University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Summer 2008 Instructor: Albert Chae 2008-07-21

CS61C Midterm

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	
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 **SECTION** TA (please circle)*	Bill Omar Richard
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 5 double-sided 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.
- Question 0 (1 point) involves filling in the front of this page and putting your name & login on every front sheet of paper.
- You have 180 minutes to complete this exam. The exam is closed book, no computers, PDAs or calculators. You may use one page (US Letter, front and back) of notes and the green sheet.
- Questions are not necessarily ordered by difficulty, so don't spend too long on any one question.
- 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 3 hours...relax.

Question	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Minutes	1	10	10	24	24	36	36	141 + 39 extra
Points	**1**	**8**	**10**	**12**	**12**	**16**	**16**	**75**
Score								

Name: _____ Login: cs61c-____

Question 1: Want to see a magic trick? I'll make this number disappear…

(8 pts, 10 min)

a) Below is a table corresponding to a few different systems for representing numbers. Fill in the twelve blanks in the indicated table. Each column should contain the same number, written different ways. If a particular number cannot be represented in a particular system, put "N/A" in the space for that number and system.

Number System	First Number	Second Number	Third Number
Decimal (base 10)	**13**		
Ternary (base 3)			
8 bit Sign and Magnitude (in hex)			**0x13**
8 bit One's Complement (in hex)			
8 bit Two's Complement (in hex)		**0x80**	

b) For the different number representation systems and number of bits given, give the numbers that are closest to positive and negative infinity.

Number System	Number Closest to Positive Infinity	Number Closest to Negative Infinity
a byte using Unsigned Integer (in decimal)		
20-bits using Sign and Magnitude (in hex)		

Name: _____ Login: cs61c-____

Question 2: My memory says I told you to call the garbage man (10 pts, 10 min)

a) Assume we are dealing with a simple 8 byte memory whose layout is currently `----A---`, where A represents some allocated memory and – represents a free block. You may assume A is the most recently allocated memory. Circle the allocation scheme(s) for which the following sequence of memory requests would fail:

```
B = malloc(3); C = malloc(3); free(B); C = malloc(4);
```

First-fit **Best-fit** **Next-fit**

Scratch Space

b) Label the following properties with the ONE garbage collection scheme they are most closely related to. Use **RC** = reference counting, **MS** = mark and sweep, and **SC** = stop and copy.

____ Requires a graph traversal

____ Keeps track of number of pointers to an object in memory while program is running

____ Can only allocate up to half the total memory on the system

____ Does not work on circular data structures

c) Label the following with its corresponding step in the CALL process: Use **CO**mpiling, **AS**sembly, **LI**nking, **LO**ading

____ Copies instructions and data into physical memory

____ Translates from High Level Language to MAL

____ Replaces pseudoinstructions and generates TAL

Name: _____ Login: cs61c-____

Question 3: My half-precision float is bloated (12 pts, 24 min)

s	eee eeee	mmmm mmmm mmmm

You are designing a 20 bit floating point data type called the bloat (bizarre float). It has a sign bit, 7 bits for the exponent field, and 12 bits for the significand field. This new data type has one other property that sets it apart from IEEE standard floating-point numbers – we want to represent more large numbers and fewer small numbers. To do this, the exponent bias will be defined to be sixty (60). The implicit exponent for denorms will be defined to be negative fifty-nine (-59). To see how this changes the numbers that we can represent, fill in the table below using this definition. Note that any rounding should be done by rounding towards zero. You may leave your answer in the form of a sum or subtraction, e.g. 2^5+2^3 – 10 would be a valid answer.

The largest positive number that can be stored?	
The smallest positive number that can be stored? (0 is not positive)	
The hex representation of -2050.75?	
How many different real numbers can be stored (do not distinguish between the 0s)?	
The range of the denormalized numbers?	
How many NaNs are there?	

Name: _____ Login: cs61c-____

Question 4: Are you a MAL fighter? Meet me in the HEXagon. (12 pts, 24 min)

Below is a short section of MAL code that you should convert to the equivalent TAL, and then assemble into the equivalent machine language representation (which you should provide in hex). Note that you may not need all of the lines provided to you. The label `Go_Here` is at absolute address 0x14c888ac.

Hint: The syntax for `beq` is: `beq rs, rt, Label`

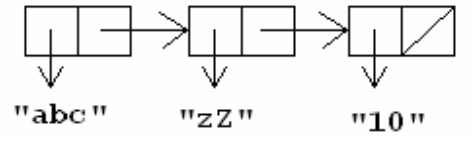
```
Go_Here:    beq     $a0, $a1, Go_Here
            addiu   $t0, $t8, 0xFFFF0001
            j       Go_Here
```

TAL	Machine Language
_____	0x _____
_____	0x _____
_____	0x _____
_____	0x _____
_____	0x _____
_____	0x _____
_____	0x _____
_____	0x _____

Name: _____ Login: cs61c-____

Question 5: Let's consjure up a belt of herbs (16 pts, 36 min)

As you may have noticed in the adventure game, players had very limited healing capabilities and were frequently mauled to death by squirrels and donkeys. We want to help the player out by building a belt for holding healing herbs. This *belt* data structure will consist of *pockets* connected in a linked list. Each *pocket* contains an *herb*, which is represented as a standard C string.



