

WELCOME TO COMP 311!



1. Course Mechanics

- What do I have to do to get an "A" in this course?
- Where are the course materials posted online, because I'm pretty sure that I am gonna sleep through a lot of these lectures.

2. Course Objectives

- How do computers work?
- Show me the binary?
- Some assembly required.

3. Course Changes

WHOS



Lectures:



Leonard McMillan (SN 316)
Office Hours: W 2-4pm

TA:



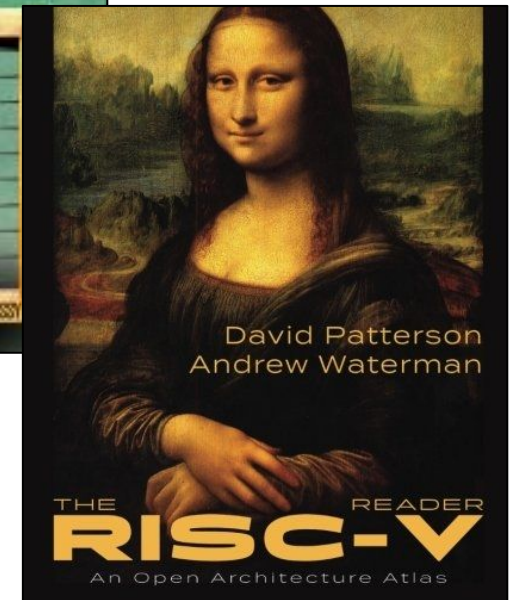
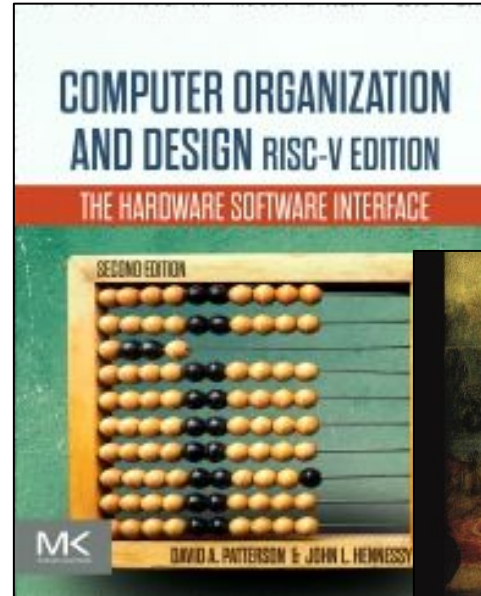
Vikram P.T.
Office Hours: TBD



WHATS

Books: None Required,
Supplemental Texts

- Will he follow any of these books?
 - Definitely not
- Are the problem set answers in the book?
 - Perhaps
- Why do I need them then?
 - In case you find yourself lost, need additional examples, or need a doorstop



COURSE MECHANICS



Grading:

Best 5 of 6 problem sets

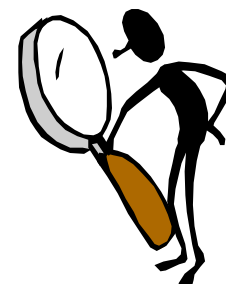
30%

2 in-class exams

40%

Final exam

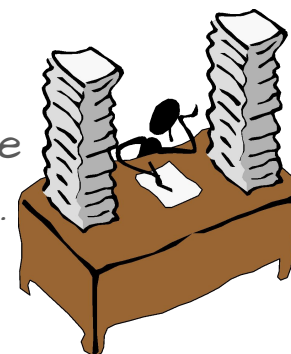
30%



You will have at least two weeks to complete each problem set.


Problem sets will be online. *Late problem sets will not be accepted*, but the lowest problem-set score will be dropped.

I will attempt to make Lecture Notes, Problem Sets, and other course materials available on the web before class on the day they are given.



