**CSC 250 – Assignment - Data Representation - KEY**

**NOTE: Expand space as needed. DO NOT cram answers in!**

1. **(Adapted from 3.20)** Show work and/or explain. While you can probably just Google answers or use a calculator, doing as much as possible by hand provides better understanding and potential edge on a test. Given the bit pattern represented in hex by: **0x01A00030:**
   1. What decimal number does this bit pattern represent if it is a signed, two's complement integer?

**\_\_\_\_\_\_\_\_**\_\_27,263,024\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_**

Its twos complement: 00000001101000000000000000110000.

* 1. What decimal number does this represent if it is an unsigned integer?

**\_\_\_\_\_\_\_\_**\_\_\_27,263,024\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_**

The left most bit is 0, so nothing will change.

* 1. What is the hex representation of the two’s complement of **0x01A00030**.

\_\_\_\_\_\_\_\_\_\_\_0xFE5FFFD0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Its two’s complement is 11111110010111111111111111001111+1 =

11111110010111111111111111010000

1. Repeat the previous question for the bit pattern **0xD0000000.**
   1. What decimal number does this represent if it is a signed, two's complement integer?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-805,306,368\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Its twos complement: 11010000000000000000000000000000.

The left most bit is 1 which causes an overflow.

* 1. What decimal number does this represent if it is an unsigned integer?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3,489,660,928\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. What is the hex representation of the two’s complement of **0xD0000000**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0x30000000\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Its two’s complement is 00101111111111111111111111111111 +1=

00110000000000000000000000000000

* 1. What is the hex representation of the two’s complement of **0xD0000001.**

\_\_\_\_\_\_\_\_\_\_\_0x2FFFFFFF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Enter the following program in Pythontutor or any other C programming environment you like. Show and explain the output for the following values of **i**: **5, -5, 50000, -50000**

**#include <stdio.h>**

**int main() {**

**int i;**

**if (i\*i > 0){**

**printf("true, i\*i = %i", i\*i);**

**}**

**else {**

**printf("false, i\*i = %i", i\*i);**

**}**

**return 0;**

**}**

When I = 5, and -5, they both show 图形用户界面, 文本, 应用程序

描述已自动生成. The output is correct because either square of 5 or -5 is 25.

When I = 50000, and -50000, they both show 表格

描述已自动生成. The output is wrong. Because for int type variable, its range lies between -2,147,483,648, and 2,147,483,647. The result of 50000 \* 50000 = 2,500,000,000 which exceeds the maximum int value. This causes an overflow which makes the output value to be negative.

1. **(Adapted from 3.22)** By hand, determine the floating point number represented by the bit pattern **0x3C800000**? Use the IEEE 754 standard. Break down the pattern into the various components of the floating point number. I.e., sign bit, …

Bit pattern: \_\_\_\_\_\_\_\_\_00111100100000000000000000000000\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sign: \_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Biased exponent bit pattern: \_\_\_ 01111001\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Biased exponent decimal value: \_\_\_\_\_\_\_\_\_\_\_121\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Actual exponent decimal value: \_\_\_\_\_\_\_\_\_\_\_\_\_\_-6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hidden bit: \_\_\_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fraction: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal number: \_\_\_\_\_\_\_\_\_\_\_0.015625\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **(Adapted from 3.23)** By hand, determine the floating representation (the binary pattern) of the decimal number 47.125 assuming the IEEE 754 single precision format. Suggestion: Break down the pattern into the various components of the floating point number. I.e., sign bit, …
   1. 47.125 in binary: \_\_\_\_\_\_\_\_\_\_101111.001\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   2. Actual exponent in decimal: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. Biased exponent in decimal: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_132\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. 32-bit binary pattern (blocked into sign, exponent, and fraction bits):

\_\_\_\_\_\_\_\_\_0 1000 0100 01111001000000000000000\_\_\_\_\_\_\_\_\_\_\_

* 1. Binary pattern in hex: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0x423C8000\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Do this by hand: Use the format defined in IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed). For the following questions, show work and give the answer in binary and decimal and the hex representation of the final bit pattern. That is, give the bit configuration in binary and hex and give the numeric value of these bit patterns in decimal. Assume reserved exponents of 00000 (denormalized mode) and 11111 (exceptional mode if fraction is all zeros) as in single precision. Otherwise, don't forget the hidden bit. I.e., there is always a hidden 1 bit unless all you want to represent a zero, which is all zeros.
   1. What is the smallest number greater than zero that can be represented? Do this for both normalized and denormalized format. Give the bit pattern in binary and hex and the decimal value of the floating point number.

**Normalized**: Bit pattern: \_\_\_\_\_0 00001 0000000000\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hex for bit pattern: \_\_\_\_\_\_\_0x0400\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal value: \_\_\_\_\_\_\_\_\_6.1035e-05\_(2^-14)\_\_\_\_\_\_

**De-normalized**: Bit pattern: \_\_0 00000 0000000001\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hex for bit pattern: \_\_\_\_\_\_\_\_\_\_\_0x0001\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal value: \_\_\_\_\_\_\_\_\_\_\_5.9605e-08\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. What is the largest negative number (closest to zero)? Give the bit pattern in binary and hex and the decimal value of the floating point number.

**Normalized**: Bit pattern: \_\_\_\_\_\_\_1 00001 0000000000\_\_\_\_\_\_\_

Hex for bit pattern: \_\_\_\_\_\_\_\_0x8400\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal value: \_\_\_\_\_\_\_\_-6.104e-5\_\_\_\_\_\_\_\_\_

* 1. What is the smallest negative number (the most negative)? Give the bit pattern in binary and hex and the decimal value of the floating point number.

**Normalized**: Bit pattern: \_\_\_\_\_\_\_1 11110 1111111111\_\_\_\_\_\_\_\_\_\_\_

Hex for bit pattern: \_\_\_\_\_\_\_\_\_\_\_\_0xFBFF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal value: \_\_\_\_\_\_\_\_\_\_\_\_\_\_-65504\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. What is the largest positive number? Think about the gap between this value and the immediately preceding value in this floating point format (the next to the largest value). Give the bit pattern in binary and hex and the decimal value.

**Normalized**: Bit pattern: \_\_\_\_0 11110 1111111111\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hex for bit pattern: \_\_\_\_\_\_0x7BFF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Decimal value: \_\_\_\_\_\_\_\_\_65504\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Consider the following floating point:

**a = 1.5e38; b = 1.0; c = -1.5e38**

These values can be summed in various orders. Write a program (very short) and **show the code and results** that demonstrate that associativity is an important consideration. You may find reference to this in chapter 3.

图形用户界面, 应用程序

描述已自动生成

If we add a and b first, then c, it will not print the correct answer. If we add a and c first, then it will give correct output.

1. Write a C function whose argument is a float and returns an int that is the biased exponent value of the argument. Using pointers could be helpful; also, Pythontutor. Test your function with various values. Use the tool in Participation Activity 3.5.2 to check your answers. Did your function work for reasonable values and boundary conditions?

文本

描述已自动生成

1. **(Adapted from 3.29)** (Advanced/optional) I get: 2.6546875e1. Calculate the sum of 2.6125e1 and 4.150390625e-1 by hand, assuming A and B are stored in IEEE 754-2008 16-bit half precision. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, bit patterns, etc.