Homework 3

Due Date: 10/3/17

Problem 1

Consider the following program:

```
L1:   sw   $t0,2052($t1)
L2:   addi $s0,$t3,-3
      beq  $t2,$t3,L4
L3:   beq  $s1,$s2,L2
L4:   j    L1
```

Assuming the above program is loaded at 0x1234ABC4, determine the machine code at each line. Please state your answer in binary or hex. (25)

Problem 2

This problem is paraphrased from an earlier edition of P&H.

Consider the following code used to implement the instruction:
foo $s0, $s1, $s2

```
mask:   .word 0xfffff83f
start:  lw    $t0,mask($zero)
        lw    $s0,shifter($zero)
        and   $s0,$s0,$t0
        andi  $s2,$s2,0x1f
        sll   $s2,$s2,6
        or    $s0,$s0,$s2
        sw    $s0, shifter($zero)
shifter:sll   $s0,$s1,0
```

Add *meaningful* comments to the code and write a paragraph describing how it works. Why do you suppose that writing "self-modifying code" such as this is a bad idea (and often times not actually allowed)?

Explanatory Notes:

- Labels are equivalent to immediates corresponding to the address that the OS will decide for them. For example, if the OS decides to start the program at 1000_{10}, mask is the immediate 1000_{10}.
- You may assume that the program is loaded at an address whose upper 16 bits are 0s. This assumption is needed since MIPS immediates are 16-bits, and references to

shifter/mask are immediates interpreted as the lower 16-bits of shifter/mask. You may also assume that shifter and mask both have a msb (bit 15) of 0 - this assumption is needed since offsets are sign-extended.

- We have not covered the .word directive yet. This is an assembler directive, which is essentially an instruction to the assembler (NOT an assembly language instruction). In this case, it tells the assembler to reserve a word of memory initialized to 0xfffff83f. Mask is the address of this word (just as start is the address of the word in memory containing the machine code for that lw instruction). (25)

Problem 3

A) Translate into MIPS, while preserving the while loop structure:

```
void string_print(char *print_me)   {
        while (*print_me != '\0')    {
                 print(*print_me);
                print_me++;
        }
}
```

(25)

B) Translate into MIPS, while preserving the for loop structure (your function is given the string length):

```
void string_print(char *print_me, int slen) {
        for (int i = 0; i < slen; i++) {
                print(*(print_me+i));
        }
}
```

(25)

Note: Please preserve the function call conventions taught in class.