COURSE WEBSITE





```
1111 1111 0000 0001 0000 0001 0001 0011 0xFF010113 main: addi sp,sp,-16
0000 0000 0001 0001 0010 0110 0010 0011 0x06112623 sw ra,12(sp) # include <stdio.h>
0000 0000 0000 0000 0000 0101 0001 0111 0x00000517 la a0,_S0002 # main() {
0000 0000 0000 0001 0000 0000 0000 0000 0x01C00513 fadd v12,v10,v0
0100 0000 0000 0000 0000 0000 0000 0000 0x00000000 li a0,0 # "Hello world!\n");
0000 0000 0000 0001 0000 0101 0001 0011 0x00000000 ri a0,0 # return 0;
0000 0000 1100 0001 0010 0000 1000 0011 0x00000203 lw ra,12(sp) # }
0000 0001 0000 0001 0000 0001 0001 0011 0x01000000 addi sp,sp,16
0000 0000 0000 0000 1000 0000 0110 0111 0x00000807 ret
0110 1100 0110 1100 0110 0101 0100 1000 0x6C6C6548 _S0002: .string "Hello World\n"
```

[Home](#) [Research](#) [Courses](#) [Publications](#) [Setup](#)

Announcements

- **August 16, 2022:** The first class meeting (summer is over).

Course Description

Comp 311, *Computer Organization*, explores the topic of how computers work, in terms of both software and hardware. It covers a wide range of topics including what a *bit* is, and why bits are the atoms in the universe of computation. We also discuss how information is represented and processed in hardware, and arrive to the conclusion that, to a computer, everything is data, including the instructions that underly software.

Comp 311 also covers the wide range of languages, and layers of translation, used for computation-- spanning from machine language to assembly language to high-level compiled and interpreted languages. We will also touch on the conventions that will enable us to construct large programs, modular software systems, and even programs that manage the loading, execution, and creation of other programs.

We will then delve deeper into computer hardware to discover what means to be *digital*. We will explore how simple combinational logic can be made to perform math and manipulate bits and how logic with state can be made to perform a series of operations. This will culminate in the virtual construction of a simple, yet fully functional computer.

In the last third of the class we will discuss issues of performance. What the measures of MIPS and CPI mean, and how they can be improved. We will discuss simple techniques for increasing the rates which computer execute instructions including pipelining and parallelism. We will then address techniques for improving the apparent memory bandwidth of a computer and finally how to simulate more memory that we can actually afford.

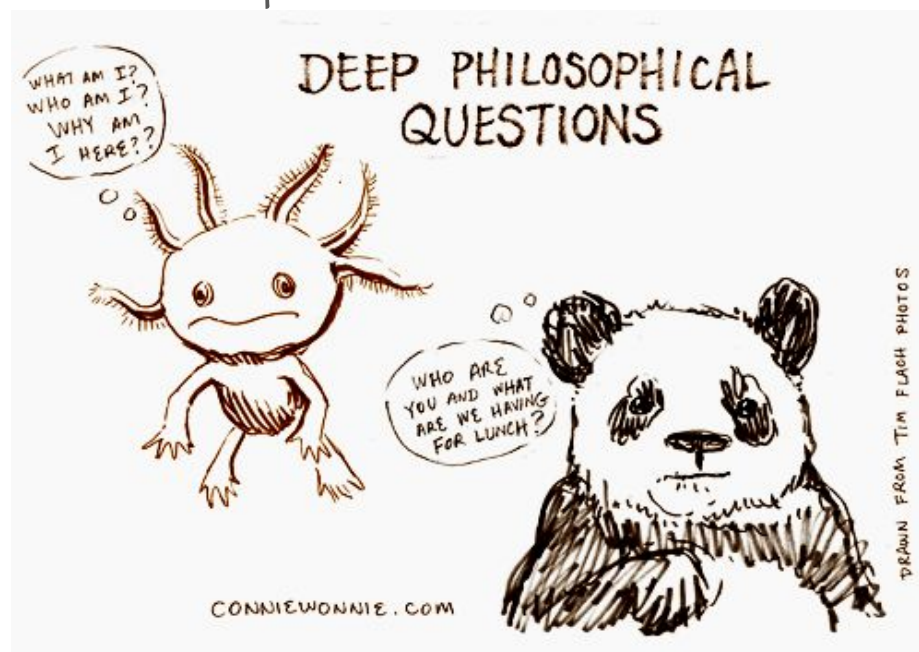
<http://csbio.unc.edu/mcmillan/index.py?run=Courses.Comp311F22>

GOALS OF COMP 311



To answer fundamental questions:

- What does a computer do with my program?
- How is data represented in a computer?
 - Numbers
 - Strings
 - Arrays
 - Photographs
 - Music
- How is a *program* represented in a computer?
- Are there any limits to what a computer can do?