For example, this is a *belt* with 3 *pockets*.

A player heals by chewing on characters of the string, and will regain health equal to the ASCII value of each character. For example, if a character eats the entire herb "abc", that character will regain 65+66+67= 198 health. However, because we don't want to give players too big an advantage, we will make our belt feature work for monsters or other game entities as well.

To summarize:

- *belt* – a linked list
- *pocket* – a node in the linked list
- *herb* – a standard C string

a) Fill in the definition of the `pocket_t` and `add_herb()`. You may need to revise this, however, after doing part b.

```
typedef struct pocket {

    ____________________ herb;

    ____________________ next;
     // Put anything else you might need below.




} pocket_t;

void add_herb(...) {
    pocket_t *new_pocket = (pocket_t *) malloc(sizeof(pocket_t));

    // changes new_pocket->next so new_pocket is added to belt correctly
    // changes new_pocket->herb to point to malloc'ed space for the herb string

    // You should add anything here to initialize anything you added to pocket_t above




}
```

Name: _____ Login: cs61c-____

← This problem starts on the back of the previous page.

b) Write the `heal()` function, which takes as arguments:

- `belt_handle`, a `pocket_t` handle (i.e. a pointer to an entity's belt)
- `current_hp`, an `int *` that points to an entity's hp variable
- `max_hp`, an `int` representing the entity's maximum hp.

`heal()` restores the entity's health to max using herbs. The entity should begin by consuming characters from the herb in its current pocket. If the current pocket's herb is not sufficient, the entity should remove the current pocket from the belt, then move to the next pocket and continue this process until the entity reaches (but does not exceed) full health or runs out of herbs. Whenever a pocket is removed, all memory associated with it must be `free`d.

Some things to note:

* You may assume the belt has at least one pocket to start with, and that there is an herb with some edible characters remaining in that pocket.
* An entity can't partially eat a character, but it can partially eat an herb.
* You may decide whether the entity either always eats the herb right to left or always eats left to right.
* Don't use `the_player` global variable! Your code must work regardless of who owns the herb belt.

```
void heal(pocket_t **belt_handle, int *current_hp, int max_hp) {
```

```
}
```

Name: _____ Login: cs61c-____

Question 6: You got MIPS in my C question! No, you got C in my MIPS question! (16 pts, 36 min)

a) Below is the full definition of a mystery MIPS function. In the space provided, fill in the definition of the equivalent C function (including the argument list and return type). Note that the C function that you write should not only produce the same results, but it should produce those results using the same method. Note that the first argument to the function is a pointer to an integer array that is initialized as follows:

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4

```
foo:
     xor   $t0, $t0, $t0
     xor   $v0, $v0, $v0

     addiu $t1, $0, 8

foo_1:
     slt   $t2, $t0, $t1
     beq   $t2, $0, foo_2

     andi  $t3, $a1, 15
     sll   $t3, $t3, 2
     addu  $t4, $t3, $a0

     lw    $t4, 0($t4)
     addu  $v0, $v0, $t4

     srl   $a1, $a1, 4
     addiu $t0, $t0, 1

     j     foo_1:

foo_2:
     jr   $ra
```

```
_____foo(int *_____________________) {




}
```

Name: _____ Login: cs61c-____

← This problem starts on the back of the previous page.

MIPS repeated here for scratch work convenience, but it will not be graded

```
foo:                # You may put comments here, but they will not be graded
     xor   $t0, $t0, $t0
     xor   $v0, $v0, $v0

     addiu $t1, $0, 8

foo_1:
     slt   $t2, $t0, $t1
     beq   $t2, $0, foo_2

     andi  $t3, $a1, 15
     sll   $t3, $t3, 2
     addu  $t4, $t3, $a0

     lw    $t4, 0($t4)
     addu  $v0, $v0, $t4

     srl   $a1, $a1, 4
     addiu $t0, $t0, 1

     j     foo_1:

foo_2:
     jr   $ra
```

b) What is the description (or common name) of what this function does?

c) If we were to copy the above code into a new function called `new_foo` and filled the array with different values, we can make `new_foo` have the same functionality as if we were to perform a bitwise (logical) negation (~ in C) on the second argument (`$a1`) before calling the original `foo`. What values would we need to put into the array to achieve this?

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value																

Name: _____ Login: cs61c-____

THIS PAGE INTENTIONALLY LEFT BLANK FOR SCRATCH SPACE