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

[Diagram of the Five Classic Components of a Computer]
Levels of Abstraction

<table>
<thead>
<tr>
<th>High Level Language Program</th>
<th>temp = v[k]; v[k] = v[k+1]; v[k+1] = temp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>he $15, 0($2)</td>
</tr>
<tr>
<td>Assembly Language Program</td>
<td>he $16, 4($2)</td>
</tr>
<tr>
<td>Assembler</td>
<td>sw $16, 0($2)</td>
</tr>
<tr>
<td>Machine Language Program</td>
<td>sw $15, 4($2)</td>
</tr>
<tr>
<td>Machine Interpretation</td>
<td>Transistors turning on and off</td>
</tr>
<tr>
<td>Control Signal Specification</td>
<td></td>
</tr>
</tbody>
</table>

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, 1001010110010101
- 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
  - 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 subtracts
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)
• \( b_i \) is the bits (binary digits) for the binary representation of \( N \)
• \( N = b_k \times 2^k + \ldots + b_2 \times 2^2 + b_1 \times 2 + b_0 \)
• binary representation: \( b_k \ldots b_3 b_2 b_1 b_0 \)
• How do I compute \( b_0 \)?
  Compute binary representation of 11?

Conversion from Decimal

\[ i=0; \]
while \( N > 0 \) do
  \( b_i = N \mod 2; \) // \( b_i = \) remainder; \( N \mod 2 \)
  \( N = N / 2; \) // \( N \) becomes quotient of division
  \( i++; \)
end while

• 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>0000000010</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0000000000</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0000000000</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>0000000000</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>0000000000</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>0000 00000</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>0000 00000</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>00 0000000</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>0 00000000</td>
</tr>
<tr>
<td>10</td>
<td>1024 (1K)</td>
<td>100000000000</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: \( 1200_{10} = 000010010110000_{2} \)
• Octal numbers use 8 digits: \{0 - 7\}
  Example: \( 1200_{10} = 04260_{8} \)
• Hexadecimal numbers use 16 digits: \{0-9, A-F\}
  Example: \( 1200_{10} = 0480_{16} = 0x480_{16} \)
  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.

\[
\begin{array}{c|c}
\text{Bin.} & \text{Oct.} \\
000 & 0 \\
001 & 1 \\
010 & 2 \\
011 & 3 \\
100 & 4 \\
101 & 5 \\
110 & 6 \\
111 & 7 \\
\end{array}
\]

**Example:**
\[
11\ 000\ 010\ 011\ 001\ 110\ 101\ 101_2
\]

\[
\begin{array}{c|c|c|c|c}
\text{Bin.} & \text{Oct.} & \text{Bin.} & \text{Oct.} \\
000 & 0 & 1000 & 8 \\
001 & 1 & 1001 & 9 \\
100 & 4 & 1010 & A \\
101 & 5 & 1011 & B \\
110 & 6 & 1100 & C \\
111 & 7 & 1101 & D \\
\end{array}
\]

\[
2^7 = 8
\]

### Binary and Hex
- To convert to and from hex: group binary digits in groups of four and convert according to table

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Hex} & \text{Bin} & \text{Hex} & \text{Bin} \\
0 & 0000 & 8 & 1000 \\
1 & 0001 & 9 & 1001 \\
2 & 0010 & A & 1010 \\
3 & 0011 & B & 1011 \\
4 & 0100 & C & 1100 \\
5 & 0101 & D & 1101 \\
6 & 0110 & E & 1110 \\
7 & 0111 & F & 1111 \\
\end{array}
\]

**Example:**
\[
1100\ 0110\ 0111\ 0100\ 1111\ 1101\ 0101_2
\]

\[
\begin{array}{c|c|c|c|c}
\text{Hex} & \text{Bin} & \text{Hex} & \text{Bin} \\
0 & 0000 & 8 & 1000 \\
1 & 0001 & 9 & 1001 \\
2 & 0010 & A & 1010 \\
3 & 0011 & B & 1011 \\
4 & 0100 & C & 1100 \\
5 & 0101 & D & 1101 \\
6 & 0110 & E & 1110 \\
7 & 0111 & F & 1111 \\
\end{array}
\]

\[
2^8 = 16
\]

### 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:**
  \[
  i = 100101 \quad i = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0
  \]
  \[
  4 \text{ bits} \Rightarrow \text{max number is 15}
  \]
- What about representing negative numbers?

