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.

2 4 9 9 8

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.

0 0 0 0 1

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

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	0	1	0	0	1

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
20	21	22	23
24	25	26	27
28	29	30	31

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31

1	3	5	7
9	11	13	15
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.
- 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).

It's a bit like making change.

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
- Next power of 2 = 16. 0
- Next power of 2 = 8. 1
  - 11-8=3
- Next power of 2 = 4. 0
- Next power of 2 = 2. 1
  - 3-2=1
- Last power of 2 = 1. 1

1010001011=651

Binary Addition

```
  01110011      115
+ 10110010     +178
-----------    ----
  11001101      293
```

- 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: 0x0=0 0x1=0 1x1=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 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])
    z[6] = sum6[1]
    sum5 = addc(x[5],y[5],sum6[0])
    z[5] = sum5[1]
    sum4 = addc(x[4],y[4],sum5[0])
    z[4] = sum4[1]
    sum3 = addc(x[3],y[3],sum4[0])
    z[3] = sum3[1]
    sum2 = addc(x[2],y[2],sum3[0])
    z[2] = sum2[1]
    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
```

```
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 (a and not b and c) or (not a and b and c) or (a and b and c)
    return([carry, bit])
```

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.