GOAL 1: TO DEMYSTIFY COMPUTERS



Strangely, most people seem to be afraid of computers.

People only fear things they do not understand!



"I do not fear computers, I fear the lack of them."

- Isaac Asimov (1920 - 1992)

"Fear is the main source of superstition, and one of the main sources of cruelty. To conquer fear is the beginning of wisdom."

- Bertrand Russell (1872 - 1970)

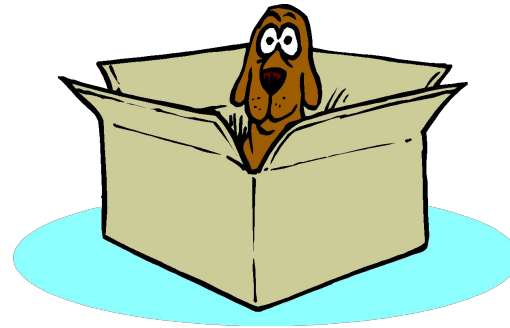
"Nobody knows exactly what's going on because of computers!"

- Donald Trump

GOAL 2: THE POWER OF ABSTRACTION



Define a function, develop a roust implementation, and then put a box around it.



Abstraction enables us to create unfathomable systems, including computer hardware and software.

Why do we need ABSTRACTION? Imagine a billion...

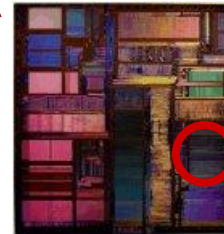
Orchestrating systems with >1G components



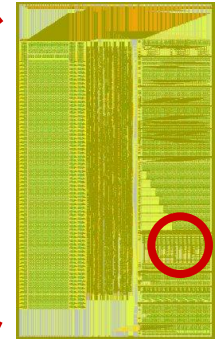
A modern computer:
Hardware & Software



Circuit Boards:
≈8 / system
1-2G devices

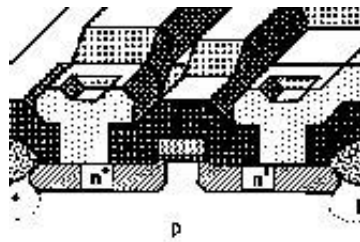


Integrated Circuit:
≈8-16 / PCB
1M-250M devices



Module:
≈8-16 / IC
100K devices

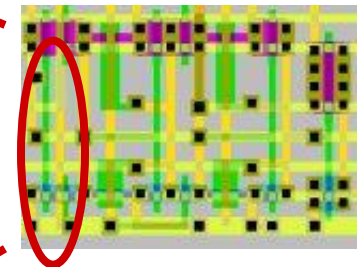
MOSFET



Scheme for
representing
information



Gate: ≈2-16 / Cell
8 devices

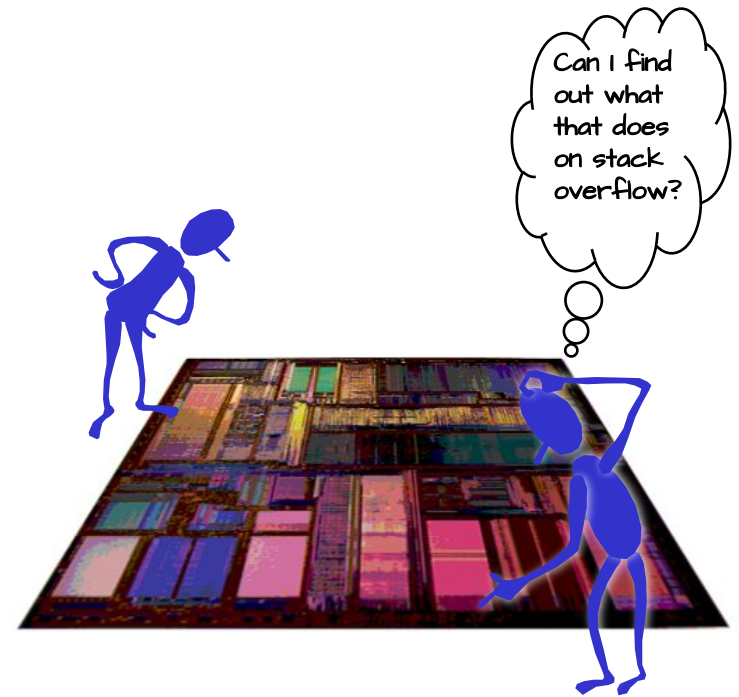


Cell:
≈1K-10K / Module
16-64 devices

WHAT'S IN A COMPUTER?



