CSE 378 Section 1
4/1/04

Announcements:

- sign up for mailing list, should have received assignment 1 correction.
- Typo on assignment 1: 0fff fff → 0fff ffff
- Check out SPIM, do assignment 0

Review of number systems

We normally count in the decimal, base-ten system.
Numbers broken down by digits:

E.g. $378.04 = 3 \times 10^2 + 7 \times 10^1 + 8 \times 10^0 + 0 \times 10^{-1} + 4 \times 10^{-2}$.

Computers use binary (base-two), because it's more convenient (why?).

Two digits: 0, 1. Example: $0010\ 1010_2 = 42_{10}$.

Addition/subtraction works just like decimal:
0+0 = 0   1+0 = 0+1 = 1   1+1 = 0 with a carry of 1

Examples:

```
  1010      1101
+ 0011    - 0111
------    ------
  1101      0110
```

LSB = least-significant bit
MSB = most-significant bit

Binary to decimal conversion:
Break numbers down using the same formula as for decimal, but using 2 as the base.

$$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}$$
$$101010_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

With fractions:

$$10.101_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 2.625_{10}$$

Decimal to binary conversion
Repeatedly divide by 2; write down the remainders. Keep doing this until you divide 1 by 2. When finished, read remainders in reverse order to get the answer. Example:

```
44 / 2 = 22      remainder 0 LSB
22 / 2 = 11      remainder 0
11 / 2 = 5       remainder 1
5  / 2 = 2       remainder 1
2  / 2 = 1       remainder 0
1  / 2 = 0       remainder 1 MSB   read this way
```

Answer: 101100

<u>Octal and Hexadecimal systems – compact representation for binary</u>
Octal = base 8; digits 0..7.
Hexadecimal (hex) = base 16; digits 0..9 then A..F. Each hex digit is four bits in binary.
Used for convenience (e.g. writing out 32 bits is a pain!)
Conversion from binary – just break bits in groups of three for octal and groups of four for hex; each group is one digit.

Example:

10011110001_2 to octal and hex:
010 | 011 | 110 | 001 = 2361_8 and 0100 | 1111 | 0001 = $4F1_{16}$

Decimal to hex: use same process as for binary to hex, but divide by 16.

Hex to binary: easy! For each digit, just write its 4-bit binary equivalent (see chart)

Quick reference for conversion:

Binary	**Decimal**	**Hex**	**Binary**	**Decimal**	**Hex**
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

<u>Negative numbers</u>

Generally three forms:

- *Sign & Magnitude*
  Reserve one bit as sign bit. Not the best way (why? *two zeros, arithmetic broken*)

- *Ones-complement*
  The binary representation of a negative number is the bitwise complement of the binary representation of the positive number (i.e. flip bits).

Example: $3_{10} = 11_2$; -3_{10} = complement (11_2) = 1100. Still not good (why? *still two zeros*).

- *Twos-complement*
  The binary representation of a negative number is the bitwise complement of the binary representation of the positive number, plus 1. Works for getting negatives of both positive numbers and negative numbers. Most significant bit = sign bit.

  -3_{10} = complement (0011_2) + 1_2 = $1100_2 + 1_2 = 1101_2$.
  $-(-3_{10})$ = compl. (-3_{10}) + 1_2 = compl. (1101_2) + 1 = 0010 + 1 = $0011_2 = 3_{10}$

  How do you convert a negative twos-complement number to decimal? And vice-versa?

Twos-complement arithmetic
Addition – same as regular binary addition. Subtraction – addition of the negative of the subtracted number.
Example: 4 – 3 becomes:

```
  0100
+ 1101
  0001   (discard final carry) = 1_10.
```

Overflow

The operations as defined above can overflow. For example, let's try (-7) + (-6), while limiting numbers to 4 bits…

```
  1001
+ 1010
  0011  (final carry discarded) = 3_10.
```

Definitely the wrong answer.

How do we know when overflow occurred?
Summing two positive numbers gives a negative result.
Summing two negative numbers gives a positive result.
Summing a positive and a negative number will never overflow. Why?

Sign-extension
Suppose we want to convert an 8-bit signed (twos-complement) number to a 16-bit signed number. Fill in lower bits from the original number; fill in the unused higher bits with the sign bit (i.e. 0 if number is positive; 1 if number is negative).
Example:
$0010\ 1010_2$ becomes $0000\ 0000\ 0010\ 1010_2$
$1110\ 1010_2$ becomes $1111\ 1111\ 1110\ 1010_2$

Number Ranges
A N-bit integer can represent 2^N different combinations. E.g. a 16-bit unsigned integer can represent 0 .. 2^{16}-1 which is 0 .. 65535. A 16-bit signed number can represent -2^{15} to 2^{15}-1, i.e. -32768 … 32767.