Lecture 2: Data Representation

- Data need to be represented in a convenient way that simplifies:
 - common operations: addition, comparison, etc.
 - How would an algorithm for adding Roman numbers look like?
 - hardware implementation (cheap, fast, reliable computers)
- Data representation is a major part of the software-hardware interface



Lecture outline

- The bit atomic unit of data
- Representing numbers
- Representing text



The bit

- Acronym for Binary digiT
- The smallest amount of meaningful info, according to information theory
 - If only 1 value is possible, there is no information
- Disadvantages: too little information per bit, must use many of them
- Advantages: easy to do computation, very reliable, simple circuits



Natural numbers representation

 Positive (unsigned) integers are very simple to represent in binary





Basic operations

Addition, subtraction with binary numbers is easy:





Fixed bit-length arithmetic

- Hardware cannot handle infinite long bit sequences
- We end up with a few fixed sized data types
 - Byte: always 8 bits
 - Word: the 'natural' unit of access, usually 32 bits
- Overflow happens when a result does not fit
 - Numbers wrap-around when they become too large
 - Comp. arithmetic is modulo 2^n , n=number of bits



What about negative numbers?

- No special symbols, e.g. +, -, available
- Sign-magnitude representation:
 - Use 1st bit (MSB) as the sign: 1-negative, 0-positive
- Complicates the arithmetic operations
 The actual operation depends on the sign
- There is a better way



Two's complement representation

- What is the result of 000 001?
 ...111
- The MSB has negative weighting:



Arithmetic operations do not depend on the operands' signs



2's complement quirks

- The MSB is the sign
- Range is asymmetric: −2ⁿ⁻¹ to 2ⁿ⁻¹-1
- There are two kinds of overflows:
 - Positive overflow produces a negative number
 - Negative underflow produces a positive number
- To negate a number
 - Invert all bits $(0 \leftrightarrow 1)$ and add 1, at the LSB
 - $-(-2^{n-1})$ overflows!
- A-B = A + 2's complement of B

Converting between data types

 Converting from a smaller to a larger representation is done by sign extension

Example: from byte to short (16 bits):





Shifting

- Shifting: move the bits of a data type left or right
 - Data bits falling off the edge are lost
- 0s fill up the empty bit places for left shifts
- For right shifts, two options:
 - Fill with 0: for non-numerical data (or positive integers)
 - Fill with the MSB: for 2's complement numbers
- Shift left by n is equivalent to multiplying by 2ⁿ
- Shift right by n is equivalent to dividing by 2ⁿ
- Example $6 = 00000110 >> 2 \rightarrow 0000001 = 1$ -8 = 11111000 >> 2 \rightarrow 1111110 = -2



Hexadecimal notation

- Binary numbers (and other data) are too long and tedious for us to use
- Hexadecimal (base 16) is very commonly used in computer programming
- Hex digits: 0-9 and A-F

– A=10, B=11, ..., F=15

 Conversion to/from binary is very easy: Every 4 bits correspond to 1 hex digit:

$$\underbrace{1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0}_{F(15) \ 8} = 0 xF8$$

Hex is just a convenience, computers use the binary form



Real numbers - floating point

- Java's fl oat (32 bits) doubl e (64 bits)
- IEEE 754:

- example 0.75 in base $10 \Rightarrow 0.11$ in base 2



- example: 25 in base $10 \Rightarrow 11001$ in base $2 \Rightarrow 1.1001 \text{x}2^4$



Floating Point

• 32 bit:



e.g.,

- 64 bit:
 - exponent = 11 bits; significand = 52 bits
- Note: processors usually have specialized floating point units and extra fp registers to perform fp arithmetic



Representing characters, strings

- Characters need to be encoded in binary too
- Operations on characters have simpler requirements than on numbers, so the encoding choice is not crucial
- Most common representation is ASCII
 - Each character is held in a byte
 - E.g. '0' is 0x30, 'A' is 0x41, 'a' is 0x61
- Java uses Unicode which can encode characters from many (all?) languages
 - 16 bits per character required
- Words, sentences, etc. are just strings of characters
 - A special character, encoded as 0x00, shows where the string ends (in C)



- Or the string length is kept with the string itself (in Java)

Summary

- Computers use binary representation
- 2's complement
- Floating point
- Characters and strings

