Lecture #2 – Number Rep & Intro to C

2005-08-31

There is one handout today at the front and back of the room!

Lecturer PSOE, new dad Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Time Lapse!

In the next 4 yrs, time-lapse movies will show the construction of the new CITRIS building. Very cool.

www.cs.berkeley.edu/~ddgarcia/tl/
Review

• Continued rapid improvement in computing
  • 2X every 2.0 years in memory size;
    every 1.5 years in processor speed;
    every 1.0 year in disk capacity;
  • Moore’s Law enables processor
    (2X transistors/chip ~1.5 yrs)

• 5 classic components of all computers
  Control Datapath Memory Input Output

Processor

• Decimal for human calculations, binary for computers, hex to write binary more easily
Putting it all in perspective…

“If the automobile had followed the same development cycle as the computer,

– Robert X. Cringely
What to do with representations of numbers?

• Just what we do with numbers!
  • Add them
  • Subtract them
  • Multiply them
  • Divide them
  • Compare them

• Example: 10 + 7 = 17
  • …so simple to add in binary that we can build circuits to do it!
  • subtraction just as you would in decimal
  • Comparison: How do you tell if X > Y ?
Which base do we use?

- Decimal: great for humans, especially when doing arithmetic
- Hex: if human looking at long strings of binary numbers, its much easier to convert to hex and look 4 bits/symbol
  - Terrible for arithmetic on paper
- Binary: what computers use; you will learn how computers do +, -, *, /
  - To a computer, numbers always binary
  - Regardless of how number is written:
    - $32_{\text{ten}} = 32_{10} = 0x20 = 100000_2 = 0b100000$
  - Use subscripts “ten”, “hex”, “two” in book, slides when might be confusing
BIG IDEA: Bits can represent anything!!

- Characters?
  - 26 letters $\Rightarrow$ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation $\Rightarrow$ 7 bits (in 8) ("ASCII")
  - standard code to cover all the world’s languages $\Rightarrow$ 8,16,32 bits ("Unicode")
    www.unicode.com

- Logical values?
  - 0 $\Rightarrow$ False, 1 $\Rightarrow$ True

- colors? Ex: Red (00) Green (01) Blue (11)

- locations / addresses? commands?

- MEMORIZE: N bits $\Leftrightarrow$ at most $2^N$ things
How to Represent Negative Numbers?

• So far, **unsigned numbers**

• Obvious solution: define leftmost bit to be sign!
  • 0 ⇒ +, 1 ⇒ –
  • Rest of bits can be numerical value of number

• Representation called **sign and magnitude**

• MIPS uses 32-bit integers. $+1_{\text{ten}}$ would be:
  \[
  0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001
  \]

• And $-1_{\text{ten}}$ in sign and magnitude would be:
  \[
  1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001
  \]
Shortcomings of sign and magnitude?

• Arithmetic circuit complicated
  • Special steps depending whether signs are the same or not

• Also, **two** zeros
  • $0x00000000 = +0_{ten}$
  • $0x80000000 = -0_{ten}$
  • What would two 0s mean for programming?

• Therefore sign and magnitude abandoned
Administrivia

• Look at class website often!

• **Homework #1** up now, due Wed @ 11:59pm

• **Homework #2** up soon, due following Wed

• There’s a LOT of reading upcoming -- start now.
Another try: complement the bits

- Example: \(7_{10} = 00111_2\), \(-7_{10} = 11000_2\)
- Called One’s Complement
- Note: positive numbers have leading 0s, negative numbers have leading 1s.

- What is -00000? Answer: 11111
- How many positive numbers in N bits?
- How many negative ones?
Shortcomings of One’s complement?

