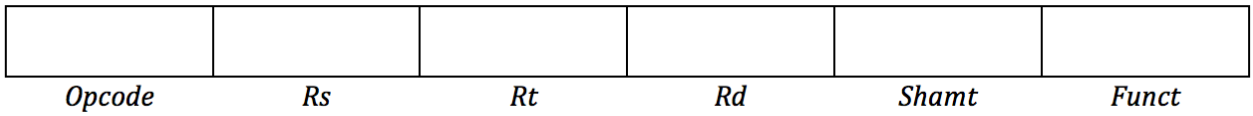
CS 61C Summer 2016
Guerrilla Section 2: MIPS & CALL

Question 1: What's that funky smell? Oh, it's Potpourri.

a) 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

b) A variant of the MIPS instruction set has been developed on the new 64-bit system for cloud computing. To emphasize the "big" in big data, the size of the "opcode" field has been increased by one bit, and the number of registers (whose width is now 64-bits) was quadrupled. Assume that instructions "expand" the rightmost field to fill all 64 bits. so here, the "funct" field would expand; in an I-type instructions, the Immediate would expand. You may use the boxes below as scratch space.



a) How many total instructions can we have? Please leave your answer as an expression involving powers of 2.

b) What is the maximum amount of bytes we could change the PC by with a single branch instruction?
(Express your answer in IEC format, e.g., 32 Kibi, 16 Mebi, etc.)

Question 2 (Fa15 Q5) MIPS Sleuth.

mystery, a mysterious MIPS function outlined below, is written without proper calling conventions. ***mystery*** calls a correctly written function, ***random***, that takes an integer i as its only argument, and returns a random integer in the range [0, i - 1] inclusive.

```
1     mystery:    addiu $sp $sp_____________________
2                 _____________________________________
3                 _____________________________________
4                 _____________________________________
5                 _____________________________________
6                 _____________________________________
7                 addu $s0 $0 $0
8                 move $s1 $a0
9                 move $s2 $a1
10    loop:       srl $t0 $s0 2
11                beq $t0 $s2 exit
12                subu $a0 $s2 $t0
13                jal random
14                sll $v0 $v0 2
15                addu $v0 $v0 $s0
16                addu $t0 $s1 $s0
17                addu $t1 $s1 $v0
18                lw $t2 0($t0)
19                lw $t3 0($t1)
20                sw $t2 0($t1)
21                sw $t3 0($t0)
22                addiu $s0 $s0 4
23                j loop
24    exit:       _____________________________________
25                _____________________________________
26                _____________________________________
27                _____________________________________
28                _____________________________________
29                _____________________________________
30                _____________________________________
```

1) Fill in the prologue and the epilogue of this MIPS function. Assume that random follows proper calling conventions, and that it may make its own function calls. You may not need all of the lines.

2) What operation does this function perform on an integer array? Assume that both the integer array and the length of the array are passed into the function.

3) Would this function work as expected if a string was passed into the function instead? Write down the line numbers of all lines of MIPS code that must be changed (if any at all), so that the function works correctly on strings. Do not write down any extraneous line numbers.

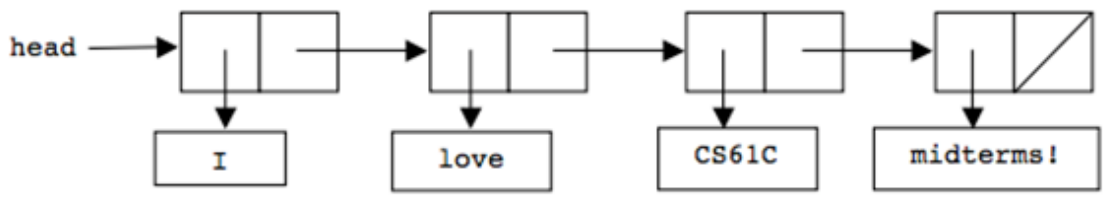
Question 3: *"free at last, thank gosh we are free at last…"*

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 (Fa15 Q7) After this, you're CALL done! (9 points)

Connect the definition with the name of the process that describes it.

a) Compiler
b) Assembler
c) Linker
d) Loader

____1) Outputs code that may still contain pseudoinstructions.
____2) Takes binaries stored on disk and places them in memory to run.
____3) Makes two passes over the code to solve the "forward reference" problem.
____4) Creates a symbol table.
____5) Combines multiple text and data segments.
____6) Generates assembly language code.
____7) Generates machine language code.
____8) Only allows generation of TAL.
____9) Only allows generation of binary machine code.