Review

- CS61C: Learn 6 great ideas in computer architecture to enable high performance programming via parallelism, not just learn C
  1. Abstraction (Layers of Representation/Interpretation)
  2. Moore’s Law
  3. Principle of Locality/Memory Hierarchy
  4. Parallelism
  5. Performance Measurement and Improvement
  6. Dependability via Redundancy

Putting it all in perspective...

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

— Robert X. Cringely

Data input: Analog → Digital

- Real world is analog!
- To import analog information, we must do two things
  1. Sample
     - E.g., for a CD, every 44,100ths of a second, we ask a music signal how loud it is.
  2. Quantize
     - For every one of these samples, we figure out where, on a 16-bit (65,536 tic-mark) “yardstick”, it lies.

Digital data not nec born Analog...

BIG IDEA: Bits can represent anything!!

- Characters?
  1. 26 letters → 5 bits ($2^5 = 32$)
  2. upper/lower case + punctuation → 7 bits (in 8) (“ASCII”)
  3. standard code to cover all the world’s languages → 8,16,32 bits (“Unicode”) www.unicode.com

- Logical values?
  1. 0 → False, 1 → True

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

- locations / addresses? commands?

- MEMORIZE: N bits ⇔ at most $2^N$ things
How many bits to represent $\pi$?

- a) 1
- b) 9 ($\pi = 3.14$, so that’s 011 “.” 001 100)
- c) 64 (Since Macs are 64-bit machines)
- d) Every bit the machine has!
- e) $\infty$

What to do with representations of numbers?

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

Example: $10 + 7 = 17$

What if too big?

- Binary bit patterns above are simply representatives of numbers. Abstraction! 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.

How to Represent Negative Numbers?

- (C’s unsigned int, C99's uintN_t)
- So far, unsigned numbers

<table>
<thead>
<tr>
<th>binary</th>
<th>00000</th>
<th>00001</th>
<th>00010</th>
<th>11110</th>
<th>11111</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned</td>
<td>00000</td>
<td>00001</td>
<td>00010</td>
<td>11110</td>
<td>11111</td>
</tr>
</tbody>
</table>

Obvious solution: define leftmost bit to be sign!
- 0 $\Rightarrow$ +
- 1 $\Rightarrow$ –
- Rest of bits can be numerical value of number
- Representation called **sign and magnitude**

How to Represent Negative Numbers?

1111 1000 0000 0001 0111 1000 0000 1111

META: Ain’t no free lunch

Shortcomings of sign and magnitude?

- Arithmetic circuit complicated
  - Special steps depending whether signs are the same or not
- Also, two zeros
  - 0x00000000 = $+0_{\text{ten}}$
  - 0x80000000 = $-0_{\text{ten}}$
- What would two 0s mean for programming?
- Also, incrementing “binary odometer”, sometimes increases values, and sometimes decreases!

Therefore sign and magnitude abandoned

Administrivia

- Upcoming lectures
  - Next few lectures: Introduction to C
- Lab overcrowding
  - Remember, you can go to ANY discussion (none, or one that doesn’t match with lab, or even more than one if you want)
  - Overcrowded labs - consider finishing at home and getting checkoffs in lab, or bringing laptop to lab
  - TAs get 24x7 cardkey access (and will announce after-hours times)
  - Enrollment
  - Soda locks doors @ 6:30pm & on weekends
- Look at class website, piazza often!
  - inst.eecs.berkeley.edu/~cs61c/piazza.com
Great DeCal courses I supervise

- UCBUGG (3 units, P/NP)
  - UC Berkeley Undergraduate Graphics Group
  - TuTh 7-9pm in 200 Sutardja Dai
  - Learn to create a short 3D animation
  - No prereqs (but they might have too many students, so admission not guaranteed)
  - http://ucbugg.berkeley.edu
- MS-DOS X (2 units, P/NP)
  - Macintosh Software Developers for OS X
  - MoWe 8-10pm in 200 Sutardja Dai
  - Learn to program iOS devices!
  - No prereqs (other than interest)
  - http://madosx.berkeley.edu

Example:

Can represent positive and negative numbers in terms of the bit value times a power of 2:

\[ d_3 x 2^3 + d_2 x 2^2 + d_1 x 2^1 + d_0 x 2^0 \]

Example: 1101 in a nibble?

