Under the Hood: Data Representation

Alvin R. Lebeck
CPS 104
Lecture 2

General Information

• Instructor: Alvin R. Lebeck (alvy)
  Office: D304 LSRC
  email: alvy@cs.duke.edu
  Office Hours: Wed 1:00 - 2:00, Thurs 3:30 - 4:30, or by appointment

• Course Web Page
  http://www.cs.duke.edu/courses/spring01/cps104
  Lecture slides available on web page

• Course News Group
  duke.cs.cps104

• You are required to monitor web page and newsgroup
  Home work will appear on web page
  If necessary, additional information about homework on newsgroup
  You can post questions about homework to newsgroup
Administrivia

Homework
• Homework #1 Due September 12
• Two parts, written due in class, program submit 12pm

Reading
• Ch. 1, skim Ch. 2
• Ch 4.1-4.3, 4.8 pages 275-280
• Start Ch. 3

Today’s Lecture

• First step in mapping high-level to machine
  Data representations

Outline
• Review
• Introduction: Number Systems
• Binary Numbers
• Integer numbers
• Floating-point numbers
• Characters
• Storage sizes: Bit, Byte, Word, Double-word
Review

Goal
• Understand basic operation of a computer

Why?
• Software performance is affected/determined by HW capabilities
• Future Computer Architects (Processor or System)

Review: The Big Picture

• The Five Classic Components of a Computer
Levels of Abstraction

High Level Language Program
Compiler
Assembly Language Program
Assembler
Machine Language Program
Machine Interpretation

Transistors turning on and off

What You Know Today

```c
...
int result;
double score;

double curve(double score) {
    return(score * 0.22124);
}
int main()
{
    int *x;
    ...
    result = x + result;
    cout << “Score is ” << curve(80) << endl;
    ...
}
```
High Level to Assembly

High Level Language (C, C++, Fortran, Java, etc.)

• Statements
• Variables
• Operators
• Methods, functions, procedures

Assembly Language

• Instructions
• Registers
• Memory

Data Representation

• Compute two hundred twenty nine minus one hundred sixty seven divided by twelve

• Compute XIX - VII + IV

• We reason about numbers many different ways

• Computers store variables (data)
• Typically Numbers and Characters or composition of these

• The key is to use a representation that is “efficient”
Number Systems

• A number is a mathematical concept
  10

• Many ways to represent a number
  10, ten, 2x5, X, 100/10, |||| |||||

• Symbols are used to create a representation

• Which representation is best for addition and subtraction?

• Which representation is best for multiplication and division?

More Number Systems

• Humans use decimal (base 10)
  digits 0-9 are composed to make larger numbers
  11 = 1*10^1 + 1*10^0
  weighted positional notation

• Addition and Subtraction are straightforward
  carry and borrow (today called regrouping)

• Multiplication and Division less so
  can use logarithms and then do adds and subtrahs
Changing Base (Radix)

• Given 4 positions, what is the largest number you can represent?

Number Systems for Computers

• Today’s computers are built from transistors
• Transistor is either off or on
• Need to represent numbers using only off and on
  two symbols
• off and on can represent the digits 0 and 1
  BIT is Binary Digit
  A bit can have a value of 0 or 1
• Binary representation
  weighted positional notation using base 2
  \[ 11_{10} = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 1011_{2} \]
  \[ 11_{10} = 8 + 2 + 1 \]
What is largest number, given 4 bits?
Conversion from Decimal to Binary

- \( N \) is a positive Integer (in decimal representation)
- \( b_i \) \( i=0,...,k \) are the bits (binary digits) for the binary representation of \( N \)
- \( N = b_k \cdot 2^k + ... + b_2 \cdot 2^2 + b_1 \cdot 2 + b_0 \)
- binary representation: \( b_k...b_3b_2b_1b_0 \)
- How do I compute \( b_0 \)?
  Compute binary representation of 11?

Conversion from Decimal

\[
\begin{align*}
i &= 0; \\
\text{while } N > 0 \text{ do} & \\
\quad & b_i = N \mod 2; \quad \text{// } b_i = \text{remainder; } N \mod 2 \\
\quad & N = N / 2; \quad \text{// } N \text{ becomes quotient of division} \\
\quad & i++; \\
\text{end while} \\
\end{align*}
\]

- Replace 2 by A and you have an algorithm that computes the base A representation for \( N \)
Powers of 2

<table>
<thead>
<tr>
<th>N</th>
<th>2^n</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>00000000001</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>00000000010</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00000000100</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>00000001000</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>00000010000</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>00000100000</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>00001000000</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>00010000000</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>00100000000</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>01000000000</td>
</tr>
<tr>
<td>10</td>
<td>1024 (1K)</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

Binary, Octal and Hexadecimal numbers

- Computers can input and output decimal numbers but they convert them to internal binary representation.
- Binary is good for computers, hard for us to read
  - Use numbers easily computed from binary
  - Binary numbers use only two different digits: {0,1}
    - Example: \(1200_{10} = 0000010010110000_2\)
- Octal numbers use 8 digits: {0 - 7}
  - Example: \(1200_{10} = 04260_8\)
- Hexadecimal numbers use 16 digits: {0-9, A-F}
  - Example: \(1200_{10} = 04B0_{16} = 0x04B0\)
  - does not distinguish between upper and lower case
Binary and Octal