- Structure
 - Hierarchical design
 - Limited complexity at each level
 - Reusable building blocks
- Interfaces
 - Key element of system engineering typically outlives its implementation
 - Isolate design from technology, allows evolution
 - Major abstraction mechanism
- What makes a good system?
 - "Bang for the buck." Minimal mechanism, maximal function
 - Reliable, resilient, reusable
 - Accommodating future improvements

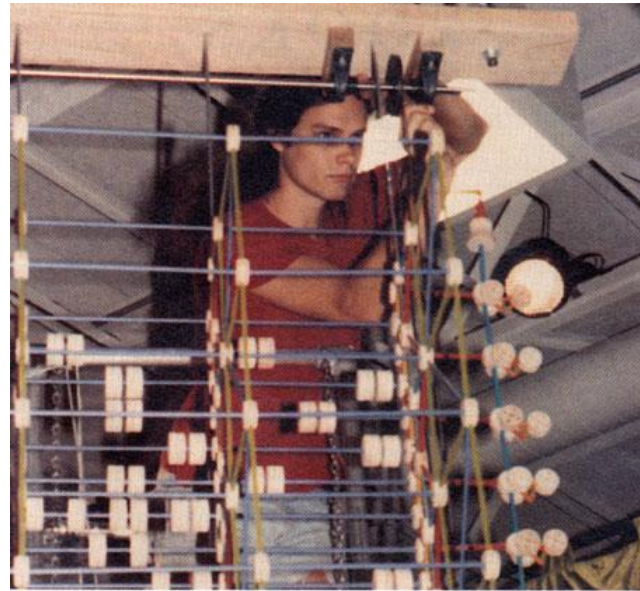
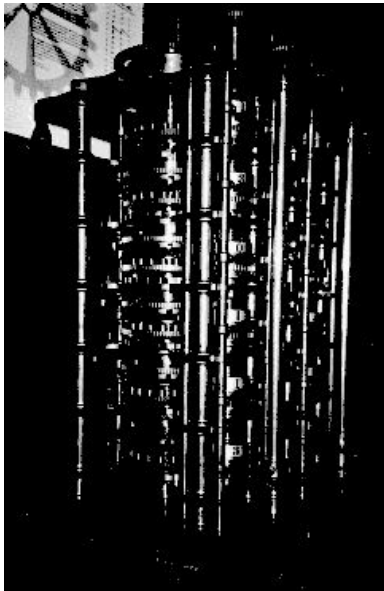




COMPUTATIONAL STRUCTURES

What are the fundamental elements of computation?

Can we define computation independent of implementation or the technology that it is implemented with?



Edward Hardebeck helps to assemble the Tinkertoy computer



WHAT DO PROGRAMS REALLY DO?

By now you should be able to look at a program specification and figure out what it does.

What does this do?

How would you figure it out?

Try $f(36)$, $f(64)$, $f(100)$

```
int f(int x) {  
    int r;  
    int odd = 1;  
    for (r = 0; x >= odd; r++) {  
        x -= odd;  
        odd += 2;  
    }  
    return r;  
}
```



HOW DOES A COMPUTER DO IT?

What does a computer do with this program specification?

```
int f(int x) {  
    int r;  
    int odd = 1;  
    for (r = 0; x >= odd; r++) {  
        x -= odd;  
        odd += 2;  
    }  
    return r;  
}
```

```
f:      mv      t0, a0  
        li      t1, 1  
        li      a0, 0  
        blt     t0, t1, return  
loop:   sub     t0, t0, t1  
        addi    t1, t1, 2  
        addi    a0, a0, 1  
        bge     t0, t1, loop  
return: ret
```

It translates it to a series of simple instructions...

ARE THERE LIMITS TO COMPUTATION?



- Will some new instruction be invented that fundamentally change how fast computers solve problems?
- Can computers solve *any* well specified problem?
- Can we predict how long it will take for a computer to solve a given problem?
- Does there exist a new model of computation?



