Computer Systems
Lecture 16

Overview

- Data codes – numeric and character
- Number representations
- Unsigned integers
- Binary vs. BCD representation
- Signed integers
- Sign-and-magnitude representation
- Complementary representations
- 10’s complementary coding

Reminder: components of computer systems

- There are three components required for the implementation of INPUT-PROCESS-OUTPUT and von Neumann model(s):
  - **Hardware**.
  - **Software**.
  - **Data that is being manipulated**.

Data codes – numeric and character

- Within the computer the **binary number system** is a system of choice, both for data storage and for internal processing of operations.

Alphanumeric data

- The majority of the data originally comes in the form of letters in **alphanumeric data**:
  - alphabet,
  - numbers, and
  - punctuation

Alphanumeric codes

- **ASCII** code (**A**merican **S**tandard **C**ode for **I**nformation **i**nterchange (7-bit code) and its extensions (8-bit codes) (well-established).
- **EBCDIC** code (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode) 8-bit code. (IBM mainframe computers)
- **Unicode.** Recent 16-bit standard. (Up to 2^{16} characters can be encoded)

Numbers vs. text

- **Numbers** (consisting of numeric characters) are often processed differently from text.
- But still they must be entered into computer just like any other alphanumeric characters, one digit at a time.
  - Number 314 consists of three alphanumeric characters.
  - Number 3.14 ?
  - Number -3.14 ??

Numbers vs. text (cont.)

- Conversion between alphanumeric representation of numbers and special representation of numbers is done usually in a program (software).
  - The method `valueOf(double)` in `java.lang.String` class converts a double float number to its string representation.
  - The method `java.lang.Integer.parseInt(String)` converts a string into the integer value that it represents.

Number representations

- We have seen that a computer stores all data in binary form.
- To store numbers we need an **encoding scheme**, which would allow us to encode:
  - The **algebraic sign** of numbers (+/-).
  - **Decimal point** that might be associated with a fractional number.

Number representations (cont.)

- One solution.
  - Store the number as a string of characters.
  - However, it is not practical for calculations.
- Unsigned integers can be represented directly in binary form.
  - What about negative numbers?

Two categories of integer data

Unsigned integers

Signed integers

Unsigned integers

- **Unsigned binary representation** – just store any whole number in its binary representation.
- Thus, a byte can store any **unsigned integer** of value between 0 and 255.
- A 32-bit word can store **unsigned integers** in a range $0 – (2^{32} - 1) = 4{,}294{,}967{,}295$.

- Multiple storage locations (words) can be used to represent bigger unsigned integers.

Unsigned integers: BCD

- **Binary-coded decimal** (BCD):
  - Digit-by-digit binary representation of original decimal integer.
- Each decimal digit is **individually** converted to binary.
  - This requires **4 bits per digit** (not all 4-bits patterns are used).
- Example.
  - Decimal value 68 is presented in BCD as 01101000.
- One byte can store **unsigned integers** in a range 0-99 under BCD encoding (Why?).

Binary vs. BCD representation

- BCD is less economical than binary representation (Why?).
- Calculations in BCD are more difficult.
- But, translation between BCD and character form is easier.
  - Last 4 bits of ASCII numeric character codes correspond precisely to the BCD representation.
- Example.
  - ASCII code of ‘5’ is 0110101, its BCD representation is 0101.
- Binary representation is more common, BCD is used for some business applications.

Signed integers

- Sign-and-magnitude representation.
- Complementary representations.

Sign-and-magnitude representation

- It is representation of **signed integers** by a **plus** or **minus** sign and a **value**.
- **Agreement**.
  - **Left-most** bit represent a sign, e.g., 0 stands for + and 1 stands for -.
- Example.
  - 00100111 represents 39.
  - 10100111 represents -39.
  - 00000000 represents ?
  - 10000000 represents ?

Sign-and-magnitude representation (cont.)

- Calculations are more difficult than in the case of binary (unsigned) representation.
- There are two different binary codes for the number 0: 00000000 and 10000000 (8-bit codes).
- Positive range of the signed integer is one-half as large as the unsigned integer of the same number of bits.
  - The same holds for negative range.
- Thus, 8-bits can represent the numbers from -127 to 127 (0 being represented twice).

Complementary representations

- For most purposes, computers use different methods of representing signed integers known as **complementary representations (conventions)**.
- Two's binary complementary representation is the most common.

10’s complementary coding

- 10’s complementary coding is decimal analogue of 2’s complementary.
- To explain the idea we first consider 10’s complementary convention.

10’s complementary coding (cont.)

- Consider the case of 3 decimal digits.
- **The problem is**:
  - How to use 3 decimal digits to represent positive and negative numbers, so that
  - the usual operations (addition, etc) are computable.

10’s complementary coding (cont.)

- Solution:

Representation	$500\ldots\ 999$	$0\ldots499$
Number being represented	$-500\ldots-001$	$0\ldots499$

- Thus, the numbers (3-digit sequences):
  - From 000 to 499 are representing **themselves**.
  - Any number ***x*** from 500 to 999 represents **negative** number -(1000 - ***x***), that is, negated complement to 1000.

10’s complementary coding (cont.)

- Examples.
  - ‘120’ is a code of the number 120.
  - ‘612’ is a code of the number -388 = -(1000 - 612).
  - ‘500’ is a code of the number -500 = -(1000 - 500).
  - ‘000’ is a code of the number 0.
  - ‘999’ is a code of the number -1 = -(1000 - 999).

  - Number 0 is not represented twice!

10’s complementary coding - n digits

- This convention is easily extended to the case of n decimal digits (How?).

- The numbers (sequences) from 0…0 to 49…9 represent themselves.
- The numbers from 50…0 to 99…9. represent negated complements to 100…0.

Method of complements

- A technique in mathematics and computing used to subtract one number from another using only addition of positive numbers.
- Commonly used in mechanical calculators and in modern computers.

Examples of ($x - y$)

- Assume $x \geq y$.
  - Instead of subtracting y from x, the complement of y is added to x and the leading '1' of the result is discarded.
- Example (working with 3 digits).

373	(x)	→	373	(x)
- 218	(y)		+ 782	(10's complement of -y)
?			1155	

  - The first '1' digit is then dropped, giving 155, the correct answer.

Examples of ($x - y$) (cont.)

- In the case $x < y$.
  - There will not be a ‘1’ digit to cross out after the addition.
- Example (working with 3 digits).

	185	(x)
-	329	(y)
	?	

→

	185	(x)
+	671	(10’s complement of -y)
	856	(10’s complement of the result -144)

  - Representing negative numbers in 10’s complements.

- Q. What are the pros & cons of BCD?

- Q. Is it possible to have a 3’s complement scheme?

- Q. Can overflow still occur under 2’s complement addition?

- Q. What are the pros & cons of 2's complement?

Exercise

- Try to implement 10’s complement in 2 decimal digits

Readings

- Wikipedia article:
  - http://en.wikipedia.org/wiki/Method_of_complements