- Easy to convert Binary numbers To/From Octal.
- Group the binary digits in groups of three bits and convert each group to an Octal digit.
- \(2^3 = 8\)

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>

**Example:**

\[1100\ 0010\ 0110\ 0111\ 0100\ 1111\ 1101\ 0101_2\]

\[3\ 0\ 2\ 3\ 1\ 6\ 4\ 7\ 5\ 2\ 5\_8\]

Binary and Hex

- To convert to and from hex: group binary digits in groups of four and convert according to table.
- \(2^4 = 16\)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bin</th>
<th>Hex</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

**Example:**

\[1100\ 0010\ 0110\ 0111\ 0100\ 1111\ 1101\ 0101_2\]

\[C\ 2\ 6\ 7\ 4\ F\ D\ 5\_16\]
Issues for Binary Representation

- Complexity of arithmetic operations
- Negative numbers
- Maximum representable number
- Choose representation that’s easy for machine not easy for humans

Binary Integers

- Unsigned Integers:
  \[ i = 100101_2 \; ; \; i = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]
- 4 bits => max number is 15
- What about representing negative numbers?
Sign-Magnitude Representation for Integers

- Add a sign bit
  - Example: $010110_2 = 22_{10}$; $110110_2 = -22_{10}$

- Advantages:
  - Simple extension of unsigned numbers.
  - Same number of positive and negative numbers.

- Disadvantages:
  - Two representations for 0: $0=000000$; $-0=100000$.
  - Algorithm (circuit) for addition depends on the arguments’ signs.

1’s Complement Representation for Integers

- Key is to use largest positive binary numbers to represent negative numbers
  - $i = 2^n - x - 1$

- Simply invert each bit (0->1, 1->0)

- Two zeros

6-bit examples:

010110 = 22_{10}; 101001 = -22_{10}
0_{10} = 000000; 0 = 111111
1_{10} = 000001; -1_{10} = 111110
2’s Complement Representation for Integers

- Still large positives to represent negatives
- \(i = 2^n - x\)
- This is 1’s complement + 1
- So, invert bits and add 1

6-bit examples:

- 000000\(_2\) = 0\(_{10}\)
- 111111\(_2\) = -1\(_{10}\)
- 010110\(_2\) = 22\(_{10}\)
- 101010\(_2\) = -22\(_{10}\)

2’s Complement

- Advantages:
  - Only one representation for 0: \(0 = 000000\)
  - Addition algorithm independent of sign bits.

- Disadvantage:
  - One more negative number than positive:
    - Example: 6-bit 2’s complement number.
    - 100000\(_2\) = -32\(_{10}\) but 32\(_{10}\) could not be represented
2’s Complement Negation and Addition

- To negate a number do:
  - Step 1. complement the digits
  - Step 2. add 1

Example

\[ 14_{10} = 001110_2 \]
\[ -14_{10} = 110001_2 \]
\[ +1 \]
\[ 110010_2 \]

- To add signed numbers use regular addition but disregard carry out

Example

\[ 18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10} \]
\[ 010010_2 \]
\[ +110010_2 \]
\[ 000100_2 \]

2’s Complement (cont.)

- Example: \( A = 0x0ABC; \) \( B = 0x0FEB. \)

- Compute: \( A + B \) and \( A - B \) in 16-bit 2’s complement arithmetic.

- Give answer in HEX
2’s Complement Precision Extension

- Most computers today support 32-bit (int) or 64-bit integers
  - 64-bit using gcc is long long
- To extend precision use sign bit extension
  - Precision is number of bits used to represent a number

Example

\[
\begin{align*}
14_{10} & = 001110_2 \text{ in 6-bit representation.} \\
14_{10} & = 00000001110_2 \text{ in 12-bit representation} \\
-14_{10} & = 110010_2 \text{ in 6-bit representation} \\
-14_{10} & = 111111110010_2 \text{ in 12-bit representation.}
\end{align*}
\]

What About Non-integer Numbers?

- There are infinitely many real numbers between two integers
- Many important numbers are real
  - Speed of light \( \approx 3 \times 10^8 \)
  - \( 3.145 \ldots \)
- Fixed number of bits limits range of integers
  - Can’t represent some important numbers
- Humans use Scientific Notation
  - \( 1.3 \times 10^4 \)
Floating Point Representation

Numbers are represented by:

\[ X = (-1)^s \cdot M \cdot 2^{E-127} \]

- **S**: 1-bit field; Sign bit
- **E**: 8-bit field; Exponent: Biased integer, 0 \( \leq E < 255 \).
- **M**: 23-bit field; Mantissa: Normalized fraction with hidden 1 (don’t actually store it)

Single precision floating point number uses 32-bits for representation

\[
\begin{array}{cccc}
31 & 30 & 22 & 0 \\
\hline
8\text{-bit} & 23\text{-bit} \\
\hline
s & \text{exp} & \text{Mantissa}
\end{array}
\]