A PROGRAM EMULATING A COMPUTER



```
int memory[16384];    // for instructions and data
int register[32];    // for variables
int pc;              // next instruction to execute
int flags;          // persistent state

void main(void) {
    pc = 0;
    while (1) {
        instruction = memory[pc];
        pc = pc + 1;
        flags = execute(instruction);
    }
}
```

A computer is just an interpreter that executes simple program loop



WHERE ARE WE GOING?

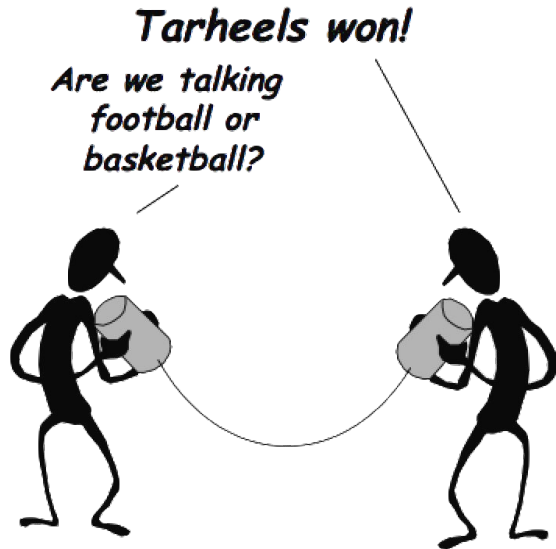
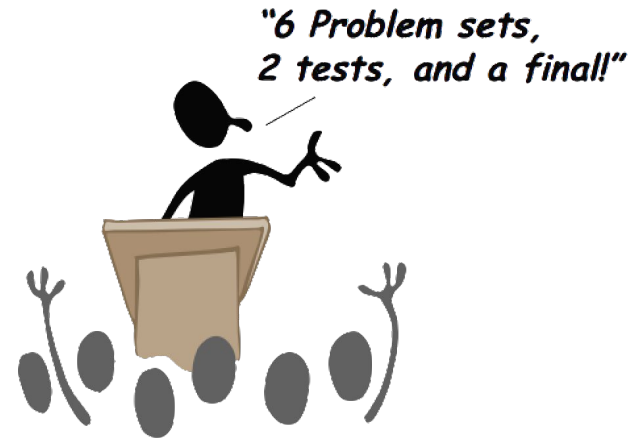
- How is data represented, stored, and manipulated in a computer?
- What basic operations does a computer use?
- What does mean to "compute"?
- Are there limits to what can be computed?
- Why are computers so fast?
- What am I asking a computer to do when I give it a program to execute?
- How are programs translated into computer instructions?
- Why are some programs faster than others that perform the same function?





WHAT IS "INFORMATION"?

information, n. Knowledge communicated or received concerning a particular fact or circumstance.



A Computer Scientist's definition:

Information resolves uncertainty.

Information is simply that which cannot be predicted. The less likely a message is, the more information it conveys.

QUANTIFYING INFORMATION

(Claude Shannon, 1948)



Suppose you're faced with N equally probable choices, and I give you a fact that narrows it down to M possibilities. Then you've been given:

$\log_2(N/M)$ bits of information

Information is measured in bits (binary digits) = number of 0/1's required to encode choice(s)

Examples:

- Outcome of a coin flip: $\log_2(2/1) = 1$ bit
- The roll of one die? $\log_2(6/1) = \sim 2.6$ bits
- Someone tells you that their 7-digit phone number is a palindrome?