- 1x(2^3) + 1x2^2 + 0x2^1 + 1x2^0
- = -8 + 4 + 0 + 1
- = -3

Two's Complement Formula

- Can represent positive and negative numbers in terms of the bit value times a power of 2:
  - \[ d_3 x 2^3 + d_2 x 2^2 + d_1 x 2^1 + d_0 x 2^0 \]
- Example: 1101 in a nibble?
  - = 1x(2^3) + 1x2^2 + 0x2^1 + 1x2^0
  - = -8 + 4 + 0 + 1
  - = -3

Shortcomings of One's complement?

- Arithmetic still a somewhat complicated.
- Still two zeros
  - \[ 0x00000000 = +0_{10} \]
  - \[ 0xFFFFFFF = -0_{10} \]
- Although used for a while on some computer products, one's complement was eventually abandoned because another solution was better.

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 leadings 1s.

\[ \begin{array}{cccccc}
00000 & 00001 & \ldots & 01110 & 01111 \\
\end{array} \]

- What is -00000? Answer: 11111
- How many positive numbers in N bits?
- How many negative numbers?

Standard Negative # Representation

- Problem is the negative mappings "overlap" with the positive ones (the two 0s). Want to shift the negative mappings left by one.
  - Solution! For negative numbers, complement, then add 1 to the result
  - As with sign and magnitude, & one's compl. leading 0s \( \Rightarrow \) positive, leading 1s \( \Rightarrow \) negative
    - \[ 00000...xxx \] is \( \geq 0 \)
    - \[ 11111...xxx \] is \( < 0 \)
  - except \[ 1...1111 \] is -1, not -0 (as in sign & mag.)
  - This representation is Two's Complement
  - This makes the hardware simple!
    - (C's int, aka a "signed integer")
    - (Also C's short, long, ..., C99's intN_t)

Two's Complement Number "line": \( N = 5 \)

\[ \begin{array}{cccccccc}
11110 & 11111 & \ldots & 00001 & 00000 \\
\end{array} \]

- How many non-negatives
- How many 2^N-1 negatives
- How many ones
- How many positives?
Two’s Complement for N=32

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0000...0000</td>
<td>0</td>
</tr>
<tr>
<td>0000...0001</td>
<td>-1</td>
</tr>
<tr>
<td>0000...0010</td>
<td>-2</td>
</tr>
<tr>
<td>0000...1111</td>
<td>-2,147,483,647</td>
</tr>
<tr>
<td>0001...1111</td>
<td>-2,147,483,646</td>
</tr>
<tr>
<td>0100...1111</td>
<td>-2,147,483,645</td>
</tr>
<tr>
<td>0101...1111</td>
<td>-2,147,483,644</td>
</tr>
<tr>
<td>1000...1111</td>
<td>-2,147,483,643</td>
</tr>
<tr>
<td>1001...1111</td>
<td>-2,147,483,642</td>
</tr>
<tr>
<td>1111...1111</td>
<td>-1</td>
</tr>
</tbody>
</table>

• One zero; 1st bit called sign bit
• 1 “extra” negative: no positive 2,147,483,648

How best to represent -12.75?

a) 2s Complement (but shift binary pt)
b) Bias (but shift binary pt)
c) Combination of 2 encodings
d) Combination of 3 encodings
e) We can’t

Shifting binary point means “divide number by some power of 2. E.g.: 
1110 = 1011.02 so (1/4)10 = 2.7510 = 10.1102

References: 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:
  • To a computer, numbers always binary
  • Regardless of how number is written:
    • 32two = 32ten = 0x20 = 01000000
  • Use subscripts “ten”, “hex”, “two” in book, slides when might be confusing

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 -4ten to 32-bit: 1111 1111 1111 1100two

And in summary...

• We represent “things” in computers as particular bit patterns: N bits => 2N things
• These 5 integer encodings have different benefits; 1s complement and sign/mag have most problems.
  • unsigned (C99’s uintN_t):
    • 0000...0000 = 0
    • 0000...0001 = 1
    • 2’s complement (C99’s intN_t) universal, learn!
    • 0000...0001 = -1
  • Overflow: numbers =; computers finite, errors!

META: We often make design decisions to make HW simple

META: Ain’t no free lunch