Lecture 4: Binary

CS105: Great Insights in Computer Science
Michael L. Littman, Fall 2006

Counting (Decimal)

- How do we count?
- Start at the bottom digit.
  - If it’s less than 9, add one to it.
  - If it’s equal to nine, make it zero and proceed to the digit to the left.
Counting (Binary)

- Counting in binary is the same idea.
- Start at the bottom (rightmost) bit.
  - If it’s less than 1, add one to it.
  - If it’s equal to 1, make it zero and proceed to the bit to the left.

Place Values

- Because of the way counting works, we expand the representation by another bit for each power of 2.
- So, 11001001 is:
- 128+64+8+1=201
Number Magic

• How does this trick work?
• http://www.brainbashers.com/games/number.asp

Which Have Your Number?

• Think of a number from 0 to 31.
• Add the upper left number from each card your number appears on.
• It is...

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 16| 17| 18| 19| | 8 | 9 | 10| 11| | 4 | 5 | 6 | 7 | | 2 | 3 | 6 | 7 | | 1 | 3 | 5 | 7 | | 9 | 11| 13| 15| |
| 20| 21| 22| 23| | 12| 13| 14| 15| | 12| 13| 14| 15| | 10| 11| 14| 15| | 18| 19| 22| 23| | 17| 19| 21| 23| | 25| 27| 29| 31| |
Conversion
• To go from decimal to binary, start with the biggest power of 2 no bigger than your number.
• Write down a 1. Subtract the power of 2 from your number.
• Cut the power of 2 in half.
  It’s a bit like making change.
• If your remaining number is larger than the power of 2, write down a 1 and subtract the power of 2.
• If not, write down 0.
• Repeat by cutting the power of 2 in half (until you get to 1).

Example: Convert 651
• Bigger than: $2^9 = 512$. 1
  • 651-512=139.
  • Next power of 2 = 256. 0
  • Next power of 2 = 128. 1
    • 139-128=11.
    • Next power of 2 = 64. 0
    • Next power of 2 = 32. 0
      • Last power of 2 = 1. 1
  1010001011=651
Binary Addition

\[
\begin{array}{c}
01110011 \\
+ 10110010 \\
\hline
11100110
\end{array}
\]

- Just like in school: work right to left, carry when needed.
- \(0+0+0=0, 0+0+1 = 1, 0+1+1=10, 1+1+1=11\)
- Can check via conversion.

Other Operations

- Can also define subtraction (with borrowing), multiplication (simpler since there are only 3 facts: \(0\times0=0, 0\times1=0, 1\times1=1\), look familiar?), and long division.
- Can do bitwise logic operations (and, or, not).
- All are quite useful...
Other Number Schemes

- Can represent negative numbers, often via twos complements. \(-1 = 256-1 = 255\).
- Fixed-width fractions (for dollar amounts).
- Floating point representations via exponential notation: \(a \times 10^b\).
- Complex numbers: real and imaginary parts. They are just bits: you can use them as you see fit.

Implementing Addition

- Half adder: Takes two bits and a carry and outputs a bit and a carry (addc).
- Adder: Adds two 8-bit numbers (discards last carry) (addbyte).

```
def addc(a, b, c):
    bit = (a and not b and not c) or (not a and b and not c) or (not a and not b and c) or (a and b and c)
    carry = (a and b and not c) or (not a and not b and c) or (not a and b and c) or (a and b and c)
    return [carry, bit]

def addbyte(x, y):
    z = [0]*8
    sum7 = addc(x[7], y[7], 0)
    z[7] = sum7[1]
    sum6 = addc(x[6], y[6], sum7[0])
    sum5 = addc(x[5], y[5], sum6[0])
    sum4 = addc(x[4], y[4], sum5[0])
    sum3 = addc(x[3], y[3], sum4[0])
    sum2 = addc(x[2], y[2], sum3[0])
    sum1 = addc(x[1], y[1], sum2[0])
    z[1] = sum1[1]
    sum0 = addc(x[0], y[0], sum1[0])
    z[0] = sum0[1]
    return z
```
Next Time

• We now have all the pieces to build a simple, working computer...
• Each cycle, inputs propagate to outputs, which are copied back to inputs to begin again.
• We need a language to talk to it in, though.
• Read Hillis Chapter 2 Section 3, Chapter 3 Sections 1-5.