CSE 3430

Class 3

Slide set A-2

**Floating point data**

Binary floating point numbers: How do they work?

Answer: Like decimal numbers, except the base is different!

Example: 1010.0101

We can divide the number into 2 parts: Integer/whole number part and fractional part.

Integer/whole number part: We did this before (B2U)

Fractional part: Think about decimal 0.345 - This is 345 thousandths (345/1000)

Binary 0.0101 is 5 sixteenths (5/16)

How do we get that?

**IEEE 754:** The most commonly used encoding (representation) of floating point numbers in binary.

We only look at 32-bit (single precision), but double precision works in the same way, except for the difference in the number of bits to encode the number.

32-bit floats use three fields:

Sign (1 bit) Exponent (8 bits) Mantissa (23 bits)

The slides show the 4 steps we need to do to convert a 32-bit binary floating point number encoded in IEEE 754 to decimal.

See the slides for a number of examples.

Class 4

**Character Representation**

**ASCII:** This is an old method for representing Latin alphabet and some related characters (decimal digits, punctuation, etc.)

ASCII uses **one-byte codes** for characters, but the msb is always 0.

Since only 7 bits are used, what number of characters can be represented?

See the **ASCII-Table.pdf** on Carmen in Files > Class slides > Part A.

Please don’t memorize the codes in the table! We will talk about a few very important ones later.

**Representating Larger Sets of Characters**

As we saw before, we need more bits!

A commonly used scheme today (used on many web pages on the internet) is UTF-8.

UTF-8 uses a variable byte length encoding for characters (it can be used to encode **over 1 million characters**):

Some characters use **1-byte codes** (the same as the ASCII characters)

Some characters use **2-byte codes** (the same as the ASCII characters)

Some characters use **3-byte codes** (the same as the ASCII characters)

Some characters use **4-byte codes** (the same as the ASCII characters)

See the table in Slide set A-2, on slide 30.

**Hexadecimal – Base 16:** It is sometimes used to represent values in computing, because it is more practical than binary, and we can easily and directly convert from binary to hex, or hex to binary.

**Character Strings:** See the simple example in the slides on slide 34.

**Error Detection:** One early method (still used) is **parity**. See the table on Slide 35.

BASIC IDEA OF PARITY: The 8th bit (msb) in a one-byte ASCII character code is used to add information that can be used to detect errors.

Sender of Character Date (over a network, for example): Count the number of 1’s in the ASCII code to be sent:

* If it is even, leave the msb as 0;
* If it is odd, make the msb in the ASCII code byte 1.

In this way, the total number of 1’s in any ASCII code byte sent to the receiver of the data is always even.

Receiver of the Character Data: Count the number of 1’s in each ASCII code byte received (every processor has an instruction to do this). If it is an even number, the byte was (probably) received correctly. If it is odd, one of the bits got changed, so contact the sender and request that the code be sent again.

ADVANTAGES of this method: This is a simple and fast method for detecting errors; it does not detect 100% of errors, but very, very close to 100%, because it is extremely rare that 2 bits in a transmitted ASCII code get changed when the code is being transmitted.

WHY DO ERRORS OCCUR: Networks send data using electronic (or electro-magnetic) signals, but there is always the presence of some electrical noise, which usually does not modify data, but occasionally does. “Flipped” bits are not very common but can occur.