CS 340
Computer Systems

#2: Character Encodings and Binary Math

Jan. 19, 2023 · Wade Fagen-Ulmschneider

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.[*: in ASCII]
  (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.

ASCII Character Encoding Examples:

Binary	Hex	Char.	Binary	Hex	Char.
`0b 0100 0001`	`0x41`	`A`	`0b 0110 0001`	`0x61`	`a`
`0b 0100 0010`	`0x42`	`B`	`0b 0110 0010`	`0x62`	`b`
		`C`			`c`
		`D`			`d`
`0b0010 0100`	`0x24`	`$`	`0b0111 1011`	`0x7b`	`{`

...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 _______.
- Standard is **ISO/IEC 10646** (Updated annually!).

Technical Details of UTF-8 Encoding

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

Length	Byte #1	Byte #2	Byte #3	Byte #4
1-byte	`0___ ____`			
2-bytes:	`110_ ____`	`10__ ____`		
3-bytes:	`1110 ____`	`10__ ____`	`10__ ____`	
4-bytes:	`1111 0___`	`10__ ____`	`10__ ____`	`10__ ____`

Unicode characters are represented by **U+##** *(where ## is the hex value of the character encoding data)* and all 1-byte characters match the ASCII character encoding:

- `'a'` is ASCII `______`, or `________`.

Example: ε (epsilon) is defined as **`U+03b5`**. How do we encode this?

Example: 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 **byte length** of this message? _______

3. What is the **character length** of this message? _______

4. What does the message say?

02/utf8-binary.c

```c
unsigned char message[] = {
  0b01001000, 0b01101001, 0b11110000,
  0b10011111, 0b10001110, 0b10001001, 0
};
printf("%s\n", message);
```

Bit Manipulation: Binary Addition

For the past two lectures we have focused on the first foundation: **DATA**. Today, we are going to begin the transition away from data and into how data applies to the **CPU.** Binary addition work just like decimal addition, but with only `0`s and `1`s:

```
  0b 010011          0b 0011
+ 0b 001001        + 0b 0111
```

Negative Numbers: ______________________________

```
  0b 010011          0b 0011
- 0b 001001        - 0b 0111
```

Two's Complement

The Two's Complement is a way to represent signed (ex: positive vs. negative) numbers in a way ______________________________!

*For simplicity, let's imagine running on an **7-bit machine**:*

-17 =

-4 =

-1 =

```
  42          18
- 18        - 42
```

```
 -42          31
-  32       + 42
```

Overflow Detection in Two's Complement:

Towards Multiplication

With Two's Complement, we can add and subtract numbers! What about more complex operations?

10 x 2 =

10 x 4 =

10 x 9 =

Bit Shift Operations:

1. [Left Shift]:

2. [Right Shift]: