University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2005 Instructor: Dan Garcia 2005-03-07

☺ CS61C Midterm ☺

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 z y z
*The name of your **LAB** TA (please circle)*	Andy Casey Danny Steven
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 8 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.
- 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.
- 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.

Problem	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
Minutes	0	25	30	25	25	25	25	25	180
Points	**1**	**10**	**14**	**10**	**10**	**10**	**10**	**10**	**75**
Score									

Name: ____________________________ Login: `cs61c-____`

Question 1: For those about to test, we salute you... (10 pts, 25 min.)

You're coming straight from a wild pre-midterm toga party and are not quite focused on the exam. Fear not, this question will get you warmed up and ready to rock, 61C style.

a) You "peek" into a register and find the string "`#One`" (from our cheer: *Cal is #One!*) as shown here:

#	O	n	e

What would this be if we interpreted this instead as:

…a single hexadecimal number?

…an instruction? (you may leave arguments in hex)

b) You believe the IEEE 754 Floating Point Standard could use some modification. Instead of 8 bits for the exponent and 23 bits for the significand, you suggest we give two of the exponent's bits to the significand, making it 6 and 25. What are the pros and cons of this proposal?

A pro is that we will have:

A con is that we will have:

c) Suppose we want to add a new Pseudo-Instruction "branch if float is zero": `bfz`. This instruction will take two arguments, a float (stored in a register) and a label, e.g., `bfz $a0 done`. It takes the branch if the floating point number in the register is zero (either one). To what TAL instruction(s) (from the "core instruction set") could an Assembler expand this instruction?

`bfz $a0 done`

d) For each of the allocation systems on the left, circle the column that describes its *fragmentation*:

Buddy System	causes *internal* only	causes *external* only	causes *both types*
Slab Allocator	causes *internal* only	causes *external* only	causes *both types*
K&R (Free List Only)	causes *internal* only	causes *external* only	causes *both types*

e) Three sisters are running different programs on identical machines (heap memory size *M*) using a language that supports garbage collection (GC). Each is out of memory and chooses a different garbage collection scheme. Fill in the table. All answers should be a function of *M*, e.g., *"M/7"* or *"5M"*.

What is the…	*most space* their data could take up *before* GCing?	*least* space their data could take up *after* GCing?	*most* space their data could take up *after* GCing	*most* wasted space that GCing couldn't recover?
Reference counting				
Mark and Sweep				
Copying				

Name: ____________________________ Login: cs61c-____

Question 2: Three is a magic number! (14 pts, 30 min.)

Welcome to the future! Computer scientists have invented a superior number representation method. Instead of using *binary* (base 2) numbers to represent things internally, they're using *ternary* (base 3). The value of an unsigned *n*-digit ternary number is: $\mathbf{d_{n-1} \times 3^{n-1} + d_{n-2} \times 3^{n-2} + \ldots + d_1 \times 3^1 + d_0 \times 3^0}$
Unless otherwise stated, express all your answers in base 10. Show your work to receive full credit.

a) How many distinct values can be encoded with *n* ternary digits?

b) What is the largest unsigned number that can be represented in *n* ternary digits, assuming that we want to represent all the integers from 0 through this value?

Just as we have *binary signed two's complement* to represent negative numbers also, a *ternary signed three's complement* can be developed. To give you a feel for how this would work, here are some examples using three ternary digits:

Ternary Representation	**Base 10 number**
000	0
111	13
112	-13
222	-1

c) Given the description of *ternary signed three's complement* above, answer the following questions: For an ***n*-digit** signed three's complement ternary number (n > 1), what is the…

Ternary representation of -3	**# of Positive Integers**	**# of Negative Integers**	**# of Zeros**

Just as in Project 1, we can express an *arbitrarily large* binary or ternary number as a string where each character holds one digit, and *we will always have n digits* (leading zeros added when needed).

