University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2015 Instructors: Vladimir Stojanovic, John Wawrzynek 2015-09-06

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	
First Name	
Student ID Number	
CS61C Login	**cs61c-**
*The name of your **SECTION** TA (please circle)*	Alex | Austin | Chris | David | Derek | Eric | Fred | Jason | Manu | Rebecca | Shreyas | Stephan | William | Xinghua
Name of the person to your LEFT	
Name of the person to your RIGHT	
*All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. **(please sign)***	

Instructions (Read Me!)

- This booklet contains 7 numbered pages including the cover page. **The back of each page is blank and can be used for scratch-work, but will not be looked at for grading.** (i.e. the sides of pages without the printed "SID: _____" header will not even be scanned into gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 80 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one handwritten 8.5"x11" page (front and back) of notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Points Possible	10	10	15	15	15	16	9	90
Points Earned								

SID: _____

Clarifications during the exam:

Q3-3. The node should free itself also.

- if you can traverse to a node through a series of prev and/or next, that node is reachable

Q4. If you see a 90 under the “Exit” label, please ignore it.

beargit redux - In is_commit_msg_ok(), int i is declared before the for loop.

- if you need to use it, you can use $sp, $ra, $gp, $fp

Q6-3 - Give the address in hex

Q6-1 - Instructions are still 32 bits

SID: _____

Q1: A *dozen* ways to ask about *bits* (10 points)

1) For a 12-bit integer represented with two's complement, what is the:

a) Most positive value (in decimal): _____

b) Binary representation of that number: _____

c) Most negative value (in decimal): _____

d) Hex representation of that number: _____

e) In general, for an n-bit, two's complement integer:

i) What is the most positive you can represent, in decimal? _____

ii) What is the most negative you can represent, in decimal? _____

2) Fill in the blank below so that the function `mod16` will return the remainder of `x` when divided by 16. The first blank should be a bitwise operator, and the second blank should be a single decimal number:

```
unsigned int mod16(unsigned int x) {
    return x ________ _________________;
}
```

SID: _____

Q2: Wow! If only you could *C* the *main memory* (10 points)

Consider the following C program:

```
int a = 5;
void foo(){
    int temp;
}
int main()
{
    int b = 0;
    char* s1 = “cs61c”;
    char s2[] = “cs61c”;
    char* c = malloc(sizeof(char) * 100);
    foo();
    return 0;
};
```

1) Sort the following values from least to greatest: **`&b, c, b, &temp, &a`**.

2) For each of the following values, state the location in the memory layout where they are stored. Answer with *code*, *static*, *heap*, or *stack*.

`s1`	
`s2`	
`s1[0]`	
`s2[0]`	
`c[0]`	

SID: _____

Q3: *Links*, *Links*, and *Lists* (15 points)

Here, we have a two-sided linked list, where each node has a reference to both the previous and next node in the list. The `HEAD` of the list is defined as the node with a `NULL prev` pointer, and the `TAIL` of the list is defined as the node with a `NULL next` pointer.

```
struct ll_node {
    unsigned short id;
    struct ll_node *prev;
    struct ll_node *next;
};
```

For the remainder of the questions, assume that the `struct ll_node` is tightly-packed (i.e, all its elements are contiguous in memory).

1) We are given a `struct ll_node` **`current_node`**. Assuming that the type `unsigned short` is 2 bytes wide and that we are working with a 32-bit memory address space, what can we expect the function call **`sizeof`**`(current_node)` to return?

_____

2) Assume that we have access to **`id_addr`**, the address of the id of **`current_node`** in memory. Using only **`id_addr`**, fill in the blank line so that **`next_node`** is equivalent to **`current_node.next`**.

```
unsigned short *id_addr = &(current_node.id);
struct ll_node *next_node = *(_____________________________________);
```

3) Now, fill in the blanks to complete this function that, given a random node in the list, frees all reachable nodes from that given node. Keep in mind that the node may be the `HEAD` of the list, the `TAIL` of the list, or a node in between the `HEAD` and `TAIL`. You may not need every blank.

```
void free_twosided_ll(struct ll_node *node) {
    if (node != NULL) {
        if (node->prev != NULL) {
            ____________________________________________________
            ____________________________________________________
        }
        if (node->next != NULL) {
            ____________________________________________________
            ____________________________________________________
        }
     ____________________________________________________
     ____________________________________________________
    }
}
```

SID: _____

Q4: *beargit* redux (15 points)

From project 1, you may remember the function `is_commit_msg_ok()` that you needed to implement in C. Here is a simpler rendition where commit messages are deemed okay *if and only if* those null-terminated commit messages exactly match `go_bears`. Using the **fewest number of empty lines possible**, finish writing the code below. You are only allowed to use the registers already provided **and** registers **`$t0-3`**, and **`$s0-s2`** (but you will not need all of them). Assume these registers are initialized to 0 before the call to ISCOMMITOK.

```
const char* go_bears = "THIS IS BEAR TERRITORY!";

int is_commit_msg_ok(const char* msg, const char* go_bears) {
    for (int i = 0; msg[i] && go_bears[i]; i++) {
        if (go_bears[i] != msg[i]) return 0;
    }
    if (!msg[i] && !go_bears[i]) return 1;
    return 0;
}
```

```
ISCOMMITOK:     _____________________________
                _____________________________
                ____ $t0   ____($a0)
                ____ $t1   ____($a1)
  COND:         and ________________________
                _____________________________
                _____________________________
                addiu $a0 $a0 1
                addiu $a1 $a1 1
                _____________________________
                _____________________________
  EXIT:         ____ $t2 $t0 $t1
  90            _____________________________
                li $v0 1
                _____________________________
  FAILED:       li $v0 0
  END:          _____________________________
                _____________________________
```

SID: _____

Q5: *MIPS* Sleuth (15 points)

`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.

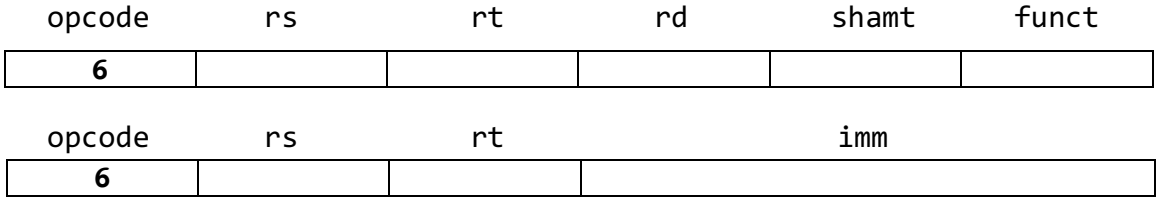
3) Would this function work as expected if a string was passed into the function instead? Write down the line numbers of all lines of MIPS code that must be changed (if any at all), so that the function works correctly on strings. Do not write down any extraneous line numbers.

SID: _____

Q6: *Registers*: bigger is not always better (16 points)

You decide that instead of having 32, 32-bit registers, you would like to build a machine with 16, 64-bit registers. You also need to make a modified MIPS instruction set for this architecture.

1) In the box below, specify the size of the fields to best utilize the 32-bit instructions on this new architecture. Do not modify the size of the `opcode`.



2) How many different R-type instructions can we now have? _____

3) If `PC = 0x061C`, what is the largest address that we can branch to? _____

4) Translate the following machine code into MIPS using your new field sizes. Use register numbers instead of register names, since we'd have to think of a new convention for the names…

`0xAE9FFFF8`

_____

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. _____