$$\log_2(10^7/10^4) = \sim 9.966 \text{ bits}$$

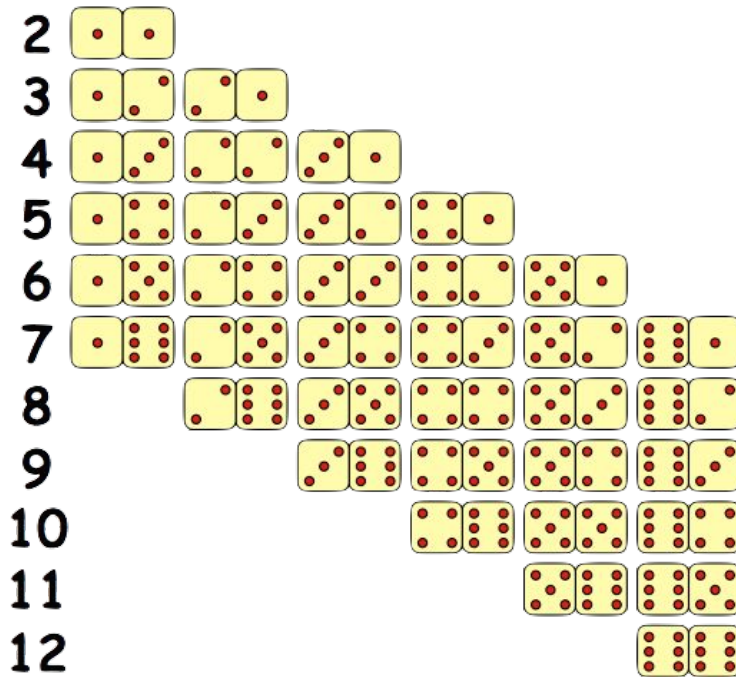


Information is the theoretical underpinning of why digital computers use bits, there is also an engineering rationale that we'll discuss later on.





ANOTHER EXAMPLE: SUM OF 2 DICE



$$\begin{aligned}i_2 &= \log_2(36/1) = 5.170 \text{ bits} \\i_3 &= \log_2(36/2) = 4.170 \text{ bits} \\i_4 &= \log_2(36/3) = 3.585 \text{ bits} \\i_5 &= \log_2(36/4) = 3.170 \text{ bits} \\i_6 &= \log_2(36/5) = 2.848 \text{ bits} \\i_7 &= \log_2(36/6) = 2.585 \text{ bits} \\i_8 &= \log_2(36/5) = 2.848 \text{ bits} \\i_9 &= \log_2(36/4) = 3.170 \text{ bits} \\i_{10} &= \log_2(36/3) = 3.585 \text{ bits} \\i_{11} &= \log_2(36/2) = 4.170 \text{ bits} \\i_{12} &= \log_2(36/1) = 5.170 \text{ bits}\end{aligned}$$

The average information provided by the sum of 2 dice is: 3.274

$$i_{\text{ave}} = \sum_{i=2}^{12} \frac{M_i}{N} \log_2\left(\frac{N}{M_i}\right) = -\sum_i p_i \log_2(p_i)$$

The average information of a process is called its **Entropy**.



SHOW ME THE BITS!

- Is there a concrete ENCODING that achieves its information content?
- Can the sum of two dice REALLY be represented using 3.274 bits?
- If so, how?
- The fact is, the average information content is a strict lower-bound on how small of a **representation** that we can achieve.
- In practice, it is difficult to reach this bound. But, we can come very close.





VARIABLE-LENGTH ENCODING

- Of course we can use differing numbers of "bits" to represent each item of data
- This is particularly useful if all items are not equally likely
- Equally likely items lead to fixed length encodings:
 - Ex. Encode a "particular" roll of 5?
 - $\{(1,4),(2,3),(3,2),(4,1)\}$ which are equally likely if we use fair dice

$$\text{Entropy} = \sum_{i=1}^4 p(\text{roll}_i | \text{roll} = 5) \log_2(p(\text{roll}_i | \text{roll} = 5)) = \sum_{i=1}^4 \frac{1}{4} \log_2\left(\frac{1}{4}\right) = 2$$

- $00 = (1,4), 01 = (2,3), 10 = (3,2), 11 = (4,1)$
- Back to the original problem. Let's use this encoding:

2 = 10011	3 = 0101	4 = 011	5 = 001
6 = 111	7 = 101	8 = 110	9 = 000
10 = 1000	11 = 0100	12 = 10010	



VARIABLE-LENGTH DECODING

2 = 10011

3 = 0101

4 = 011

5 = 001

6 = 111

7 = 101

8 = 110

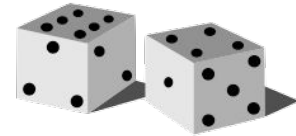
9 = 000

10 = 1000

11 = 0100

12 = 10010

- Notice how unlikely rolls are encoded using more bits, whereas likely rolls use fewer bits
- Let's use our encoding to decode the following bit sequence