- Arithmetic still a somewhat complicated.
- Still two zeros
  - $0x00000000 = +0_{\text{ten}}$
  - $0xFFFFFFF = -0_{\text{ten}}$
- Although used for awhile on some computer products, one’s complement was eventually abandoned because another solution was better.
Standard Negative Number Representation

• What is result for unsigned numbers if tried to subtract large number from a small one?
  • Would try to borrow from string of leading 0s, so result would have a string of leading 1s
    - \( 3 - 4 \Rightarrow 00\ldots0011 - 00\ldots0100 = 11\ldots1111 \)

• With no obvious better alternative, pick representation that made the hardware simple

• As with sign and magnitude, leading 0s \(\Rightarrow\) positive, leading 1s \(\Rightarrow\) negative
  - \(000000\ldots\text{xxx} \) is \(\geq 0\), \(111111\ldots\text{xxx} \) is \(< 0\)
  - except \(1\ldots1111\) is -1, not -0 (as in sign & mag.)

• This representation is **Two’s Complement**
2’s Complement Number “line”: N = 5

- $2^{N-1}$ non-negatives
- $2^{N-1}$ negatives
- One zero
- How many positives?

00000 00001 ... 01111
10000 ... 11110 11111
## Two’s Complement for N=32

<table>
<thead>
<tr>
<th>Two’s Complement</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 ... 0000 0000 0000 0000&lt;sub&gt;two&lt;/sub&gt;</td>
<td>0&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0001&lt;sub&gt;two&lt;/sub&gt;</td>
<td>1&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0010&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1101&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,645&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1110&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,646&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1111&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,647&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0000&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-2,147,483,648&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0001&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-2,147,483,647&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0010&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-2,147,483,646&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1101&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-3&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1110&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-2&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1111&lt;sub&gt;two&lt;/sub&gt;</td>
<td>-1&lt;sub&gt;ten&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

- One zero; 1st bit called **sign bit**
- 1 “extra” negative: no positive \(2,147,483,648<sub>ten</sub}\
Two’s Complement Formula

- Can represent positive and negative numbers in terms of the bit value times a power of 2:
  \[ d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + ... + d_{2} \times 2^{2} + d_{1} \times 2^{1} + d_{0} \times 2^{0} \]

- Example: 1101_{two}
  \[= 1 \times -(2^{3}) + 1 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} \]
  \[= -2^{3} + 2^{2} + 0 + 2^{0} \]
  \[= -8 + 4 + 0 + 1 \]
  \[= -8 + 5 \]
  \[= -3_{ten} \]
Two’s Complement shortcut: Negation

*Check out www.cs.berkeley.edu/~dsw/twos_complement.html

• Change every 0 to 1 and 1 to 0 (invert or complement), then add 1 to the result

• Proof*: Sum of number and its (one’s) complement must be 111...111<sub>two</sub>

  However, 111...111<sub>two</sub> = -1<sub>ten</sub>

  Let x’ ⇒ one’s complement representation of x

  Then x + x’ = -1 ⇒ x + x’ + 1 = 0 ⇒ x’ + 1 = -x

• Example: -3 to +3 to -3

| x  | 1111 1111 1111 1111 1111 1111 1111 1101<sub>two</sub> |
| x’ | 0000 0000 0000 0000 0000 0000 0000 0010<sub>two</sub> |
| +1 | 0000 0000 0000 0000 0000 0000 0000 0011<sub>two</sub> |
| ()’| 1111 1111 1111 1111 1111 1111 1111 1100<sub>two</sub> |
| +1 | 1111 1111 1111 1111 1111 1111 1111 1101<sub>two</sub> |

You should be able to do this in your head...
Two’s comp. shortcut: Sign extension

- Convert 2’s complement number rep. using n bits to more than n bits
- Simply **replicate** the most significant bit (sign bit) of smaller to fill new bits
  - 2’s comp. positive number has infinite 0s
  - 2’s comp. negative number has infinite 1s
- Binary representation hides leading bits; sign extension restores some of them
- 16-bit $-4_{\text{ten}}$ to 32-bit:
  \[
  \begin{align*}
  &1111 1111 1111 1100_{\text{two}} \\
  &1111 1111 1111 1111 1111 1111 1100_{\text{two}}
  \end{align*}
  \]
What if too big?