### Sign-Magnitude Representation for Integers
- **Add a sign bit**
  \[
  \text{Example: } 010110 \Rightarrow 22_{10}, \quad 101101 \Rightarrow -22_{10}
  \]
- **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.

### Binary Integers
- **Unsigned Integers:**
  \[
  i = 100101 \quad i = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0
  \]
  \[
  4 \text{ bits} \Rightarrow \text{max number is 15}
  \]
- What about representing negative numbers?

### 1’s Complement Representation for Integers
- **Key is to use largest positive binary numbers to represent negative numbers**
  \[
  0000 \Rightarrow 0, \quad 0001 \Rightarrow 1, \quad 0010 \Rightarrow 2, \quad 0011 \Rightarrow 3, \quad 0010 \Rightarrow 4, \quad 0011 \Rightarrow 5
  \]
  \[
  \text{Simply invert each bit (0->1, 1->0)}
  \]
- **Two zeros**
  \[
  0111 \Rightarrow 7, \quad 1000 \Rightarrow 0, \quad 1001 \Rightarrow 1, \quad 1010 \Rightarrow 2, \quad 1011 \Rightarrow 3, \quad 1100 \Rightarrow 4
  \]
- **Non-complementing examples:**
  \[
  01010 \Rightarrow 10, \quad 10101 \Rightarrow 5, \quad 01000 \Rightarrow 8, \quad 11111 \Rightarrow 15
  \]
- **1’s complement number in 6-bit examples:**
  \[
  110110 \Rightarrow -6, \quad 110100 \Rightarrow -4, \quad 111100 \Rightarrow -2, \quad 000000 \Rightarrow 0
  \]
- **1’s complement number in 8-bit examples:**
  \[
  11111110 \Rightarrow -1, \quad 00000001 \Rightarrow 1
  \]
- **Algorithm (circuit) for addition depends on the arguments’ signs.**
2’s Complement Representation for Integers

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

6-bit examples:

010110 \(_2\) = 22 \(_{10}\)
001010 \(_2\) = -22 \(_{10}\)
000000 \(_2\) = 0 \(_{10}\)
111111 \(_2\) = -1 \(_{10}\)

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 \(_{10}\); 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

\[ 14_{10} = 001110, \quad -14_{10} = 110010, \quad \text{add 1} \]

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

Example

\[ 18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10} + 110010_{2} = 000100_{2} \]

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

Example

\[ 14_{10} = 001110, \text{in 6-bit representation.} \\
14_{10} = 00000000111110, \text{in 12-bit representation} \\
-14_{10} = 110010, \text{in 6-bit representation} \\
-14_{10} = 111111110010, \text{in 12-bit representation.} \]

What About Non-integer Numbers?

- There are infinitely many real numbers between two integers
  - Many important numbers are real
    - speed of light \( \approx 3 \times 10^8 \)
  - 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 ≤ 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

Floating Point Representation

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

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

\[ -0.75_{10} = (-1) \times 1.1 \times 2^{-1} \]

\[ S = 1; \quad E = 126 \]

\[ M = 100000000000000000000000 \]

Example:

What floating-point number is: \(0xC1580000?\)

Floating Point Representation

- Double Precision Floating point:
  
  64-bit representation: 1-bit sign, 11-bit (biased) exponent; 52-bit mantissa (with hidden 1).

\[ X = (-1)^S \cdot M \cdot 2^{E-1023} \]

Double precision floating point number

<table>
<thead>
<tr>
<th>S</th>
<th>Exp</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11-bit</td>
<td>20-bit</td>
</tr>
<tr>
<td></td>
<td>32-bit</td>
<td></td>
</tr>
</tbody>
</table>

ASCII Character Representation

Oct. Ch.

