Homework

Due: Monday, October 25, 2021

Submit through Canvas

Complete the problems below. Show all work. Your answers must be in RED.

1. 10 points:

Show the steps needed to multiply the following binary numbers:

1002(M = multiplicand) \* 1012(Q = multiplier)

Step 0 Initialize the data M = 100 C = 0 Acc = 0 Q = 101

Fill in the remaining steps.

100

X 101

----------------

Initially: C ACC MQ

0 000 101

MDR

100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step 1: 0 000 101

+ 100

---------------------

0 100 101

>> right shift

0 010 010

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step 2: 0 010 010

+ 000 ^ lsb = 0 so no add

---------------

0 010 010

>> right shift

0 001 001

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step 3: 0 001 001

+ 100 ^ add based on lsb = 1

----------------

0 101 001

>> right shift

0 010 100

Final Answer = 0010100

1. 5 Points:

Using positional notation for fractional binary numbers convert the following binary number to decimal. You must show your work.

1010.101 This is not IEEE. There are examples of this on the slides.

(1\*2^3 + 0\*2^2 + 1\*2^1 + 0\*2^0) + (1\*2^-1 + 0\*2^-2 + 1\*2^-3)

(1\*8 + 0\*4 + 1\*2 + 0\*1) + (1\*0.5 + 0\*0.25 + 1\*-.125)

(8 + 0 + 2 + 0) + (0.5 + 0 + 0.125)

(10) + (0.625)

Final answer = 10.625

1. 5 Points:

For both single and double precision, name the three sets of bit(s) that make up the IEEE 754 floating point number. Also list the number of bits required for each of the three parts that make up the single and double precision.

Single precision: s (the sign bit, 1 bit required), k (the exponent set, 8 bits required), n (the mantissa, 23 bits required)

Double precision: s (the sign bit, 1 bit required), k (the exponent set, 11 bits required), n (the mantissa, 52 bits required)

1. 10 Points:

An IEEE Floating-Point value encoded by a given bit representation can be divided into three different cases with the third having two variants depending on the value of the mantissa (frac).

Name and describe each case.

* 1. Normalized – this is when the bit pattern of the exponent portion is neither all 0s or all 1s and is the most common case.
  2. De-normalized – This is when the exponent field is 0. It provides a way to represent the number value 0 and numbers very close to 0.0, as it provides the property known as gradual underflow.
  3. Special Case
     1. Infinity – this occurs when the exponent is all 1s and the frac fields are all 0s. This represents positive infinity with S = 0 or negative infinity with S = 1. It can be used to represent results that overflow.
     2. NaN – Not a number, this is a case when the exponent is all 1s but the frac section is not all 0s. This can happen sometimes when there is uninitialized data.

1. 10 Points:

This question is going to require you to use your knowledge of signed an unsigned data. Take a deep breath and just think about what the problem is describing and asking you to do. This is not a hard problem, but it may seem that way at first.

Suppose the function below is supposed to determine whether one string is longer than another. The function makes use of the string library, more specifically the function strlen which has the following declaration.

/\*prototype for library function strlen\*/

size\_t strlen (const char \*s);

/\*Determine whether string **s** is longer than string **t** \*/

int strlonger(char \*s, char \*t){

return strlen(s) – strlen(t) > 0;

}

When tested with various data samples, the output just does not seem correct. Further investigation shows that, when compiled as a 32-bit program data type size\_t is defined (via typedef) in header file stdio.h to be unsigned. With this information answer the following questions.

1. For what cases will the function produce an incorrect result?
   1. The function will produce an incorrect result when the first operand (strlen(s)) is less than the second, strlen(t).
2. Explain how this incorrect result happens.
   1. In C, the unsigned values associated with a function are not automatically promoted to a signed value when doing an arithmetic action. When two unsigned values are subtracted, the maximum possible result is returned so it could end up returning the two lengths added together rather than subtracted because the sign is unknown.
3. Show how to fix the code so that it will work reliably.
   1. A way to fix the code would to be to assign each strlen to an int variable and then do the computation with the integer variables, this would convert the lengths to their signed forms. Then, you would no longer have to worry about the weird unsigned behavior during the computation.
4. 10 Points:

Consider the following code that attempts to sum the elements of an array, where the number of elements is given by parameter length.

float sum\_elements(float a[], unsigned length){

int i;

float result = 0;

for(i = 0; i <= length-1; i++)

result += a[i];

return result;

}

This code has a bug in it. When calling this function passing 0 as the length, I get a Bus error or Segfault, depending what machine I am running.

Explain why the error occurred? How can you fix the problem?

The error occurred because length is unsigned. This turns the length – 1 statement into a statement where two unsigned numbers are being subtracted. Because both types are of the same rank, the unranked type for length is used for both expressions. Wraparound rules will be followed when converting to a signed type and so we will end up with a very large number being subtracted from length. In order to avoid this error, the length variable should be signed rather than unsigned.

1. 10 Points:

In class we talked about how to determine if two unsigned integers could be added without an overflow. You are to implement the following function that will return 1 if the two unsigned integers can be added without causing an overflow. Test your algorithm with a short program. You must test with a testcase that will fail and one that will pass. Put a screenshot of your program along with the output in this document.

**Text

Description automatically generated**

**Text

Description automatically generated**

1. 10 Points:

In class we talked about how to determine if two signed integers could be added without an overflow. You are to implement the following function that will return 1 if the two signed integers can be added without causing an overflow. You must test with a testcase that will fail and one that will pass. Put a screenshot of your program along with the output in this document.

Text

Description automatically generated

Text

Description automatically generated

Chapter 3

1. 5 Points:

Describe what the compile flag -Og does.

The compile flag -Og tells the compiler to use a level of optimization that produces machine code that follows the structure of the original C code. This can optimize the debugging experience as it offers a reasonable level of optimization while maintaining fast compilation and a good debugging experience.

1. 5 Points:

Describe the significance of the compile flag -S.

The compile flag -S creates a file of assembler code or in other words a .s file. With this the linker is not run.

1. 5 Points:

Describe the significance of the compile flag -c.

Using the compile flag -c will compile and assembly the code. It will produce the .o flag but not the executable. It is helpful with incremental compilation.

1. 5 Points:

If I want to be able to understand the content of an “.o” file I can use a disassembler. I mentioned two disassemblers in class. Only one of them can be used with linux and Mac OS. What command would I use to view the content of a “.o” file on a linux architecture.

To view the content of a “.o” file on Linux architecture, the command objdump -d is used. Objdump is a program on Linux and Mac that is a disassembler that will help us inspect the machine code.

1. 5 Points:

Much of the success of computers in the last 60+ years has been due to the use of \_transistors\_\_\_\_\_.

1. 5 Points:

How many general-purpose registers does x86-64 have?

X86-64 has 16 general purpose registers.