• Binary bit patterns above are simply representatives of numbers. Strictly speaking they are called “numerals”.

• Numbers really have an $\infty$ number of digits
  • with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  • Just don’t normally show leading digits

• If result of add (or -, *, /) cannot be represented by these rightmost HW bits, overflow is said to have occurred.
Preview: Signed vs. Unsigned Variables

• Java and C declare integers `int`
  • Use two’s complement (`signed integer`)

• Also, C declaration `unsigned int`
  • Declares a `unsigned` integer
  • Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit
Number summary...

- We represent “things” in computers as particular bit patterns: \( N \) bits \( \Rightarrow 2^N \)

- Decimal for human calculations, binary for computers, hex to write binary more easily

- 1’s complement - mostly abandoned

\[
\begin{array}{cccc}
00000 & 00001 & \ldots & 01111 \\
10000 & \ldots & 11110 & 11111 \\
\end{array}
\]

- 2’s complement universal in computing: cannot avoid, so learn

\[
\begin{array}{cccc}
00000 & 00001 & \ldots & 01111 \\
10000 & \ldots & 11110 & 11111 \\
\end{array}
\]

Overflow: numbers \( \infty \); computers finite, errors!
Peer Instruction Question

X = 1111 1111 1111 1111 1111 1111 1111 1100\text{two}
Y = 0011 1011 1001 1010 1000 1010 0000 0000\text{two}

A. X > Y (if signed)
B. X > Y (if unsigned)
C. An encoding for Babylonians could have $2^N$ non-zero numbers w/N bits!
Administrivia: Near term

- **Upcoming lectures** [Monday is a holiday!]
  - C pointers and arrays in detail

- **Lab tomorrow**
  - We’ll ask you to sign a document saying you understand the cheating policy (from Lec #1) and agree to abide by it.

- **HW**
  - HW0 due in discussion next week
  - HW1 due next Wed @ 23:59 PST
  - HW2 due following Wed @ 23:59 PST

- **Reading**
  - K&R Chapters 1-5 (lots, get started now!); 1st quiz due Sun!

- **Get cardkeys from CS main office Soda Hall 3rd fl**
  - Soda locks doors @ 6:30pm & on weekends

- **CSUA Info-session (Th 6-7pm, 310 Soda, “Inst env”)**
  - Following Th will be “Intro to Emacs” @ 5pm in 310 Soda
Disclaimer

• **Important:** You will not learn how to fully code in C in these lectures! You’ll still need your C reference for this course.
  
  - K&R is a great reference.
  - But... check online for more sources.
  
  • “JAVA in a Nutshell” – O’Reilly.
  - Chapter 2, “How Java Differs from C”.
Compilation: Overview

C compilers take C and convert it into an architecture specific machine code (string of 1s and 0s).

• Unlike Java which converts to architecture independent bytecode.

• Unlike most Scheme environments which interpret the code.

• Generally a 2 part process of compiling .c files to .o files, then linking the .o files into executables.
Compilation: Advantages

- **Great run-time performance**: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)

- **OK compilation time**: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled
Compilation: Disadvantages

• All compiled files (including the executable) are **architecture specific**, depending on **both** the CPU type and the operating system.

• Executable must be **rebuilt** on each new system.
  • Called “**porting your code**” to a new architecture.