<table>
<thead>
<tr>
<th>Oct.</th>
<th>Ch</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>\nul</td>
</tr>
<tr>
<td>011</td>
<td>\soh</td>
</tr>
<tr>
<td>012</td>
<td>\stx</td>
</tr>
<tr>
<td>013</td>
<td>\etx</td>
</tr>
<tr>
<td>014</td>
<td>\eot</td>
</tr>
<tr>
<td>015</td>
<td>\enq</td>
</tr>
<tr>
<td>016</td>
<td>\ack</td>
</tr>
<tr>
<td>017</td>
<td>\bel</td>
</tr>
<tr>
<td>018</td>
<td>\bs</td>
</tr>
<tr>
<td>019</td>
<td>\ht</td>
</tr>
<tr>
<td>020</td>
<td>\nl</td>
</tr>
<tr>
<td>021</td>
<td>\vt</td>
</tr>
<tr>
<td>022</td>
<td>\np</td>
</tr>
<tr>
<td>023</td>
<td>\cr</td>
</tr>
<tr>
<td>024</td>
<td>\so</td>
</tr>
<tr>
<td>025</td>
<td>\si</td>
</tr>
<tr>
<td>026</td>
<td>\dle</td>
</tr>
<tr>
<td>027</td>
<td>\dc1</td>
</tr>
<tr>
<td>028</td>
<td>\dc2</td>
</tr>
<tr>
<td>029</td>
<td>\dc3</td>
</tr>
<tr>
<td>030</td>
<td>\dc4</td>
</tr>
<tr>
<td>031</td>
<td>\nak</td>
</tr>
<tr>
<td>032</td>
<td>\syn</td>
</tr>
<tr>
<td>033</td>
<td>\etb</td>
</tr>
<tr>
<td>034</td>
<td>\can</td>
</tr>
<tr>
<td>035</td>
<td>\em</td>
</tr>
<tr>
<td>036</td>
<td>\sub</td>
</tr>
<tr>
<td>037</td>
<td>\esc</td>
</tr>
<tr>
<td>038</td>
<td>\fs</td>
</tr>
<tr>
<td>039</td>
<td>\gs</td>
</tr>
<tr>
<td>040</td>
<td>\rs</td>
</tr>
<tr>
<td>041</td>
<td>\us</td>
</tr>
<tr>
<td>042</td>
<td>\sp</td>
</tr>
<tr>
<td>043</td>
<td>!</td>
</tr>
<tr>
<td>044</td>
<td>&quot;</td>
</tr>
<tr>
<td>045</td>
<td>#</td>
</tr>
<tr>
<td>046</td>
<td>$</td>
</tr>
<tr>
<td>047</td>
<td>%</td>
</tr>
<tr>
<td>048</td>
<td>&amp;</td>
</tr>
<tr>
<td>049</td>
<td>'</td>
</tr>
<tr>
<td>050</td>
<td>(</td>
</tr>
<tr>
<td>051</td>
<td>)</td>
</tr>
<tr>
<td>052</td>
<td>*</td>
</tr>
<tr>
<td>053</td>
<td>+</td>
</tr>
<tr>
<td>054</td>
<td>,</td>
</tr>
<tr>
<td>055</td>
<td>-</td>
</tr>
<tr>
<td>056</td>
<td>.</td>
</tr>
<tr>
<td>057</td>
<td>/</td>
</tr>
<tr>
<td>058</td>
<td>0</td>
</tr>
<tr>
<td>059</td>
<td>1</td>
</tr>
<tr>
<td>060</td>
<td>2</td>
</tr>
<tr>
<td>061</td>
<td>3</td>
</tr>
<tr>
<td>062</td>
<td>4</td>
</tr>
<tr>
<td>063</td>
<td>5</td>
</tr>
<tr>
<td>064</td>
<td>6</td>
</tr>
<tr>
<td>065</td>
<td>7</td>
</tr>
<tr>
<td>066</td>
<td>8</td>
</tr>
<tr>
<td>067</td>
<td>9</td>
</tr>
<tr>
<td>068</td>
<td>:</td>
</tr>
<tr>
<td>069</td>
<td>;</td>
</tr>
<tr>
<td>070</td>
<td>&lt;</td>
</tr>
<tr>
<td>071</td>
<td>=</td>
</tr>
<tr>
<td>072</td>
<td>&gt;</td>
</tr>
<tr>
<td>073</td>
<td>?</td>
</tr>
<tr>
<td>074</td>
<td>@</td>
</tr>
<tr>
<td>075</td>
<td>A</td>
</tr>
<tr>
<td>076</td>
<td>B</td>
</tr>
<tr>
<td>077</td>
<td>C</td>
</tr>
<tr>
<td>078</td>
<td>D</td>
</tr>
<tr>
<td>079</td>
<td>E</td>
</tr>
<tr>
<td>080</td>
<td>F</td>
</tr>
<tr>
<td>081</td>
<td>G</td>
</tr>
<tr>
<td>082</td>
<td>H</td>
</tr>
<tr>
<td>083</td>
<td>I</td>
</tr>
<tr>
<td>084</td>
<td>J</td>
</tr>
<tr>
<td>085</td>
<td>K</td>
</tr>
<tr>
<td>086</td>
<td>L</td>
</tr>
<tr>
<td>087</td>
<td>M</td>
</tr>
<tr>
<td>088</td>
<td>N</td>
</tr>
<tr>
<td>089</td>
<td>O</td>
</tr>
<tr>
<td>090</td>
<td>P</td>
</tr>
<tr>
<td>091</td>
<td>Q</td>
</tr>
<tr>
<td>092</td>
<td>R</td>
</tr>
<tr>
<td>093</td>
<td>S</td>
</tr>
<tr>
<td>094</td>
<td>T</td>
</tr>
<tr>
<td>095</td>
<td>U</td>
</tr>
<tr>
<td>096</td>
<td>V</td>
</tr>
<tr>
<td>097</td>
<td>W</td>
</tr>
<tr>
<td>098</td>
<td>X</td>
</tr>
<tr>
<td>099</td>
<td>Y</td>
</tr>
<tr>
<td>100</td>
<td>Z</td>
</tr>
<tr>
<td>101</td>
<td>[</td>
</tr>
<tr>
<td>102</td>
<td>]</td>
</tr>
<tr>
<td>103</td>
<td>^</td>
</tr>
<tr>
<td>104</td>
<td>_</td>
</tr>
<tr>
<td>105</td>
<td>`</td>
</tr>
<tr>
<td>106</td>
<td>a</td>
</tr>
<tr>
<td>107</td>
<td>b</td>
</tr>
<tr>
<td>108</td>
<td>c</td>
</tr>
<tr>
<td>109</td>
<td>d</td>
</tr>
<tr>
<td>110</td>
<td>e</td>
</tr>
<tr>
<td>111</td>
<td>f</td>
</tr>
<tr>
<td>112</td>
<td>g</td>
</tr>
<tr>
<td>113</td>
<td>h</td>
</tr>
<tr>
<td>114</td>
<td>i</td>
</tr>
<tr>
<td>115</td>
<td>j</td>
</tr>
<tr>
<td>116</td>
<td>k</td>
</tr>
<tr>
<td>117</td>
<td>l</td>
</tr>
<tr>
<td>118</td>
<td>m</td>
</tr>
<tr>
<td>119</td>
<td>n</td>
</tr>
<tr>
<td>120</td>
<td>o</td>
</tr>
<tr>
<td>121</td>
<td>p</td>
</tr>
<tr>
<td>122</td>
<td>q</td>
</tr>
<tr>
<td>123</td>
<td>r</td>
</tr>
<tr>
<td>124</td>
<td>s</td>
</tr>
<tr>
<td>125</td>
<td>t</td>
</tr>
<tr>
<td>126</td>
<td>u</td>
</tr>
<tr>
<td>127</td>
<td>v</td>
</tr>
<tr>
<td>128</td>
<td>w</td>
</tr>
<tr>
<td>129</td>
<td>x</td>
</tr>
<tr>
<td>130</td>
<td>y</td>
</tr>
<tr>
<td>131</td>
<td>z</td>
</tr>
<tr>
<td>132</td>
<td>{</td>
</tr>
<tr>
<td>133</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>}</td>
</tr>
<tr>
<td>135</td>
<td>~</td>
</tr>
</tbody>
</table>

Basic Data Types

- **Bit**: 0, 1
- **Bit strings**: sequence of bits of a particular length
- **8 bits is a byte**
- **16 bits is a word**
- **32 bits is a double-word**
- **Character**: 7-bit code
- **Decimal (BCD code)**
  
  digits 0 to 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