CSC 120(A) - Principles of Computer Science I
Prof. Nadeem Abdul Hamid
Fall 2004

Lecture Handout: Binary Numbers

1. Converting base-2 numbers to base-10

To figure out the value of a base-2 number, just take the sum of all the digits times their place values.

For example:

```
place values    8s  4s  2s  1s

                 0   1   1   0    =  0x8 + 1x4 + 1x2 + 1x0 = 4 + 2 = 6₁₀
```

2. Converting base-10 numbers to base-2

Follow this algorithm:

let our input be **n**, a base-10 number
our output will be a base-2 number, which initially has no digits

while (n is not zero) do the following
  divide n by 2
  let the remainder be the new left-most digit of the output
  set n to the quotient of the division
loop to the beginning of the while block

Example: Convert 14_{10} to binary

```
   Division                       Binary number output
 14/2 = 7 R 0                                0
  7/2 = 3 R 1                              1 0
  3/2 = 1 R 1                            1 1 0
  1/2 = 0 R 1                          1 1 1 0
```

The answer is 1110_2. (Try converting it back to base-10 to check.)

3. Binary addition...

... is just like decimal addition. Do the addition column by column and carry over if necessary.

Examples:

```
    1 0 0 1        9            |       1 0 1 1          11
  + 1 1 0 1     + 13            |     + 0 1 1 1       +   7
 ----------     ----            |    ----------       ----
  1 0 1 1 0       22            |     1 0 0 1 0          18
```

Note: The computer stores the data in a fixed number of bits, so in these examples, it would discard the overflow **1**s in both examples.

4. Negative numbers

Using 4 bits we have 16 possible combinations:

```
0000        1000
0001        1001
0010        1010
0011        1011
0100        1100
0101        1101
0110        1110
0111        1111
```

One possible way to represent negative numbers is to use the left-most bit as a sign bit, where `0`="+" and `1`="-". One problem with this approach is that there are two representations of zero: `0000` and `1000` (+0 and –0).

An alternative representation (and one that is used by the computer) is called **Two's complement representation**. The number line below shows which binary numbers are used to represent numbers in the range of –8 to +7.

```
 -8   -7   -6   -5   -4   -3   -2   -1    0   +1   +2   +3   +4   +5   +6   +7

--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-

1000 1001 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101 0110 0111
```

With this representation, addition and subtraction (addition of negative numbers) work very conveniently:

```
   1 0 0 0          - 8          |          0 1 1 0          + 6
 + 0 1 0 1          + 5          |        + 1 0 1 1          - 5
----------          ----         |        ----------         ----
   1 1 0 1          - 3          |        ~~1~~ 0 0 0 1          + 1
```

Notice the overflow 1 in the second example is just discarded. Notice also, the leftmost bit of negative numbers is always `1`.

5. Computing the negative of a two's complement number

Just invert the bits (replace `0`s with `1`s and vice versa) and add `1`.

Example:

5_{10} = 0 1 0 1

-5_{10} = 1 0 1 0

```
  1 0 1 0
+       1
---------
  1 0 1 1
```

6. Overflow

Sometimes overflow works just fine, like when adding positive and negative numbers. However, overflow can often be a problem when working with a computer and your data gets larger than the fixed number of bits. Thus, if you are not careful,

```
0 1 0 1  +  0 1 0 1  =  1 0 1 0     ( that is,  5 + 5 = -6 !!! )
```