Welcome to Comp 411!

I thought this course was called “Computer Organization”

1) Course Mechanics
2) Course Objectives
3) Information
Meet the Crew...

Lectures: Leonard McMillan (SN-258)
Office Hours T 2-3

TA: TBA (We’ll find out next Monday)

Book: Patterson & Hennessy
Computer Organization & Design
(However, you won’t need it for the next couple of weeks)
Course Mechanics

Grading:

Best 8 of 10 problem sets 40%
2 Quizzes 30%
Final Exam 30%

Problem sets will be distributed on Wednesdays and are due back on the next Wednesday *lecture* (before it begins). Usually you will have one week to complete each set. Late problem sets will not be accepted, but the lowest two problem-set scores 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.
Comp 411: Course Website

Welcome to Comp 411! Check back here regularly for course updates and news. For now you can download the course syllabus. We will hold our first class meeting on 8/22 in SN014 from 11:00 am to 12:15 pm.

Textbook:

Computer Organization & Design, 3rd Edition, by Patterson & Hennessy. We will begin using this text on our 4th class meeting (9/5/06).

http://www.unc.edu/courses/2007fall/comp/411/001
Goal 1: Demystify Computers

Strangely, most people (even some computer scientists) are afraid of computers.

We are only afraid of things we 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)
Goal 2: Power of Abstraction

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

Abstraction enables us to create unfathomable systems (including computers).

Why do we need ABSTRACTION…

Imagine a billion --- 1,000,000,000
The key to building systems with >1G components

Personal Computer: Hardware & Software

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

Integrated Circuit:
≈8-16 / PCB
.25M-16M 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 do we See in a Computer?

• **Structure**
  - hierarchical design:
    - limited complexity at each level
    - reusable building blocks

• **Interfaces**
  - Key elements of system engineering; typically outlive the technologies they interface
  - Isolate technologies, allow evolution
  - Major abstraction mechanism

• **What makes a good system design?**
  - “Bang for the buck”:
    - minimal mechanism, maximal function
  - reliable in a wide range of environments
  - accommodates future technical improvements
Computational Structures

What are the fundamental elements of computation? Can we define computation independent of implementation or the substrate that it built upon?)
Our Plan of Attack...

- Understand how things work, by alternating between low-level *(bottom-up)* and high level *(top-down)* concepts
- Encapsulate our understanding using appropriate abstractions
- Study organizational principles: hierarchy, interfaces, APIs.

- Roll up our sleeves and design at each level of hierarchy
- Learn engineering tricks
  - from a historical perspective
  - using systematic design approaches
  - diagnose, fix, and avoid bugs
What is “Computation”?

Computation is about “processing information”

- Transforming information from one form to another
- Deriving new information from old
- Finding information associated with a given input
- “Computation” describes the motion of information through time
- “Communication” describes the motion of information through space
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 predictable a message is, the more information it conveys!

“10 Problem sets, 2 quizzes, and a final!”

Tarheels won!

Are you sure? It is football season...
Real-World Information

Why do unexpected messages get allocated the biggest headlines?

... because they carry the most information.
What Does A Computer Process?

• Toasters processes bread and bagels
• Blenders processes smoothies and margaritas
• What does a computer process?
• 2 allowable answers:
  – Information
  – Bits
• How does information relate to bits?
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 choices. Then you’ve been given:

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

Examples:

- information in one coin flip: $$\log_2(2/1) = 1$$ bit
- roll of a single die: $$\log_2(6/1) = \sim 2.6$$ bits
- outcome of a Football game: 1 bit
  (well, actually, “they won” may convey more information than “they lost”…)

Information is measured in bits (binary digits) = number of 0/1’s required to encode choice(s)
Example: Sum of 2 dice

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

The average information provided by the sum of 2 dice:

\[ i_{\text{ave}} = \sum_{i=2}^{12} \frac{M_i}{N} \log_2 \left( \frac{N}{M_i} \right) = -\sum_i p_i \log_2(p_i) = 3.274 \text{ bits} \]
Show Me the Bits!

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
  - Entropy = \(-\sum_{i=1}^{4} p(roll_i|roll = 5) \log_2 (p(roll_i|roll = 5)) = -\sum_{i=1}^{4} \frac{1}{4} \log_2 (\frac{1}{4}) = 2\) bits
  - \(00\) – \(1,4\), \(01\) – \(2,3\), \(10\) – \(3,2\), \(11\) – \(4,1\)
- 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 Encoding

• Taking a closer look

<table>
<thead>
<tr>
<th>Number</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10011</td>
</tr>
<tr>
<td>3</td>
<td>0101</td>
</tr>
<tr>
<td>4</td>
<td>011</td>
</tr>
<tr>
<td>5</td>
<td>001</td>
</tr>
<tr>
<td>6</td>
<td>111</td>
</tr>
<tr>
<td>7</td>
<td>101</td>
</tr>
<tr>
<td>8</td>
<td>110</td>
</tr>
<tr>
<td>9</td>
<td>000</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>0100</td>
</tr>
<tr>
<td>12</td>
<td>10010</td>
</tr>
</tbody>
</table>

Unlikely rolls are encoded using more bits

More likely rolls use fewer bits

• Decoding

Example Stream: 1001100101011110011100101

2 5 3 6 5 8 3
Huffman Coding

- A simple *greedy* algorithm for approximating an entropy efficient encoding
  1. Find the 2 items with the smallest probabilities
  2. Join them into a new *meta* item whose probability is the sum
  3. Remove the two items and insert the new meta item
  4. Repeat from step 1 until there is only one item
Converting Tree to Encoding

