Welcome to CS 240: Introduction to Computer Systems
Course Website: https://courses.engr.illinois.edu/cs240/

Description: Basics of computer systems. Number representations, assembly/machine language, abstract models of processors (fetch/execute, memory hierarchy), processes/process control, simple memory management, file I/O and directories, network programming, usage of cloud services. 3 credit hours.

Instructors:
Prof. Wade Fagen-Ulmschneider <waf>, Course Instructor
Teaching Associate Professor of Computer Science, Grainger College of Engineering

Natalia Ozymko <nozymko2>, Lead Course Assistant
Computer Science, Grainger College of Engineering

Coursework and Grading
A total of 1,000 points are available in CS 240, along with many opportunities to earn extra credit. The points are broken down in the following way:

- **200 points**: Homeworks (10 × 20 points)
  - Points over 200 are extra credit!
  - Usually on PrairieLearn, but occasionally another platform
- **400 points**: Open-book Midterm Exams (2 × 200 points)
  - Midterm 1 Exam: Thursday, October 8, 2020
  - Midterm 2 Exam: Thursday, November 19, 2020
- **250 points**: Machine Projects (10 weeks × 25 points)
  - 7-8 MPs, including short (1-week) and long (2-week) MPs
  - Long MPs are worth 50 points, short MPs are worth 25 points
- **100 points**: Final Project
  - Multi-week Final Project, presented during finals weeks instead of a final exam (no final exam!)
- **50 points**: Participation
  - If you regularly engage with the course, you’ll receive the full points. I really want your feedback on how to build CS 240 to be the best course possible and you to enjoy it! :)

Final Course Grades
Your final course grade is determined by the number of points you earned during the semester:

<table>
<thead>
<tr>
<th>Points</th>
<th>Grade</th>
<th>Points</th>
<th>Grade</th>
<th>Points</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>[870, 900)</td>
<td>B+</td>
<td>[830, 870)</td>
<td>B</td>
<td>[800, 830)</td>
<td>B-</td>
</tr>
<tr>
<td>[770, 800)</td>
<td>C+</td>
<td>[730, 770)</td>
<td>C</td>
<td>[700, 730)</td>
<td>C-</td>
</tr>
<tr>
<td>[670, 700)</td>
<td>D+</td>
<td>[630, 670)</td>
<td>D</td>
<td>[600, 630)</td>
<td>D-</td>
</tr>
<tr>
<td>(600, 0]</td>
<td></td>
<td>(600, 0]</td>
<td></td>
<td>(600, 0]</td>
<td></td>
</tr>
</tbody>
</table>

We never curve individual exam or assignment scores; instead, if necessary, we may lower the points required for each grade cutoff to be lower than the stated cutoff. In no case will we raise the cutoff.

Foundations of Computer Systems
There are six major components to a computer, which we will refer to as the “foundations” of a computer system:

[1]:

#1: Introduction and C Programming
CS 240, Spring 2021 - Week 1
Wade Fagen-Ulmschneider
System-level Abstractions
After covering the “foundations”, we will begin to abstract the entire system as node and explore more complex topics:

Representing Data: Binary
All data within a computer is _____________; either 0 or 1.

Converting between base-2 and base-10:

\[
\begin{align*}
1_2 &: \ \text{___________}_{10} \\
10_2 &: \ \text{___________}_{10} \\
11_2 &: \ \text{___________}_{10} \\
100_2 &: \ \text{___________}_{10} \\
101 1000_2 &: \ \text{___________}_{10}
\end{align*}
\]

Any value can be represented in binary by writing it in base-2:

\[
\begin{align*}
4_{10} &: \ \text{___________}_{2} \\
7_{10} &: \ \text{___________}_{2} \\
18_{10} &: \ \text{___________}_{2}
\end{align*}
\]

In C/C++, you can write a number in binary by prefixing the number with 0b:

\[
\begin{align*}
11_{10} &: \ \text{___________} \\
33_{10} &: \ \text{___________}
\end{align*}
\]

Bit Manipulation:
We can manipulate bits by binary operations:

**AND, & operator:**

**OR, | operator:**

**XOR, ^ operator:**

**NOT, ! or ~ operator:**
Bit Manipulation:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp; B</th>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110011</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Representing Data: Hexadecimal**

Binary data gets really long, really fast! The number of students enrolled at University of Illinois is $\text{0b}1100011111111100$ (!!).

- To represent binary data in a compact way, we often will use **hexadecimal** -- or “base-16 -- denoted by the prefix $0x$.