• The “**change→compile→run [repeat]”** iteration cycle is slow
C vs. Java™ Overview (1/2)

Java
  • Object-oriented (OOP)
  • “Methods”
  • Class libraries of data structures
  • Automatic memory management

C
  • No built-in object abstraction. Data separate from methods.
  • “Functions”
  • C libraries are lower-level
  • Manual memory management
  • Pointers
Java
- **High** memory overhead from class libraries
- **Relatively Slow**
- Arrays initialize to **zero**
- Syntax:
  ```
  /* comment */
  // comment
  System.out.print
  ```

C
- **Low** memory overhead
- **Relatively Fast**
- Arrays initialize to **garbage**
- Syntax:
  ```
  /* comment */
  printf
  ```
C Syntax: Variable Declarations

- Very similar to Java, but with a few minor but important differences

- All variable declarations must go before they are used (at the beginning of the block).

- A variable may be initialized in its declaration.

- Examples of declarations:
  - correct: 
    ```
    int a = 0, b = 10;
    ...
    ```
  - incorrect: 
    ```
    for (int i = 0; i < 10; i++)
    ```
C Syntax: True or False?

• What evaluates to FALSE in C?
  • 0 (integer)
  • NULL (pointer: more on this later)
  • no such thing as a Boolean

• What evaluates to TRUE in C?
  • everything else…
  • (same idea as in scheme: only #f is false, everything else is true!)
C syntax : flow control

• Within a function, remarkably **close to Java** constructs in methods (shows its legacy) in terms of flow control
  • if-else
  • switch
  • while and for
  • do-while
C Syntax: main

• To get the main function to accept arguments, use this:
  
  ```c
  int main (int argc, char *argv[])
  ```

• What does this mean?
  
  • `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument).
    
    - Example: `unix% sort myFile`

  • `argv` is a pointer to an array containing the arguments as strings (more on pointers later).
Address vs. Value

• Consider memory to be a single huge array:
  • Each cell of the array has an address associated with it.
  • Each cell also stores some value
  • Do you think they use signed or unsigned numbers? Negative address?!

• Don’t confuse the address referring to a memory location with the value stored in that location.

  101 102 103 104 105 ...
  ...
  23 42
Pointers

• An address refers to a particular memory location. In other words, it points to a memory location.

• **Pointer**: A variable that contains the address of another variable.
Pointers

• How to create a pointer:

   & operator: get address of a variable

   int *p, x;
   p ? x ?

   x = 3;
   p ? x 3

   p = &x;
   p x 3

• How get a value pointed to?

   * “dereference operator”: get value pointed to

   printf(“p points to %d\n”, *p);
Pointers

• How to change a variable pointed to?
  • Use dereference * operator on left of =

*p = 5;

p x 3

p x 5
Pointers and Parameter Passing

• Java and C pass a parameter “by value”

  • procedure/function gets a copy of the parameter, so changing the copy cannot change the original

```c
void addOne (int x) {
    x = x + 1;
}

int y = 3;
addOne (y);
```

• y is still = 3
Pointers and Parameter Passing

- How to get a function to change a value?

```c
void addOne (int *p) {
    *p = *p + 1;
}

int y = 3;

addOne (&y);

• y is now = 4
```
Pointers

• Normally a pointer can only point to one type (int, char, a struct, etc.).
  • `void *` is a type that can point to anything (generic pointer)
  • Use sparingly to help avoid program bugs... and security issues... and a lot of other bad things!
void main(); {  
    int *p, x=5, y; // init  
y = *(p = &x) + 10;  
    int z;  
    flip-sign(p);  
    printf("x=%d,y=%d,p=%d\n",x,y,p);  
}  
flip-sign(int *n){*n = -(*n)}

How many errors?
void main() {  
    int *p, x=5, y; // init
    y = *(p = &x) + 10;
    int z;
    flip-sign(p);
    printf("x=%d,y=%d,p=%d\n",x,y,*p);
}
flip-sign(int *n){*n = -(*n);}

And in conclusion...

• All declarations go at the beginning of each function.

• Only 0 and NULL evaluate to FALSE.

• All data is in memory. Each memory location has an address to use to refer to it and a value stored in it.

• A pointer is a C version of the address.
  • * “follows” a pointer to its value
  • & gets the address of a value