CIS 351 - Computer Organization & Assembly Language

Nathan Bowman

Images taken from Harris & Harris book

Using Binary Numbers

A recurring theme in this course is going to be the question of how much information can be stored with a certain number of bits

We will not always think of binary numbers as "numbers"

A single bit allows us to differentiate between two "things", such as:

- left/right
- high/low
- odd/even

For example, in some application we might store a value where `0` indicates "left" and `1` indicates "right"

We only need one bit to keep that information

What if we want to distinguish between north (N), south (S), east (E), and west (W)?

Let's assign each one a bit value:

- N: 00
- E: 01
- S: 10
- W: 11

So, to distinguish between four things, two bits is enough

In general, each bit gives us an additional factor of 2

- 1 bit: 2 things
- 2 bits: 4 things (= 2 * 2)
- 3 bits: 8 things (= 2 * 2 * 2)
- 4 bits: 16 things (= 2 * 2 * 2 * 2)

With n bits, we can distinguish between 2^n different things

That is exponential growth, which means that what we can store goes up very, very quickly as a function of n

In the abstract, how many "things" you can represent in one byte (8 bits)?

2^8 = 256

If we think of our byte as a number, the easiest way is as the values from 0 - 255

They could also be the values from -128 - 127 -- we would just need to change how we interpret them

Or, they could represent 256 different characters (a-z, A-Z, punctuation, etc.)

You don't need to memorize them, but some handy rules of thumb:

- 2^10 is approximately 1,000 (1024)
- 2^20 is approximately 1,000,000 (1,048,576)
- 2^30 is approximately 1,000,000,000 (1,073,741,824)

The inverse of an exponential is a logarithm

Because we are working with exponentials of the form 2^X, we should work with the base-2 logarithm (log_2) to get the inverse

Logarithms turn enormous numbers into manageable ones, which is why computer scientists love them

For example, we just saw that 2^30 is approximately one billion. That means log_2(one billion) is approximately 30

Because they are the inverse, logarithms help us answer the opposite question of the one we have been asking:

Given X things to represent, how many bits do I need?

The answer: log_2(X) (rounding up)

If I want to represent 6 different sports (e.g., hockey, soccer, tennis, baseball, football, golf), how many bits would it take to do so?

log_2(6) = 2.58... , which rounds up to 3

Or, if you like: 2^2 < 6 < 2^3, so it takes 3 bits to store the 6 sports

Since there is no particular ordering among the sports, any encoding will do. For example:

- 000: hockey
- 001: soccer
- 010: tennis
- 011: baseball
- 100: football
- 101: golf
- 110: [not used]
- 111: [not used]

It is important to keep in mind those two questions throughout the course:

- how many bits would I need to store X things?
- how many things could I store with Y bits?

These same ideas hold true for other bases, but we will not often use them

In base Z:

- we need log_Z(X) digits to store X things
- we can represent Z^Y things using Y digits

Try to think about the case where Z = 10 to get some intuition for this

Essentially, you just need to be able to count to the number of things you want to store