2	5	3	6	5	8	3
10011	1001	0101	111	1001	110	0101

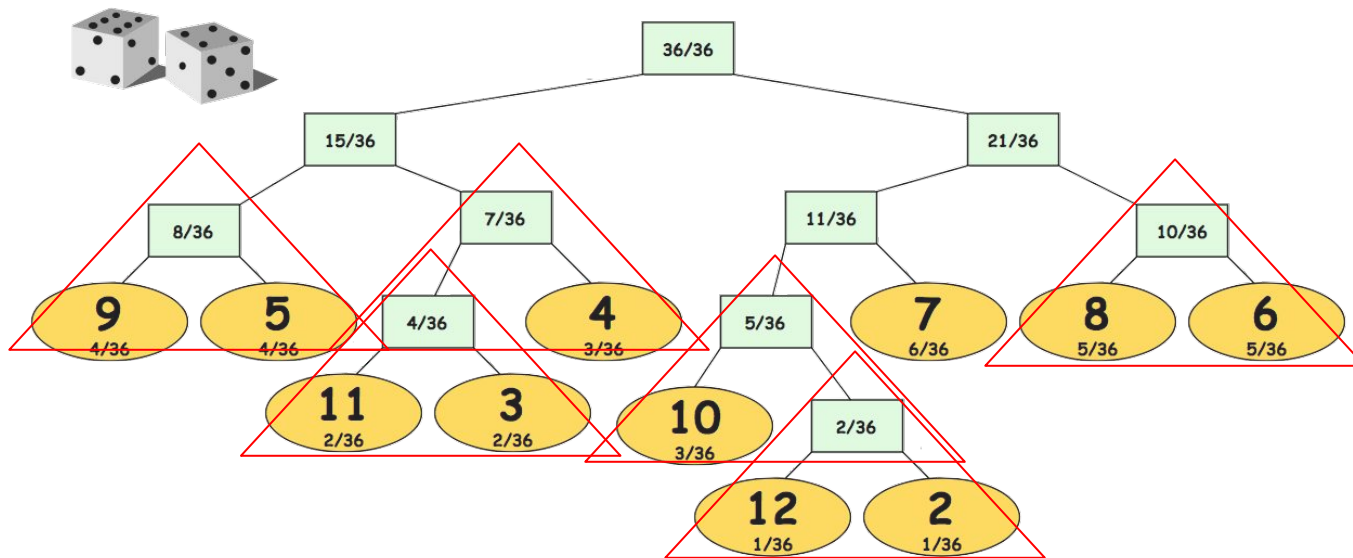
- Where did this code come from?



HUFFMAN CODING

A simple greedy algorithm for approximating a minimum encoding

1. Find the 2 items with the smallest probabilities
2. Join them into a new *meta* item with probability of their sum
3. Remove the two items and insert the new meta item
4. Repeat from step 1 until there is only one item

