined problem
- The task/problem that we will use to illustrate the process of developing an algorithm is:

  - Replacing a burned out light bulb

Instruction for humans on replacing a burned out light bulb

- Typical instructions given to humans on how to replace a light bulb:

  - Remove the burned-out bulb
  - Insert a new bulb

These brief instructions assume a lot of common sense knowledge that a machine does not have !!!

Instruction for humans on replacing a burned out light bulb (cont.)

- What can go wrong if a machine uses these instructions:

  - A machine does not know how to remove a bulb
    It could yank the bulb out of its socket and damage the fixture in the process.
  - A machine does not know how to insert a bulb
  - A machine can replace the bulb with one that has an inadequate wattage (too bright or too dim)
Instructions for computers on replacing a burned out light bulb

- Computers have no common sense knowledge (really dumb)
- Instructions for computers must be given very explicitly (in “baby steps”)

A real life algorithm

- Here is a famous algorithm that you can try out:
  - Take 2 pieces of paper and write on one piece “A” and on the other piece “B”:
  - Write a positive integer on each piece of paper.

A real life algorithm (cont.)

- Example of the Euclid Algorithm:
  - We start with the following 2 numbers:
  - Since number on A < number on B, we replace the number on B (= 36) by the value (B - A) = 8:

A real life algorithm (cont.)

- Euclid’s Algorithm:
  - As long as one of the number is not zero (0) do
    - if ( number on A ≥ number on B )
      - replace the number on A by the value (A - B)
    - otherwise
      - replace the number on B by the value (B - A)
  - The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers

A real life algorithm (cont.)

- Now number on A ≥ number on B, so we replace the number on A (= 28) by the value (A - B) = 20:
  - Again, number on A ≥ number on B, so we replace the number on A (= 20) by the value (A - B) = 12:
A real life algorithm (cont.)

• Yet again, number on A ≥ number on B, so we replace the number on A (= 12) by the value (A – B) = 4:

\[
\begin{array}{c}
A & B \\
4 & 8 \\
\end{array}
\]

\[ \text{This time, number on B > number on A, so we replace the number on B (= 8) by the value (B – A) = 4:} \]

\[
\begin{array}{c}
A & B \\
4 & 4 \\
\end{array}
\]

A real life algorithm (cont.)

• Now, number on A ≥ number on B, so we replace the number on A (= 4) by the value (B – A) = 0:

\[
\begin{array}{c}
A & B \\
0 & 4 \\
\end{array}
\]

• One of the number on the papers is now zero

According to the Euclid's Algorithm, the other number (= 4) is equal to the GCD of (28, 36)

Intro to programming a computer

What does programming a computer mean?

• Programming a computer:

\[ \text{Programming a computer = instruct a computer to perform a task.solve a problem} \]

• Since a computer can only execute machine instructions (encoded in binary numbers), this means:

\[ \text{Write the steps of an algorithm using machine instructions} \]

What does programming a computer mean? (cont.)

This is a painfully complicated process

But fortunately, we have developed tools that help make this process less painful

Instructing a computer to execute an algorithm

• A computer is the ideal machine to execute computational algorithms because:

\[ \text{The computer can perform arithmetic operations} \]

\[ \text{It can also perform an operation only when some condition is satisfied (using the conditional branch instruction)} \]

• Problem

\[ \text{We (a human) has to tell (communicate) the computer what steps to do in machine language} \]

Humans are not good in machine languages
Instructing a computer to execute an algorithm (cont.)

- We have **developed specialized languages** (based on the English language) to instruct/command a computer.

These specialized languages are called:

- **High level programming languages**

**Types of languages used in computer programming** (cont.)

- **Machine language**
  - The **machine language** (or instruction code) consists of (binary) numbers that encode instructions for the computer.
  - Every computer (CPU) has its own **machine language** (i.e., the instruction code 1 can encode a different instruction for different CPUs).
  - Instruction encoding was discussed in this webonote: click here.

**Types of languages used in computer programming** (cont.)

- **Assembler language or low level programming language**
  - An assembler language consists of (English-like) mnemonics.
  - There is one mnemonic for each machine instruction of the computer.

**High level programming language**

- A high level programming language consists of (English-like) "people" language to simplify the writing computer algorithms.
- A high level programming language allows the programmer to write sentences in this language which can be easily translated into machine instructions.
- The sentences written in a high level programming language are called:
  
  - **statements**

**Example high level program:**

```java
main() {
  if (x > y) // One sentence (statement) in a high level prog. lang.
    { max = x; 
    }
  else
    { max = y; 
    }
  ....
}
```
**High level programming language (cont.)**

One statement in a high level programming language is usually translated into multiple machine instruction that achieve the effect specified by the statement.

---

**A taste of computer algorithm**

- **Recall** the Euclid’s Algorithm: (to find the GCD)

  ```java
  As long as one of the number is not zero (0) do
  
  if ( number on A ≥ number on B )
  replace the number on A by the value (A - B)
  otherwise
  replace the number on B by the value (B - A)
  
  The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers.
  ```

---

**A taste of computer algorithm (cont.)**

- **Note:**
  - Don’t worry if you do not understand this program.
  - All I want to do is to show you is the fact that a computer can be instructed by a specialized (programming) language to do things.
  - You will learn to yourself very soon!!!
Computer Programs

- Computer Program:
  - Computer program = a computer algorithm that is expressed (written) in some programming language
  - Most programming languages are “English-like”
  - They use words from the English language

Representation gap: algorithm in programming and comp instructions

- Notice the following representation gap:
  1. An algorithm expressed in a programming language is "English-like"
     The algorithm consists of instructions that the computer must perform to accomplish a given task/solve a given problem
  2. However: the instructions that a computer executes are numbers (that encode the instruction)
     These numbers that encode computer instructions are not "English-like" !!!

Bridging the translation gap: compiler

- We bridge the representation gap using 2 techniques:
  1. We design the programming language in such a way that the sentences in the programming language can be easily understood
  2. We write a (pretty complex) computer program to do the translation from:

Solution

- An algorithm expressed in a programming language cannot be executed by a computer

- Someone (or something) must translate the algorithm expressed in a programming language into numbers that encode the same instructions as the original algorithm

Bridging the translation gap: compiler (cont.)

- The (pretty complex) computer program that does this translation is called:

The compilation (= translation) process (cont.)

- Before the computer can execute the instructions in a Java program, we must first compile (translate) it into machine operations using a Java translator (compiler):

  - javac prog.java

  - javac is the command to run the Java compiler (translator)
  - The second word prog.java is the name of the file that contains the Java program (that is being translated)
  - The machine operations that result from the translation process is stored in the file:

 prog.java
Writing computer programs: trust your compiler!

- When you write a computer program, you will write it in a high level programming language and trust the compiler (translator) to do the translation.
- It is virtually impossible to write a large computer program in machine language (binary numbers).

Writing computer programs: trust your compiler! (cont.)

- Therefore:
  - We will not dwell on machine code any longer in this course.
  - We will learn to write computer programs in a high level programming language.

- The high level programming language used in this course is:
  - Java