SIENAcollege
Computer Science

Computer Science 010
Introduction to Computer Applications
Siena College
Fall 2010

Topic Notes: Binary Representations

Binary Basics

Question: how high can you count on one finger?

That finger can either be up or down, so you can count 0, 1 and that's it.

(Computer scientists always start counting at 0, so you should get used to that...)

So then... How high can you count on one hand/five fingers?

When kids count on their fingers, they can get up to 5. The number is represented by the number of fingers they have up.

But we have multiple ways to represent some of the numbers this way: 1 0, 5 1's, 10 2's, 10 3's, 5 4's and 1 5.

We can do better. We have 32 different combinations of fingers up or down, so we can use them to represent 32 different numbers.

Given that, how high can you count on ten fingers?

To make this work, we need to figure out which patterns of fingers up and down correspond to which numbers.

To keep things manageable, we'll assume we're working with 4 digits (hey, aren't fingers called digits too?) each of which can be a 0 or a 1. We should be able to represent 16 numbers. As computer scientists, we'll represent numbers from 0 to 15.

Our 16 patterns are base 2, or *binary*, numbers. In this case, we call the digits *bits* (short for **b**inary dig**its**).

Each bit may be 0 or 1. That's all we have.

Just like in base 10 (or *decimal*), where we have the 1's (10^0) place, the 10's (10^1) place, the 100's (10^2) place, etc, here we have the 1's (2^0), 2's (2^1), 4's (2^2), 8's (2^3), etc.

As you might imagine, binary representations require a lot of bits as we start to represent larger values. Since we will often find it convenient to think of our binary values in 4-bit chunks, we will also tend to use base 16 (or *hexadecimal*).

Since we don't have enough numbers to represent the 16 unique digits required, we use the numbers 0–9 for the values 0–9 but then the letters A–F to represent the values 10–15.

```
0  0000
```

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

Any number can be used as the base, but the common ones are base 2, base 8 (*octal*, 3 binary digits), base 10, and base 16.

A byte (8 bits) of data can be written as a decimal number in the range 0 to 255, as 8 binary bits, or as two hexadecimal symbols.

Two bytes of data: decimal in the range 0-65535, 16 bits, or 4 hex digits.

Decimal-Binary-Hex Conversions

Since we will encounter values in decimal, binary, and hexadecimal at various times, we should think about how to convert among these represenations.

We'll first look at the easiest case: converting back and forth between binary and hex. Each hex digit represents 4 bits and each group of 4 bits can be represented by a hex digit.

So to convert the hex value `7A4D` to binary, we convert each digit:

```
0111 1010 0100 1101
```

And to go back the other way, we can start with

```
0100 1011 0000 0110
```

(conveniently written in groups of 4 bits) to obtain `4B06` in hex.

Converting from binary or hex to decimal is also pretty straightforward. We just add up the values represented by each digit. Start with a binary value:

```
01011011
```

This binary string is interpreted:

$$0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

which is 64+16+8+2+1 = 91 in decimal.

Notice that this is **exactly** the same way we interpret a decimal value. The value 2872 in decimal is really:

$$2 \times 10^3 + 8 \times 10^2 + 7 \times 10^2 + 2 \times 10^0$$

It's a similar idea converting from hex. Here, we'll look at the place values in base 16. Starting with the hex value `30C7`:

$$3 \times 16^3 + 0 \times 16^2 + 12 \times 16^1 + 7 \times 16^0$$

We'll probably grab a calculator for this:

$$3 \times 4096 + 12 \times 16 + 7 = 12288 + 192 + 7 = 12487$$

The remaining conversions start from decimal. We'll first convert decimal to binary. We assume here that we are going to store our result in 8 bits, but the procedure can be extended to larger values.

We will use 108 (decimal) as our value.

Let's remember what each position in our 8-bit binary value represents:

The first bit is the number of $2^7 = 128$'s in the number.
The second bit is the number of $2^6 = 64$'s
The third bit is the number of $2^5 = 32$'s
The fourth bit is the number of $2^4 = 16$'s
The fifth bit is the number of $2^3 = 8$'s
The sixth bit is the number of $2^2 = 4$'s
The seventh bit is the number of $2^1 = 2$'s
The eighth bit is the number of $2^0 = 1$'s

We work in order from left to right. If the number is greater than or equal to the value stored in a position, we place a 1 in that position and subtract the value for that position from the number. Otherwise, we place a 0 in that position.

So for our number 108, we start with the 128's place. 108 is smaller, so we place a 0 there:

`0???????`

Next, we notice that 108 is larger than 64, so we place a 1 in the 64's place and subtract 64 from our number. Subsequent steps will work with the number 44.

```
01??????
```

44 is greater than 32, so we have a 1 in the 32's place, and are left with 12 to work with.

```
011?????
```

There are no 16's in 12, so we put a 0.

```
0110????
```

There is an 8 in 12, so we place a 1 and subtract, leaving us with 4.

```
01101???
```

And there is a 4, so we place a 1 in the 4's and are left with 0.

```
011011??
```

The 0 remaining will not contain any 2's or 1's, so we will be placing 0's in the final two positions.

```
01101100
```

And there's our answer.

We will follow a similar procedure to this to convert decimal to hex, but it's a little more tricky since we're dealing with powers of 16.

Here, we will assume that we want the answer in 4 hex digits, and we'll start with the decimal number 19,832.

First, we need to remind ourselves what the place values are in hex:

$16^3 = 4096$
$16^2 = 256$
$16^1 = 16$
$16^0 = 1$

So we begin by figuring out how many 4096's there are in 19,832. If we divide 19,832 by 4096, we get 4, with a remainder of 3448. So our first hex digit will be a 4.

```
4???
```

And we continue working with that remainder, 3448. Note that you can also think about subtracting $4 \times 4096 = 16384$ from our starting number to get that remainder.

How many 256's are there in 3448? Well, $\frac{3448}{256}$ gives us 13, with a remainder of 120. This means we need to use 13 as our second hex digit. Recall that the character we use to represent 13 is D.

```
4D??
```

Continuing on, we are left with 120 and we need to fill in the 16's place. $\frac{120}{16}$ gives 7 with a remainder of 8. So we have our last two digits, and the hex representation of our number is

```
4D78
```

It's All Binary

So we have seen how we can represent unsigned (*i.e.*, non-negative) whole numbers in binary. However, that is just one of many kinds of data that we will wish to store and process.

- Integers (including negative numbers): the representation is similar to what we use for unsigned numbers. The usual representation is called *2's Complement*, but the details are not our concern.
- "Real" numbers with fractions/decimal points: there are many choices of how to represent non-intergral values. The representations almost always used are called *floating point* representations. Not all values can be represented exactly – approximations are needed.
- Characters: representing text. The two most common mappings of binary values to characters:
  - The *American Standard Code for Information Interchange* (ASCII) – provides a set of one-byte representations for English characters, numerical digits, and common punctuation.

    See: `http://en.wikipedia.org/wiki/ASCII`
  - *Unicode* – introduced about 20 years ago – representations are 2 bytes per character, allowing the inclusion of many international characters.

    See: `http://en.wikipedia.org/wiki/List_of_Unicode_characters`

  Text such as that in the document you are reading now is represented as a sequence of ASCII or Unicode characters.

- Other media, such as sound, images, and video each have many possible representations. When we refer to an “MP3” audio file, or a “JPEG” image, or an “MPEG” movie, we refer to a file that follows a specific set of rules about what bit patterns correspond to the sound, image, or video being represented.
- Computer instructions: both the data and instructions are stored as binary values in the computer’s memory. We will look at this in more detail next.