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

Huffman decoding tree
Encoding Efficiency

How does this encoding strategy compare to the information content of the roll?

\[ b_{\text{ave}} = \frac{1}{36} \cdot 5 + \frac{2}{36} \cdot 4 + \frac{3}{36} \cdot 3 + \frac{4}{36} \cdot 3 + \frac{5}{36} \cdot 3 + \frac{6}{36} \cdot 3 \]

\[ \quad + \frac{5}{36} \cdot 3 + \frac{4}{36} \cdot 3 + \frac{3}{36} \cdot 4 + \frac{2}{36} \cdot 4 + \frac{1}{36} \cdot 5 \]

\[ b_{\text{ave}} = 3.306 \]

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!
Encoding Considerations

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

- Data compression allows us to store our entire music collection in a pocketable device
- Data redundancy enables us to store that *same* information *reliably* on a hard drive
Error detection using parity

Sometimes we add extra redundancy so that we can detect errors. For instance, this encoding detects any single-bit error:

<table>
<thead>
<tr>
<th>Number</th>
<th>Bitstream</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1111000</td>
</tr>
<tr>
<td>3</td>
<td>1111101</td>
</tr>
<tr>
<td>4</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0000</td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>1010</td>
</tr>
<tr>
<td>10</td>
<td>1100</td>
</tr>
<tr>
<td>11</td>
<td>1111110</td>
</tr>
<tr>
<td>12</td>
<td>1111011</td>
</tr>
</tbody>
</table>

There's something peculiar about those codings.

Same bitstream – w/4 possible interpretations if we allow for only one error.
Property 1: Parity

The sum of the bits in each symbol is even.

This is how errors are detected.

How much information is in the last bit?

The sum of the bits in each symbol is even.

Property 1: Parity
## Property 2: Separation

Each encoding differs from all others by at least **two bits** in their overlapping parts.

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1111000</td>
<td>1111x0x</td>
<td>xx11</td>
<td>x1x1</td>
<td>x11x</td>
<td>xxxx</td>
<td>1xx1</td>
<td>1x1x</td>
<td>11xx</td>
<td>1111xx0</td>
</tr>
<tr>
<td>3</td>
<td>1111101</td>
<td>xx11</td>
<td>x1x1</td>
<td>x11x</td>
<td>xxxx</td>
<td>1xx1</td>
<td>1x1x</td>
<td>11xx</td>
<td>11111xx</td>
<td>1111xx1</td>
</tr>
<tr>
<td>4</td>
<td>0011</td>
<td>0xx1</td>
<td>0x1x</td>
<td>00xx</td>
<td>x0x1</td>
<td>x01x</td>
<td>xxxx</td>
<td>xx11</td>
<td>xx11</td>
<td>xx11</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>01xx</td>
<td>0x0x</td>
<td>xx01</td>
<td>xxxx</td>
<td>x10x</td>
<td>x1x1</td>
<td>x1x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0xx0</td>
<td>xxxx</td>
<td>xx10</td>
<td>x1x0</td>
<td>x11x</td>
<td>x11x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0000</td>
<td>x00x</td>
<td>x0x0</td>
<td>xx00</td>
<td>xxxx</td>
<td>xxxx</td>
<td>xxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
<td>10xx</td>
<td>1x0x</td>
<td>1xx1</td>
<td>1x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1010</td>
<td>1xx0</td>
<td>1x1x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11xx</td>
<td>11xx</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1111110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1111x1x</td>
</tr>
</tbody>
</table>

This difference is called the "Hamming distance".

"A Hamming distance of one-bit is needed to uniquely identify an encoding."
Error correcting codes

We can actually **correct** 1-bit errors in encodings separated by a Hamming distance of **three**. This is possible because the sets of bit patterns located a Hamming distance of 1 from our encodings are distinct.

However, **attempting error correction with such a small separation is dangerous**. Suppose, we have a 2-bit error. Our error correction scheme will then misinterpret the encoding. Misinterpretations also occurred when we had 2-bit errors in our 1-bit-error-detection (parity) schemes.

A **safe 1-bit error correction scheme would correct all 1-bit errors and detect all 2-bit errors**. What Hamming distance is needed between encodings to accomplish this?
An alternate error correcting code

We can generalize the notion of parity in order to construct error correcting codes. Instead of computing a single parity bit for an entire encoding we can allow multiple parity bits over different subsets of bits. Consider the following technique for encoding 25 bits.

1-bit errors will cause both a row and column parity error, uniquely identifying the errant bit for correction. An extra parity bit allows us to detect 1-bit errors in the parity bits.

Many 2-bit errors can also be corrected. However, 2-bit errors involving the same row or column are undetectable.

This approach is easy to implement, but it is not optimal in terms of the number of bits used.
Summary

Information resolves uncertainty

- Choices equally probable:
  - N choices down to M $\rightarrow$ $\log_2(N/M)$ bits of information
- Choices not equally probable:
  - choice $i$ with probability $p_i$ $\rightarrow$ $\log_2(1/p_i)$ bits of information
  - average number of bits = $\sum p_i \log_2(1/p_i)$
  - use variable-length encodings

Next time:
- Technology and Abstraction
  What makes a computer tick...