CS61C B. Harvey
Fall 2003 Midterm Exam 1 J. Wawrzynek

1) (6pts) Number Representation
   a. (1/2 pt each) Given the following choice of representation for signed numbers:

      a. 1’s complement
      b. 2’s complement
      c. Sign and magnitude
      d. None of the above

   If numbers are 8 bits wide, for each of the bit patterns shown below, write the letter (a,b,c,d) corresponding to the representation in which each is interpreted as -23(base 10)

   11101001____
   10010111____
   11101000____
   11101010____

   b. (1pt) What is the value in decimal of the most negative 8-bit 2’s complement integer?

   c. (1pt) What is the value in decimal of the most positive 8-bit unsigned integer?

   d. (1pt) Write out in hex the 16-bit result of adding the following 2’s complement numbers: 0xFA25 + 0xB705

   e. (1pt) Does the add operation in (d) result in overflow?

2) (6pts) C short answer:

a. (3pts) Consider the following C program.

```
#include <stdio.h>

char* set(char c, int i) {
    /* See below for line to insert here */
    str[i] = c;
    Return str;
}

int main(){
    char* output;

    output = set('o', 2);
    output = set('w', 0);
    output = set('r', 1);

    printf("%s", output);

    return 0;
}
```

For each of the following lines inserted as indicated into procedure **set**, what is printed when the program executes? (If the program causes an error during compilation, say "compilation error"; if it causes an error or undefined results while running, say "runtime error.")

a1) static char str[] = "thing";

a2) char str[] = "thing";

a3) char *str = malloc(6);
strcpy(str, "thing");

b. (2pts) Given the following declarations:

char a[14] = “pointers in c”;
char c = ‘b’;
char *p1 = &c, **p2 = &p1;

<u>Cross out</u> any of the following statements that are <u>not</u> correct C:

p1 = a + 5;

&p1 = &a[0];

p2 = a;

*(a + 10) = ‘t’;

*p2 = %c;

c. (1pt) The C language allocates call frames on the stack instead of on the heap because doing so (circle one)

i. Simplifies allocation and freeing of frames

ii. Speeds up access to local variables, or

iii. Both?

3) (5pts) MIPS short answer:

a. (1pt) For the following MIPS assembly language program:

```
loop: addi $t0, $t0, -1
      bne  $t0, $zero, loop
```

Translate the second instruction into MIPS machine language and write it in hex.

b. (1pt) Which best describes the reason that we maintain the stack pointer in a register? (circle one)

i. The hardware forces use of a stack pointer.
ii. We need a local pointer because we are often limited to relative addressing.
iii. We need functions to have addresses for use with function pointers in C.
iv. The stack pointer is just a legacy concept that no longer serves a purpose.

c. (1pt) Write the MIPS assembly language instructions corresponding to the following pseudo-instruction: addi $s0, $s1, 0x7FF333

d. (2pts) Given the declaration for a **struct point**:

```
struct point {
        union { int i; double d; } x;
        union { int I; double d; } y;
} p[10];
```

Consider the following C code:

```
int I;

for(i=0; i<10; i++) {
        p[i].x.i = 0;
        p[i].y.i = 0;
}
```

Fill in the blanks in the following translation of this C code into MIPS:

```
        la      $8, p               # $8 points to p[0]
        addi    $9, $8, ____        # $9 points to p[10]
L1:     bge     $8, $9, L2          # done if past end of array
        sw      ___, 0($8)          # p[i].x.i = 0
        sw      ___, ___($8)        # p[i].y.i = 0
        addi    $8, $8, ____        # i++
        b L1
L2:
```

4) (4pts) Floating point.
   a. (3pts) Consider a special floating-point representation that is the same as IEEE single and double precision (with the largest exponent value representing infinity and NaN), except that the **exponent** field is 3 bits and the **significand** field is 4bits, for a total of 8 bits (once we include the sign bit). With this representation:
      i. What is the value in base 10 of the largest representable positive number? (infinity and NaN are not numbers.)
      ii. What is the value in base 10 of the smallest representable positive number(not zero)? (infinity and NaN are not numbers. You may represent the result as a fraction rather than as a decimal.)
      iii. How many numbers total can be represented? (infinity and NaN are not numbers.)
   b. (1pt) In IEEE single precision floating point, which is greater: (circle one)
      i. The number of representable numbers between 1 and 2 inclusive.
      ii. The number of representable numbers between 2 and 3 inclusive.
      iii. They are the same.

5) (2pts) Performance.
   a. (1pt) A 500MHZ machine takes 3 clock cycles for every instruction. You wish to make sure that your program completes in less than 15 microseconds. How many instructions should you limit your program to?
   b. (1pt) Consider a floating point processor. The typical instruction stream contains 30% FMUL, 60% FADD, 10% FDIV operations. Each of these instructions takes the following number of cycles to execute:

      FMUL: 2
      FADD: 1
      FDIV: 10

   What is the average cycles per instruction (CPI)?

6) (2pts) Memory Allocation:

   Suppose that you are writing the memory allocator for an application in which only one size of memory block is ever requested or allocated.

   For each of the following concepts, enter YES if the concept is still relevant even in this situation (all blocks the same size), or NO if the concept is not relevant in this situation:

   a) fragmentation ______
   b) list of free blocks ______
   c) coalescing free blocks ______
   d) garbage collection ______

7) (5pts) MIPS coding:

Here is some C code to sort a list using the mergesort algorithm:

```
struct node {
        int value;
        struct node *next;
};

struct node *mergesort (struct node *list) {
        if (list == 0 || list->next == 0) {
                return list;
        return merge(mergesort(evens(list)),mergesort(odds(list)));
}
```

Translate this procedure into MIPS assembly language, following our standard conventions for register use (arguments in registers, not stack, whenever possible). You don't have to write the helper procedures **evens**, **odds**, or **merge**.