Floating Point Representation

- The mantissa represents a fraction using binary notation:
  \[ M = s_1 \cdot s_2 \cdot s_3 \ldots = 1.0 + s_1 \cdot 2^{-1} + s_2 \cdot 2^{-2} + s_3 \cdot 2^{-3} + \ldots \]

- Example: \( X = -0.75_{10} \) in single precision \((-\frac{1}{2} + \frac{1}{4})\)

  \(-0.75_{10} = -0.11_2 = (-1) \times 1.1_2 \times 2^{-1} = (-1) \times 1.1_2 \times 2^{126-127}\)

  \( S = 1 \); \( E = 126_{10} = 0111\ 1110_2 \);
  \( M = 100\ 0000\ 0000\ 0000\ 0000\ 0000_2 \)

\[
\begin{array}{cccc}
31 & 30 & 23 & 22 & 0 \\
\hline
\text{s} & \text{E} & \text{M}
\end{array}
\]
Floating Point Representation

Example:
What floating-point number is:
0xC1580000?
### ASCII Character Representation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nul</td>
<td>001</td>
<td>soh</td>
<td>002</td>
<td>stx</td>
<td>003</td>
<td>etx</td>
<td>004</td>
<td>eot</td>
<td>005</td>
<td>enq</td>
<td>006</td>
<td>ack</td>
<td>007</td>
<td>bel</td>
<td>010</td>
<td>bs</td>
<td>011</td>
<td>ht</td>
<td>012</td>
<td>nl</td>
<td>013</td>
<td>vt</td>
<td>014</td>
<td>np</td>
<td>015</td>
<td>cr</td>
<td>016</td>
<td>so</td>
<td>017</td>
<td>si</td>
<td>020</td>
<td>can</td>
<td>031</td>
<td>em</td>
<td>032</td>
<td>sub</td>
<td>033</td>
<td>esc</td>
<td>034</td>
<td>fs</td>
<td>035</td>
<td>gs</td>
<td>036</td>
<td>r</td>
<td>037</td>
<td>us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td>041</td>
<td>!</td>
<td>042</td>
<td>&quot;</td>
<td>043</td>
<td>#</td>
<td>044</td>
<td>$</td>
<td>045</td>
<td>%</td>
<td>046</td>
<td>&amp;</td>
<td>047</td>
<td>'</td>
<td>050</td>
<td>(</td>
<td>051</td>
<td>)</td>
<td>052</td>
<td>*</td>
<td>053</td>
<td>+</td>
<td>054</td>
<td>,</td>
<td>055</td>
<td>-</td>
<td>056</td>
<td>.</td>
<td>057</td>
<td>/</td>
<td>060</td>
<td>0</td>
<td>061</td>
<td>1</td>
<td>062</td>
<td>2</td>
<td>063</td>
<td>3</td>
<td>064</td>
<td>4</td>
<td>065</td>
<td>5</td>
<td>066</td>
<td>6</td>
<td>067</td>
<td>7</td>
<td>070</td>
<td>8</td>
<td>071</td>
<td>9</td>
<td>072</td>
<td>;</td>
<td>073</td>
<td>=</td>
<td>074</td>
<td>&lt;</td>
<td>075</td>
<td>=</td>
<td>076</td>
<td>&gt;</td>
<td>077</td>
<td>?</td>
<td>100</td>
<td>@</td>
<td>101</td>
<td>A</td>
<td>102</td>
<td>B</td>
<td>103</td>
<td>C</td>
<td>104</td>
<td>D</td>
<td>105</td>
<td>E</td>
<td>106</td>
<td>F</td>
<td>107</td>
<td>G</td>
<td>110</td>
<td>H</td>
<td>111</td>
</tr>
</tbody>
</table>

- Each character is represented by a 7-bit ASCII code.
- It is packed into 8-bits

### Basic Data Types

**Bit:** 0, 1  
**Bit String:** sequence of bits of a particular length  
- 4 bits is a nibble  
- 8 bits is a byte  
- 16 bits is a half-word  
- 32 bits is a word  
- 64 bits is a double-word  

**Character:**  
- ASCII: 7 bit code  
- Decimal: (BCD code)  
- digits 0-9 encoded as 0000 thru 1001  
- two decimal digits packed per 8 bit byte  

**Integers:**  
- 2's Complement (32-bit representation).  

**Floating Point:**  
- Single Precision (32-bit representation).  
- Double Precision (64-bit representation).  
- Extended Precision (128-bit representation).  
  - How many +/- #'s?  
  - Where is decimal pt?  
  - How are +/- exponents represented?
Summary

• Computers operate on binary numbers (0s and 1s)
• Conversion to/from binary, oct, hex
• Signed binary numbers
  2’s complement
  arithmetic, negation
• Floating point representation
  hidden 1
  biased exponent
  single precision, double precision

Next Time

Homework #1
Memory
• Pointers
• Arrays
• Strings
Bitwise operations
Reading
• Start Chapter 3