CS 61C Fall 2015
Guerrilla Section 2: MIPS

Question 1 (SP14 Section 4): *MIPS Warm-Up.*

1. How should $sp be used? When do we add or subtract from $sp?

2. Which registers need to be saved or restored before using jr to return from a function? 3. Which registers need to be saved before using jal?

4. How do we pass arguments into functions?

5. What do we do if there are more than four arguments to a function?

6. How are values returned by functions?

Question 2: It's a bloody *MIPStery. . .*

a) Write the MAL MIPS function reverse_str(char *string, int string_length), that can reverse strings (with an even length) in-place. The MIPS should be non-delayed branch, and you will probably not use all the lines. In your solution, register $a0 should signify the parameter char *string and register $a1 should signify the parameter int string length.

```
reverse_str:beq $a1 $0 done
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            _____________________________
            j reverse_str
done:jr $ra
```

b) Complete the code below, using at most two TAL MIPS instructions, so that the function returns false if $a0 contains an R-type instruction and true otherwise.

```
NotRType: _____________________________
          _____________________________
          _____________________________
```

Question 3: ***"free at last, thank gosh we are free at last…"***

We wish to free a linked list of strings (example below) whose nodes are made up of this struct. Complete the code below; we have started you off with some filled in. You may use fewer lines, but do not add any.

```
// Assume compiler packs tightly
struct node {
        char *string;
        struct node *next;
};

void FreeLL(struct node *ptr) {
        if (ptr == NULL) return;
        else {
                FreeLL(ptr->next);
                free(ptr->string);
                free(ptr);
        }
}
```

FreeLL:beq ____, _____, NULL_CASE

_____

_____

_____

_____

_____

jal FreeLL

lw $a0 0($sp)

_____

_____

_____

_____

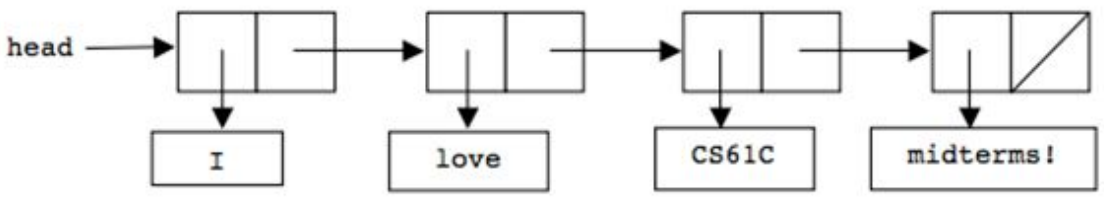
jal free

_____

_____

_____

NULL_CASE:jr $ra



Question 4 (su13 q2): *MIPStifying* (9 points, 20 minutes)

Answer the questions below about the following MIPS function. Answer each part separately, assuming each time that `mystery()` has not been called yet.

```
        mystery:
1             andi  $a0, $a0, 3
2             ori   $t0, $0,  1
3             sll   $t0, $t0, 6
4       Lbl1: beq   $a0, $0,  Lbl2
5             sll   $t0, $t0, 5
6             addi  $a0, $a0, -1
7             j     Lbl1
8       Lbl2: la    $s0, Lbl3
8             lw    $s1, 0($s0)
9             add   $s1, $s1, $t0
10            sw    $s1, 0($s0)
11      Lbl3: add   $v0, $0,  $0
12            jr    $ra
```

A. Which instruction (number) gets modified in the above function?
B. Write an equivalent arithmetic (not logical) C expression to instruction 1. a0 = ____

C. Which instruction field gets modified when mystery is called with $a0 = 3?

D. How many times can mystery(2) be called before the behavior of mystery() changes?

E. How many times can mystery(0) be called before the behavior of mystery() changes?

F. A program calls mystery with the following sequence of arguments: 0, 1, 2, 3, 4, 5. What MIPS instruction gets stored in memory?

Question 5 (SP07 Final M1): Do You Remember?

Decode the binary numbers into MIPS instructions *with proper register names* (`$s0`, `$t0`, etc.).If there are any memory addresses, represent them in hex.

Address	**32-bit Binary Instruction**	**Type (R, I, J)**	**MIPS Instruction w/args**
0xAFFFFFF8	0000 0001 0000 1000 0100 0000 0010 0110		
0xAFFFFFFC	0001 0100 0000 1000 1111 1111 1111 1110		
0xB0000000	0000 1000 0000 0000 0000 0000 0000 0001		
0xB0000004	...whatever...	whatever	ori $v0, $0, 0x61C
0xB0000008	...whatever...	whatever	jr $ra

You can replace the first instruction with a *new* instruction and save 2 clock cycles on a single-cycle non-delayed branch MIPS machine. What is it (in MIPS)? *Careful!*

_____