CS 61C Fall 2015
Guerrilla Section 2: MIPS Solution (9/26 & 9/29)

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

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

   `$sp` points to a location on the stack to load or store into. Subtract from `$sp` before storing, and add to `$sp` after restoring.

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

   All `$s*` registers that were modified during the function must be restored to their value at the start of the function.

3. Which registers need to be saved before using `jal`?

   `$ra`, and all `$t*`, `$a*`, and `$v*` registers if their values are needed later after the function call.

4. How do we pass arguments into functions?

   `$a0`, `$a1`, `$a2`, `$a3` are the four argument registers.

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

   Use the stack to store additional arguments

6. How are values returned by functions?

   `$v0` and `$v1` are the return value registers.

Question 2 (fa14 final): *MIPS*

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
                ____addu $t0 $a0 $a1_____________
                ____addiu $t0 $t0 -1_____________
                ____lbu $v0 0($t0)_______________
                ____lbu $v1 0($a0)_______________
                ____sb $v0 0($a0)________________
                ____sb $v1 0($t0)________________
                ____addiu $a0 $a0 1______________
                ____addiu $a1 $a1 -2_____________
                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.

```
lui $t0, 0xFC00 or srl $v0 $a0 26
and $v0 , $a0, $t0
jr $ra
```

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 $a0, $0, NULL_CASE
           addiu $sp $sp -8
           sw $ra 4($sp)
           sw $a0 0($sp)
           lw $a0 4($a0)
           ------------------------------
           jal FreeLL
           lw $a0 0($sp)
           lw $a0 0($a0)
           jal free
           lw $a0 0($sp)
           ------------------------------
           jal free
           lw $ra 4($sp)
           addiu $sp $sp 8
           ------------------------------
NULL_CASE:      jr $ra
```

Question 4 (su13m1 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?
   < line 11: add $v0, $0, $0 >

B. Write an equivalent arithmetic (not logical) C expression to instruction 1. a0 = ____
   <a0 % 4 >

C. Which instruction field gets modified when mystery is called with $a0 = 3?
   <Executing mystery with $a0 = 3 results in $t0 being shifted left by 21. The 1 bit in $t0 was aligned with the last bit of the rs field, so the addition incremented rs by 1, changing $0 to $at>

D. How many times can mystery(0) be called before the behavior of mystery() changes?
   <31 times because the $a0 field is written into the shamt field, which as 5 bits (can be incremented up to 2^5-1 = 31)

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

   add $a0, $at, $at
   The first instruction takes the modulus of $a0 by 4, so it was equivalent to calling the function with arguments 0, 1, 2, 3, 0, 1. Thus, rs and rt incremented by 1 while rd and shamt are incremented by 2.

Question 5 (sp07 final q2): *MIPStifying* (9 points, 20 minutes)

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

b) 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	R	xor $t0, $t0, $t0
0xAFFFFFFC	0001 0100 0000 1000 1111 1111 1111 1110	I	bne $0, $t0, -2
0xB0000000	0000 1000 0000 0000 0000 0000 0000 0001	J	j 0xB0000004
0xB0000004	...whatever...	...whatever...	ori $v0, $0, 0x61C
0xB0000008	...whatever...	...whatever...	jr $ra

Can't use "j 0xB0000004" in 0xAFFFFFF8, since can't jump across 256MB line (0xA..->0xB..)