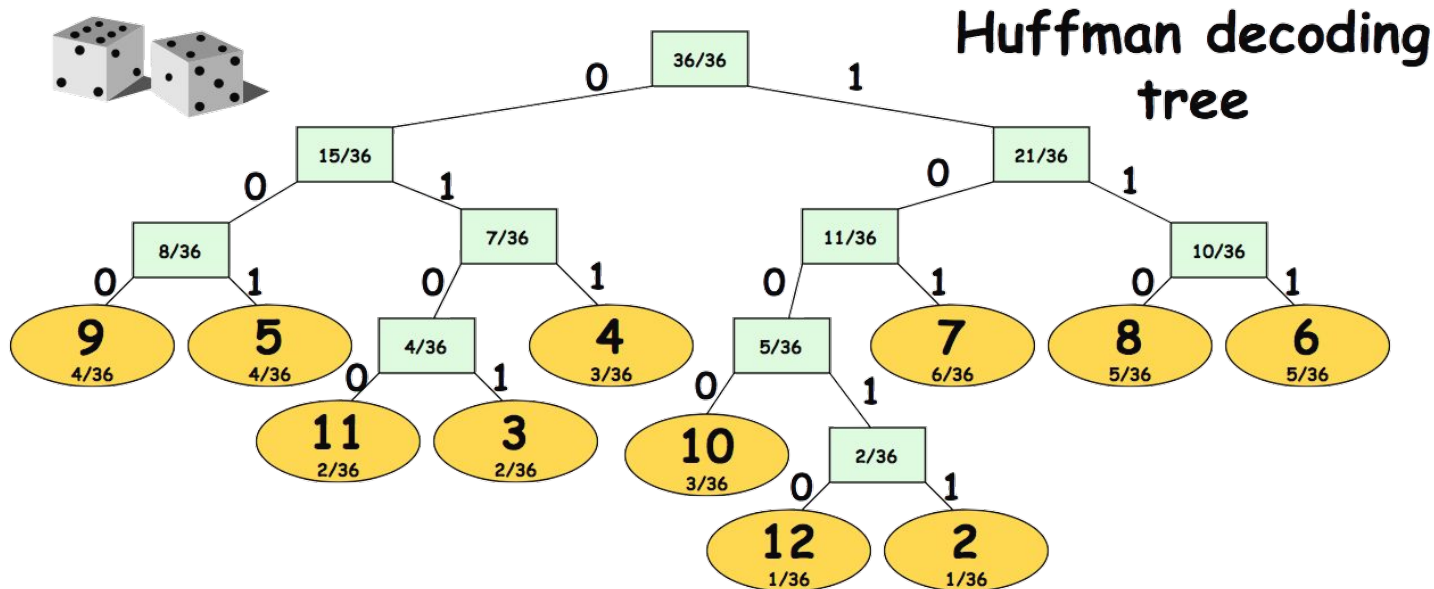


CONVERTING A TREE TO AN ENCODING



Once the *tree* is constructed, **label its edges consistently** and follow the paths from the largest *meta* item to each of the real items to find the encoding.

2 = 10011 3 = 0101 4 = 011 5 = 001 6 = 111 7 = 101
8 = 110 9 = 000 10 = 1000 11 = 0100 12 = 10010





CODING EFFICIENCY

How does this code compare to the information content?

$$b_{ave} = \frac{1}{36}5 + \frac{2}{36}4 + \frac{3}{36}3 + \frac{4}{36}3 + \frac{5}{36}3 + \frac{6}{36}3 + \frac{5}{36}3 + \frac{4}{36}3 + \frac{3}{36}4 + \frac{2}{36}4 + \frac{1}{36}$$

$$b_{ave} = 3.306$$

Pretty close. Recall that the lower bound was 3.274 bits.

However, an efficient encoding (as defined by having an average code size close to the information content) is not always what we want!

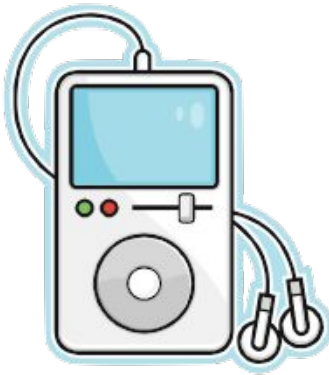
Sometimes a uniform code is easier to deal with.

Sometimes redundancy is a good thing.



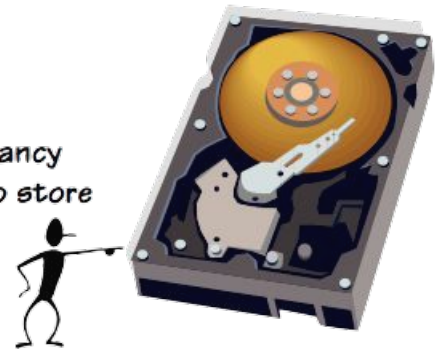
ENCODING CONSIDERATIONS

- Encoding schemes that attempt to match the information content of a data stream **remove redundancy**. They are **data compression** techniques.
- Make the information easier to manipulate (fixed-sized encodings)
- However, sometimes our goal in encoding information is **increase redundancy**, rather than remove it. Why?
- Adding redundancy can make data resilient to noise (**error detecting and correcting codes**)



-Data compression allows us to store our entire music and video collections in a pocketable device

-Data redundancy enables us to store that *same* information *reliably* on a hard drive





SUMMARY

- 311 answers the following questions:
 - How is information represented, stored, and manipulated by a computer?
 - What does a computer really do with my program?
 - How do you design, build, and manage large systems?
- Information is all about bits, and Computers process it
 - Entropy
 - Using bits to encode things
 - Efficient variable-length encodings
 - Redundancy