d) Below on the left you will find `BinaryNegate`, C code used to take a binary two's complement number stored in a string and negate it. (Assume storage for `out` is allocated by the caller.) Complete `TernaryNegate` on the right to negate a ternary three's complement number.

```
/* Assume AddBinaryConstant exists */

/* Invert the bits and add one */
void BinaryNegate(char *in, char *out) {
  char *tmp = out;
  while (*in)
    *tmp++ = InvertBinaryDigit(*in++);
  AddBinaryConstant(out, 1);
}

char InvertBinaryDigit(char c) {
  return (c == '0') ? '1' : '0';
}
```

```
/* Assume AddTernaryConstant exists */

void TernaryNegate(char *in, char *out) {





}

/* Write any helpers you need here */
```

Question 2: Three is a magic number! (continued) (14 pts, 30 min)

e) Using the same notion of arbitrary-length ternary three's complement numbers stored in a string, write a C function that returns 1 if a number is negative and 0 otherwise. We've copied the 3-digit table from the previous page on the right if that helps.

Ternary Representation	Base 10 Number
000	0
111	13
112	-13
222	-1

```
int isNegative(char *in) {










}
```

Question 3: Potpourri (10 pts, 25 min)

a) Assemble the following code assuming that the label `begin` corresponds to the address `0x000E0000`. You should fill in the blanks below with the hexadecimal value for the instruction (see Instruction 13 for an example).

```
begin:  addi $v0, $s1, -4       #Inst 01
        beq $v0, $zero, endif   #Inst 02
        sw $v1, 4($a0)          #Inst 03
        # There are 9 more instructions here
endif:  subu $t1, $t2, $t3      #Inst 13
```

01:	
02:	
03:	
13:	0x014B4823

b) Given what you know about the Game of Life 1D from Homework 2, how many *possible rules would exist* if at each generation we looked at *six* cells from the previous generation? Write your answer in IEC format (e.g., 128 Kibirules, 2 Gibirules, etc…).

c) We're designing a quad-length floating point number (16 bytes wide) with 64 yobi different *significand* bit patterns. How many *different exponent bit patterns* would we have? Don't worry about the *numbers we represent* with these bit patterns. (Hint: this isn't really a question about floating point, it's a counting question. If you're thinking about bias, NaNs, denorms, etc. you're heading down the wrong path.)

Name: ____________________________ Login: `cs61c-`____

Question 4: Float, float on… (10 pts, 25 min)

The staff of CS61C is constantly investigating new approaches and solutions to old problems. (Either that or we just don't know when to leave well enough alone!) Well, we've come up with a new floating point format that **obeys the rules of the IEEE 754 Floating Point Standard** (denorms, NaNs, ±∞) but is *only 13 bits long*. It has 1 sign bit, 3 exponent bits, and 9 bits in the fraction (significand):

S	E	E	E	F	F	F	F	F	F	F	F	F

Answer the following questions about this new float format. As a sanity-check, if you calculate the bias of this format, you should get 3.

a) What is the largest non-infinite positive number that can be represented? Leave your answer as a base 10 mixed fraction (i.e. K ± (N/D)) where either K or N can be 0.

b) What is the smallest positive number that can be stored in this format?

Convert the following floating point values in the format above (expressed in hexadecimal) to their numerical (base 10) equivalents, if appropriate.

c) `0x1000`

d) `0x1F00`

What is the representation for the following floating point numbers? (Write your answer in hex).

e) `-∞`

f) `-4.5`

Name: ___________________________ Login: cs61c-____

Question 5: Are you the keymaster? (10 pts, 25 min)

We're authoring code to build a table to store *keys* and their *values* (both numbers). We're using a two-tiered linked list whose main spine is a linked list of structures of type `keynode_t` (defined below), which themselves contain linked lists of structures of type `valuenode_t` (also defined below).

