CS61c Midterm Spring 96

Question 1 (2 points):

Subtract 0x001f from 0xfff7, interpreting each as a 2's complement 16-bit integer. Express the answer in decimal and hexadecimal. Show your work.

Question 2 (2 points):

Two lab partners Harry and David are arguing. Harry says "All integers greater than 0 exactly divisible by 6 have exactly 2 1's in their binary representation." David disagrees. He says, "No, but all such numbers have an even number of 1's in their representation."

Do you agree with Harry or David, or neither? (Why).

Question 3 (4 points):

The instruction **sllv $8, $9, $10** uses the value in register 10 as the shift amount. Actually, it uses only the least significant 5 bits. Why doesn't it use ALL the bits?

Another way of doing the same SLLV would be to do the mysterious:

```
lw $8, shifter # first instruction
and $8, mask # where mask is 0xfffff83f
andi $10, 0x1f
sll $10, $10, 6
or $8, $8, $10 # fifth instruction
sw $8, shifter # last instruction
shifter:
sll $8, $9, 0
```

In particular,
What part of the word is masked out by the first AND instruction?
What is in register $8 after the second instruction?
What is in register $8 after the fifth instruction?
Why do you suppose this code sequence is a bad idea?

Question 4 (3 points):

Recall that in the homework for week 4 you translated C code for conversion of decimal numbers into a simplified version of Roman numerals. The program is reproduced in its entirety at the end of this test.

At the end of the C program there are these two statements:

```
*currentRoman = 0;
printf ("Roman numeral equivalent = %s\n", roman);
```

What does that first statement do, and why?

Assuming that your input was 123, and the statement ***currentRoman = 0;** was omitted, what would be printed?

Question 5 (4 points):

A "real" lisp allows one to write **(+ 1 2 3 4)**, and our micro-lisp will not. This question asks what changes you would make to your micro-lisp interpreter to allow some functions to have a variable number of arguments.

Do *not* write code. Explain in a few complete English sentences what changes you would have to make to data structures (which!). Also state which parts of the micro-lisp interpreter code would have to be changes, and how.

Question 6 (4 points):

Convert the following fragment of a C program to MAL. The program counts the number of 1 bits in a word w:

```
unsigned int w, count;

count=0;
while (w != 0)
   {count = count + (w&1);
   w = w > > 1;}
```

You can assume that the variables are in registers, as follows:

$19 count
$18 w

Depending on how you translated the shift operation into MAL, this loop might not terminate if w is, for example, 0x80000000. Why?

Question 7 (2 points):

Given an implementation of a dialect of the C programming language which has a only one "floating-point number" representation named **float**, write a C or C++ program that prints out whether this representation is IEEE single-precision or double-precision (Assume that those are the only possibilities.) There are many possible solutions. Keep it simple.

Question 8 (3 points):

Draw a logic circuit that takes three input bits A, B, and C, and produces an output that is equal to C if A and B are not equal. It is equal to 0 otherwise. Use AND, OR, and, and NOT gates as the building blocks.

This is the Roman Numeral program discussed in question 4.

```
#include <stdio.h>

main () {
   int decimal;

   char romanDigits[8] = "mdclxvi";
   char* currentDigit = romanDigits;
   int romanValues[8] = {1000, 500, 100, 50, 10, 5, 1};
   int* currentValue = romanValues;

   /* longest Roman numeral < 5000 has 19 digits. */
   char roman[20] = "-------------------";
      /* hyphens are ddebugging */
   char* currentRoman = roman;

   printf ("Please type a positive integer less than 5000: ");
   scanf ("%d", &decimal);
   if (decimal <= 0) {
      exit(1);
   }

   while (decimal > 0) {
      if (decimal >= *currentValue) {
         *(currentRoman++) = *currentDigit;
         decimal = decimal - *currentValue;
      } else {
```

```
        currentDigit++;
        currentValue++;
      }
  }
  *currentRoman = 0;
  printf("Roman numeral equivalent = %s\n", roman);

}
```