Lecture 4A: Numbering Systems

- Introduction
- Base-2 (Binary)
  - Converting From Decimal to Binary
  - Converting from Binary to Decimal
- Base-8 (Octal)
- Base-16 (Hexadecimal)
- Summary
  - Bonus (DLC?) Content: Why Use Binary At All?
    - Calcula the Numb of Bits for a Given Decim Numb

Introduction

We use a base-10 numbering system. What does this mean? Let's take an example of counting to 20. In a *tally* system, you might make a mark for every number. You'd have 20 marks by the end of it. This is very inefficient. Our normal way of representing numbers is to count to 9, then represent 9+1 by resetting our count to zero and *carrying over* the 1.

10^2	10^1	10^0
0	1	0

Hopefully makes sense, we've been doing it like this since Kindergarten! This is more efficient, since instead of requiring one *character* for every thing we count, we can compress our representation of that number count to two *digits*.

Base-2 (Binary)

Hopefully everyone is familiar with the idea that computers use a *binary* number system. What does that mean? It means that every one of our *digits* can have only two possible values that we understand: 1 or 0. More accurately, we chould say that every digit has two voltage levels: a high voltage and a low voltage. Let's use the number 5 as an example.

2^3	2^2	2^1	2^0
0	1	0	1

$2^2 = 4$ $2^0 = 1$

$4 + 1 = 5$

Simple enough, so far!

This, by the way, is four bis, which we can call a *nibble*.

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

And this is 8 bits, which we call a *byte*. If you add 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1, you get 255, which is the maximum *unsigned* value you can represent with a byte.

Converting From Decimal to Binary

One skill you will required to demonstrate in this course is the ability to do this conversion *without a calculator*.

There are several strategies to do this, but here's one that I recommend:

Let's start with an example: **78**.

Use long division to divide *78* by *2*.

78/2 = 39 with a remainder of 0.

The remainder becomes your *least significant bit*.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
–	–	–	–	–	–	–	**0**

Next, divide your result by 2 again:

39/2 = 19, *remainder* = 1.

Again, your remainder gets added to the *next* least significant bit.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
–	–	–	–	–	–	**1**	0

19/2 = 9, *remainder* = 1.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
–	–	–	–	–	**1**	1	0

9/2 = 4, *remainder* = 1.

4/2 = 2, *remainder* = 0.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
–	–	–	**0**	**1**	1	1	0

The last division of 2/2 = 1 with a remainder of 0. **Remember to add the remainder to your binary number first**. Since our final result is less than our *radix*, we can add this to the *most significant bit* of our binary number.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
–	**1**	**0**	0	1	1	1	0

One word about style: It is good practice to divide your binary numbers into nibbles to make it easier to read. We will add a leading zero and write our answer like so:

```
0100 1110
```

Converting from Binary to Decimal

This should be easy if you can memorize the following:

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

For this course, we aren't going to ask you to convert exceedingly long numbers, so this should work for any quiz/test questions. `0b` is one common way to *represent* binary numbers, which we will use here.

Example: `0b10101100` .

First, arrange this number in a way that's easy to read:

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	0	1	0	1	1	0	0

(1 * 128) + (1 * 32) + (1 * 8) + (1 * 4) = 172.

Another Handy Resource

Base-8 (Octal)

Octal numbers have digits between 0 and 7, at which point they carry over. We use octal numbers with `chmod` , which we will discuss this week.

8^2	8^1	8^0
16	8	1

When being asked to do conversions with octal, the easiest approach is to start with binary and to group bits into *groups of three*.

`0b10101100` becomes **10 101 100**. Convert from binary to 'decimal' to get: **2 5 4**

The way to correctly identify an octal number varies, but a leading zero seems to be most common. So our number is `0254` .

To convert from octal to another format, convert each digit to binary, then group your binary digits into nibbles.

Base-16 (Hexadecimal)

Hexadecimal is used often to represent memory addresses. Hexadecimal numbers have digits between 0 and 15, at which point they carry over. Here's how we represent these digits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

When being asked to do conversions with octal, the easiest approach is to start with binary and to group bits into *groups of four*.

Let's convert from Hex into decimal. Here's our Hex Number: `0xD6` .

`0x` is the most common way of representing Hex numbers.

D = 13, which is 1101 in binary. 6 = 6, which is 0110 in binary.

`0xD6` = 1101 0110. From here we can convert to decimal to get 214.

Summary

- Base-10 is the number system you are already familiar with.
- Base-2 is binary. Two possible values: **1** or **0**.
- To convert decimal to binary, divide the decimal number by 2, and assign the remainder to the least signficant bit. (LSB). Continue until you have a result less than 2, and add this to your binary number.
- Base-8 is octal. Possible values: **0 - 7**. Octal digits contain **3** bits.
- Base-16 is hexadecimal. Possible values: **0 - F**. Hex digits contain **4** bits.

Bonus (DLC?) Content: Why Use Binary At All?

- Why Use Binary? - My Attempt
- Why Use Binary? - Computerphile

Calculating the Number of Bits for a Given Decimal Number

This is *not* a calculation that you will need to demonstrate on any tests in this course, but it *is* something that you will find useful in many practical situations.

Let's say you are given a decimal number 65001. You want to decided what *unsigned* data type is required to store this number. Use this formula:

$log_2 65001 = 15.988$ Take the *ceiling* of that number (round up to the next integer) to get the number of bits required for this number. (In this case, 16 bits are required.)

Using a calculator, it can be difficult to program a log_2 calculation, so in a pinch, you can use log_{10}:

$log_{10} 65001 / log_{10} 2 = 15.988$