The `keynode_t` structure is:

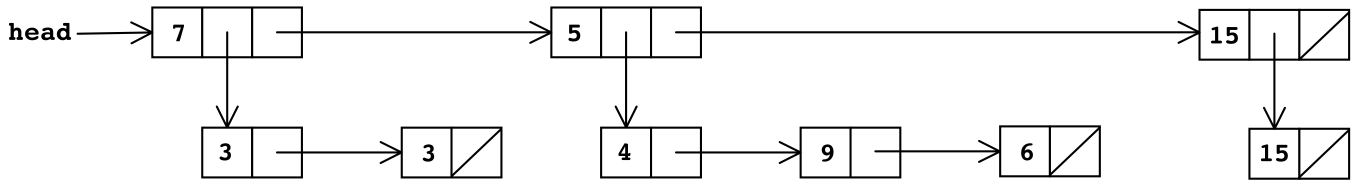
```
typedef struct keynode {
   int key;
   valuenode_t *values;
   struct keynode *next;
} keynode_t;
```

The `valuenode_t` structure is:

```
typedef struct valuenode {
   int value;
   struct valuenode *next;
} valuenode_t;
```

```
keynode_t *head = NULL;
head = Put(head,7,3);
head = Put(head,5,6);
head = Put(head,7,3);
head = Put(head,15,15);
head = Put(head,5,9);
head = Put(head,5,4)
```

When we add a (*key*,*value*) pair to our structure using the `Put` command we find the `keynode` with the same *key* (making a new one if one doesn't exist) and add the *value* to the list of values (even if there's a duplicate). So, the sequence of calls to `Put` from the box above would result in the table:



We want to be able to delete the **full structure**. Assume that the OS immediately fills any freed space with garbage, so you cannot access freed heap contents. Finish the function `DeleteTable` which must delete the `keynode_t`s *recursively*. Your code must fit in `DeleteTable()` and you may not use any additional functions. We want the tightest, cleanest code possible (measured by the number of statements which terminate in semicolons). Here's how we would call `DeleteTable`:

```
int main()
{
  keynode_t *head = NULL;
  head = FillTable();      /* Somehow fill the table via user input */
  PrintTable(head);        /* Somehow print the table */
  DeleteTable(head);       /* Delete the table (code below) */
  DoSomething();           /* Do something, but we need the space back! */
}
```

```
void DeleteTable (keynode_t *table)
{
  /* You must delete the keynode_ts recursively */

}
```

Name: ____________________________ Login: cs61c-____

Question 6: We're deep, deep undercover… (10 pts, 25 min)

You've been contracted by top secret government agencies to make a really quick, super-portable message encoder in MIPS! Go spy go! You have to implement an algorithm for encoding an ASCII zero-terminated (C-Style) string. Your algorithm should be based on the following C code:

```
void encryptThis(char* cleartext, int* cypher, int* cyphertext_buffer){
  if(*cleartext == '\0'){
      *cyphertext_buffer = 0;
  }else{
      *cyphertext_buffer = *cypher + *cleartext;
      encryptThis(++cleartext, ++cypher, ++cyphertext_buffer);
  }
}
```

Implement the above C function **in a non-recursive manner** in MIPS. Don't clobber any registers that you shouldn't, **`$a0`** corresponds to **`cleartext`**, **`$a1`** corresponds to **`cypher`**, and **`$a2`** corresponds to **`cyphertext_buffer`**. You can assume that you will only be passed the usual types of ASCII values (in the range 0-127). Use as few lines as possible (you may not need to use every blank).

`encryptThis:`

Name: ____________________________ Login: cs61c-____

Question 7: Meet my friend Andy Anderson… (10 pts, 25 min)

```
Main: ......
      # Set up $a0
      jal foo
      ......
foo:  li $v0, -1
      lbu $t0, 0($a0)
      beq $0, $t0, done
      addi $sp, $sp, -8
      sw $ra, 4($sp)
      sw $t0, 0($sp)
      addi $a0, $a0, 1
      jal foo
      lw $t0, 0($sp)
      and $v0, $v0, $t0
      lw $ra, 4($sp)
      addi $sp, $sp, 8
done: jr $ra
```

```
                foo(                         )
{



   return                    ;
}
```

a) What does the function foo return?

b) In the box above, fill in the C code for the function foo. Be sure to include arguments and return values, along with their types.

c) If we call your function foo like this: **printf("%c", foo("Cal"));** What will be printed?

d) What would foo do if we changed its first line to read "**li $v0, 0**"?