**Hexadecimal Digits:**

11$_{10}$: $0x$__________

87$_{10}$: $0x$__________

34$_{10}$: $0x$__________

Hexadecimal is particularly useful as it ____________:

| University of Illinois student population in Fall 2019 (51,196): |
|-----------------|------------|------|------|------|------|
| $0b$            | $1100$     | $0111$| $1111$| $1100$|
| $0x$            |            |      |      |      |

| Number of people following Taylor Swift on Twitter (87,042,176): |
|-----------------|------------|------|------|------|------|
| $0b$            | $101$      | $0011$| $0000$| $0010$| $1000$| $1000$| $0000$|
| $0x$            |            |      |      |      |      |

**Orders of Magnitude**

Bits are organized into 8-bit chunks called ________.

Bytes are organized into by orders of magnitude.

1. **Historical Use of $10^x$:**

   4 KB on disk == ________ B

2. **Historical Use of $2^x$:**

   4 KB in RAM == ________ B

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Magnitude</th>
<th>Prefix</th>
<th>Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilo-, K-</td>
<td>$10^3$</td>
<td>kibi-, Ki-</td>
<td></td>
</tr>
<tr>
<td>mega-, M-</td>
<td></td>
<td>mebi, Mi-</td>
<td></td>
</tr>
<tr>
<td>giga-, G-</td>
<td></td>
<td>gibi, Gi-</td>
<td>$2^{30}$</td>
</tr>
<tr>
<td>tera-, T-</td>
<td>$10^{12}$</td>
<td>tebi-, Ti-</td>
<td>$2^{40}$</td>
</tr>
</tbody>
</table>

**Example:** Downloading a 1 GiB file on a “1 gig” connection:
Representing Letters: ASCII

Representing numbers is great -- but what about words? Can we make sentences with binary data?

- **Key Idea:** Every letter is _____ binary bits.
  (This means that every letter is _____ hex digits.)

- Global standard called the **American Standard Code for Information Interchange (ASCII)** is a ___________ ________________ for translating numbers to characters.

<table>
<thead>
<tr>
<th>ASCII Character Encoding Examples:</th>
<th>Binary</th>
<th>Hex</th>
<th>Char.</th>
<th>Binary</th>
<th>Hex</th>
<th>Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b0100 0001</td>
<td>0x41</td>
<td>A</td>
<td>0b0110 0001</td>
<td>0x61</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>0b0100 0010</td>
<td>0x42</td>
<td>B</td>
<td>0b0110 0010</td>
<td>0x62</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0b0010 0100</td>
<td>0x24</td>
<td>$</td>
<td>0b0111 1011</td>
<td>0x7b</td>
<td>{</td>
</tr>
</tbody>
</table>

...and now we can form sentences!

Q: Are there going to be any issues with ASCII?

---

**Representing Letters: Other Character Encodings**

Since ASCII uses only 8 bits, we are limited to only 256 unique characters. There’s far more than 256 characters -- and what about EMOJIs?? 🎉

- Many other character encodings exist other than ASCII.
- The most widely used character encoding is known as **Unicode Transformation Format (8-bit)** or ________.

UTF-8 uses a ___________ -bit design where each character by be any of the following:

Specifically the first four bits tell us about the number of bytes used to encode the character:

<table>
<thead>
<tr>
<th>Length</th>
<th>Byte #1</th>
<th>Byte #2</th>
<th>Byte #3</th>
<th>Byte #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-byte</td>
<td>0___ ___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-bytes: 110_ ___</td>
<td>10_ ___</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-bytes: 1110 ___</td>
<td>10_ ___</td>
<td>10_ ___</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-bytes: 1111 0___</td>
<td>10_ ___</td>
<td>10_ ___</td>
<td>10_ ___</td>
<td></td>
</tr>
</tbody>
</table>

For all single-byte characters, the ASCII character encoding is used. This means ‘a’ is still 0x61. Unicode characters are represented by U+ followed by the hex value, like U+61.

**Example:** ε (epsilon) is defined as U+03B5. How do we encode this?

---

I received the following binary message encoded in UTF-8:

0100 1000 0110 1001 1111 0000 1001 1111 1000 1110 1000 1001

1. What is the hexadecimal representation of this message?

2. What is the character length of this message?

3. What does the message say?

...what technique did we just apply to find the Unicode character code?

---

Finally, UTF-8 has seen universal design due to several brilliant features: