University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2013 | Instructor: Dr. Dan Garcia | 2013-03-04

☹ CS61C Midterm ☺

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

Last Name	**Answer Key**
First Name	
Student ID Number	
Login	cs61c-
Login First Letter (please circle)	a b c d e f g h i j k l m
Login Second Letter (please circle)	a b c d e f g h i j k l m n o p q r s t u v w x y z
*The name of your **LAB** TA (please circle)*	Justin Alan Paul Sagar Sung-Roa Zachary
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!)

- Don't Panic!
- This booklet contains 6 numbered pages including the cover page. Put all answers on these pages; don't hand in any stray pieces of paper.
- Please turn off all pagers, cell phones & beepers. Remove all hats & headphones. Place your backpacks, laptops and jackets at the front. Sit in every other seat. Nothing may be placed in the "no fly zone" spare seat/desk between students.
- You have 120 minutes to complete this exam. The exam is open book, no computers, PDAs, calculators.
- 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. You have 2 hours...relax.

Question	**1**	**2**	**3**	**4**	**Total**
Minutes	48	24	24	24	120
Points	**30**	**15**	**15**	**15**	**75**
Score	**30**	**15**	**15**	**15**	**75**

1) What is that funky smell? Oh, it's just Potpourri… (48 min, 30 pts)

Suppose for questions (a)-(d) that we were to modify the MIPS ISA so that it exposed 64 registers instead of 32, and adjusted the field widths of our R, I and J instruction formats to be able to address all the registers, *but did not change the size of the **opcode** or **shamt** fields*. Registers and instructions will remain 4 bytes wide.

a) At most how many instructions can a single `beq` instruction now reach? **2^{14}**

b) How many *more* addresses can now be reached from a `jal` instruction? **None**

c) How many different R-type instructions can we now have? **8**

d) If it costs 256 cents per *bit* of register memory (yikes!), how much would these 64 registers cost, in cents? Answer using IEC notation (e.g,. 4 mebicents). **512 kibicents**

Questions (e)-(g) concern the *IEEE floating point standard*.

e) What `float` is encoded by the following bits: **`0xc14c0000`**? **-12.75**
(show all work here)

f) What is the smallest positive normalized number? Number: **2^{-126}** encoded as 0x**00800000**
(show all work here)

g) Write the MAL MIPS function **`IsNotInfinity`** to return non-0 if the input is **NOT** ±∞, 0 if it *is* ±∞.

```
IsNotInfinity: sll  $a0 $a0 1   ### make –∞ and +∞ look the same
               xori $v0 $a0 0xFF000000 (or addiu 0x0100 0000)

               jr $ra
```

1) What is that funky smell? Oh, it's just Potpourri… (continued)

h) Complete the below C code so that it obeys its comments. This should work on 1986 hardware, where `int`s were only 2 bytes wide, as well on our current 32-bit MIPS machines.

```
                                  (sizeof(int)*8)
#define INT_BITS ____________________________________; // # bits in an int
// Rotates (the numeral that p points to) to the left by 0<=n<INT_BITS.
// "rotate" means the leftmost bits fall off the left, appear on the right

void drotL(unsigned int *p, unsigned int n) {
                                  *p = (*p << n) | (*p >> (INT_BITS-n))
  ____________________________________________________________________ ;
}
```

i) You define a short recursive MIPS procedure `foo` that is statically linked by two executables. Can the binary for the procedure `foo` be different in the two executables? Why, or why not?

They CAN be different because the address of foo (which will be a part of the recursive jal call) depends on where the code is placed.

j) We want to represent 10-digit phone numbers using bits. Assume that any combination of 10 digits is legal. One scheme is to represent *each digit with a certain number of bits*, then repeat it 10 times. What is the minimum number of bits required to represent a phone number using this scheme?

Ceil(Log2(10))=4 * 10 = 40

k) Using <u>any</u> scheme, what is the fewest number of bits required to "address" all phone numbers?

Log2(10,000,000,000) ➔ Log2(10 billion) ➔ 34

l) Consider the C code here, and assume the `malloc` call succeeds. Rank the following values from 1 to 5, with 1 being the least, right before `bar` returns. Use the memory layout from class; *Treat all addresses as unsigned #s.*

```
#include <stdlib.h>
int FIVE = 5;

int bar(int x) {
    return x * x;
}

int main(int argc, char *argv[]) {
    int *foo = malloc(sizeof(int));
    if (foo) free(foo);
    bar(10); // ← snapshot just before it returns
    return 0;
}
```

foo ___3___

&foo ___5___

FIVE ___1___

&FIVE ___2___

&x ___4___

2) "`free` at last, thank gosh we are `free` at last…" (24 min, 15 pts)

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);
}
```

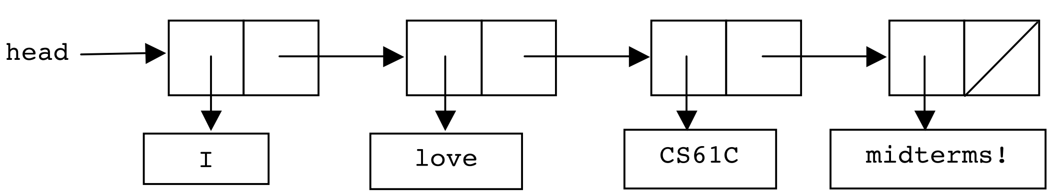
```
                   $a0     $0
FreeLL:      beq ______, ______, NULL_CASE
             addiu $sp $sp -8
             sw $ra 4($sp)
             sw $a0 0($sp)
             lw $a0 4($a0)

             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
```



Login: cs61c-__ **Answers** __

3) "*Cache*, money. Dollar bills, y'all." (24 min, 15 pts)

Suppose we have a standard 32-bit byte-addressed MIPS machine, a single direct-mapped 32KiB cache, a write-through policy, and a 16B block size.

a) Give the T:I:O breakup. **17:11:4**

b) How many bits are there per row on the cache? **16*8(data)+17(tag)+1(valid) = 146**

Use the C code below and the description of the cache above to answer the questions that follow it. Suppose that the only memory accesses are accesses and stores to arrays and that all memory accesses in the code are valid. Assume `A` starts on a block boundary (byte 0 of `A` in byte 0 of block).

```
#define NUM_INTS 32
#define OFFSET   8192   // 8192 = 2^13

int rand(int x, int y); // returns a random integer in the range [x, y)

int main(){
      int A[NUM_INTS + OFFSET];              // Assume A starts on a block boundary

      // START LOOP 1
      for ( int count = 0 ; count < NUM_INTS ; count += 1 ) { // count by 1s
            A[count]          = count;       // ACCESS #1
            A[count + OFFSET] = count+count; // ACCESS #2
      }
      // END LOOP 1

      // START LOOP 2
      for ( int count = 0 ; count < NUM_INTS ; count += 4 ) { // count by 4s now
            for ( int r = 0 ; r < 4 ; r++ ) {                 // …but do it 4 times
                  printf("%d", A[rand(count, count+4)]);
            }
      }
      // END LOOP 2
}
```

c) Hit rate for Loop 1? **0%** What types of misses are there? **compulsory, conflict**

d) Hit rate for Loop 2? **75%** What types of misses are there? **conflict**

Questions (e), (f), and (g) below are three independent variations on the original code & settings.

e) If the cache were 2-way set associative, what would be the hit rate for Loop 2? **100%**
(assume the standard LRU replacement policy)

f) If instead we removed the line labeled `ACCESS #2`, what would be the hit rate for Loop 2? **100%**

g) Instead, what's the smallest we could shrink `OFFSET` to maximize our Loop 2 hit rate? **32**
(assume we still need to maintain the same functionality)

4) You thought YOUR numbers were big? Try these! (24 min, 15 pts)

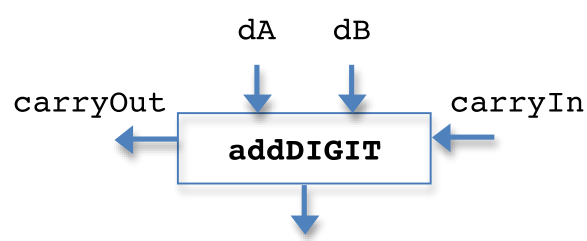
We'd like to represent infinitely long positive integers. We decide to use an array of `DIGIT`s, where a `DIGIT` is an unsigned 8-bit quantity specified by `uint8_t`. We'd also like to know where the number ends, so we'll declare `0xF` to be the *terminator* (like `0x0` is for strings). Let's see how we'd store "57":

Byte	2								1								0							
Bit	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1
Hex	0				F				0				5				0				7			

a) Fill in `addDIGIT` to make it work properly. The idea is that we'll build this small black box that can add *one* `DIGIT` as shown below, then to add the whole thing we'll virtually hook many of these together. This block simulates what we do when we are adding, say, the <u>tens</u> column on the right: **`dA`** is **`2`**, **`dB`** is **`7`**, **`carryIn`** from the ones column is **`1`**, **`carryOut`** for the hundreds column gets **`1`** and the return value is **`0`**.

```
   1 1
   1 2 8
+    7 7
   -----
     0 5
```

```
typedef uint8_t DIGIT;
DIGIT addDIGIT(DIGIT dA, DIGIT dB, DIGIT carryIn, DIGIT *carryOut) {
    uint8_t tempSum = dA + dB + carryIn;
    *carryOut = tempSum / 10;
    ______________________________
    return tempSum % 10;
}
```



b) Now, let's use `addDIGIT` to write the full adder! Unfortunately, there are bugs; indicate your changes to fix the code in the table below. We use two helpers, **`sizeofDIGIT()`**, which would return 2 for the "57" example above, and **`safeDigit()`** which takes an array `A`, `index` and `length` returning `A[index]` if we're still in array bounds (`index < length`), and `0` otherwise. `A`, `B` are never `NULL`.

```
01 DIGIT *add-DIGITs(DIGIT *A, DIGIT *B) {
02   unsigned int lenA = sizeofDIGIT(A);
03   unsigned int lenB = sizeofDIGIT(B);
04
05   DIGIT sum[max(lenA,lenB)];      /* make space for the answer */
06   DIGIT carryIn = 0, carryOut;
07   int i;
08   for(i=0 ; i < max(lenA,lenB) ; i++) {
09     sum[i] = addDIGIT(safeDigit(A,i,lenA), safeDigit(B,i,lenB), carryIn, carryOut);
10     carryIn = carryOut;
11   }
12
13   sum[i] = 0xF; /* terminator */
14   return sum;
15 }
```

Line	**Indicate the fix here and what it is (insertion, deletion, change).** All lines not nec needed.
01	Remove dash
05	`DIGIT *sum = (DIGIT *) malloc (sizeof(DIGIT) * (max(lenA,lenB)+2));`
09	`&carryOut`
12	`if (carryOut) sum[i++]=carryOut;`