University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2007 Instructor: Dr. Dan Garcia 2007-03-05

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 **SECTION** TA (please circle)*	Aaron Alex Brian David Matt Michael Valerie
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 7 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.

Question	0	1	2	3	4	5	Total
Minutes	1	36	36	36	36	36	180
Points	**1**	**15**	**14**	**15**	**15**	**15**	**75**
Score	**1**	**15**	**14**	**15**	**15**	**15**	**75**

Name: _____Answers_____ Login: cs61c-____

Question 1: Is this the best midterm in memory? No, we `free`d it! (15 pts, 36 min)

a) I have **N** bits to represent data, and every bit pattern has a unique meaning. I want to represent **3 times** as many things. How many more bits do I need?

2

b) We are given two nibbles, *A* (=`0xF`) and *B* (=`0b0010`), and we wish to calculate their *SUM* = *A* + *B*. *We only have a nibble* to store the SUM result. What is *SUM* if *all three nibbles (A, B, SUM)* were…

Algorithm: SUM = encode(decode-into-decimal(A) + decode-into-decimal(B))

	SUM (single hex character)	Decimal number SUM encodes	Was there overflow?
…sign magnitude?	0xD	-5	no
…ones complement?	0x2	2	no
…unsigned?	0x1	1	yes
…twos complement?	0x1	1	no
...encoded with a bias of 7 (like the way the exponent is encoded w/`float`)	0xA	3	No

S/M:-7+2=-5==>1101, 1s:-0+2=2==>0010, us:15+2=17==>10001, 2s:-1+2=1==>0001, b:8+-5=3==>1010

c) Put the following in chronological order. We've started it for you.

6	Code and Data from various places are stitched together.
1	A CS61C student is assigned a project that implements `big_nums`.
9	Execution begins at `main`.
2	The student writes his or her code in C.
5	Link tables are produced.
4	MAL is translated into TAL.
8	Static, code, and global space are reserved/initialized in memory.
3	The student's C code is translated into MIPS.
7	Links are "edited"

d) Assume we have just enough bits to byte-address 512_{10} zebibytes. We want to define some number of the most-significant bits to encode $9_{10} \times 2^{50}$ things, and some number of the least-significant bits to encode $2{,}000_{10}$ things. How many things can we encode with the remaining bits? Use IEC language, like "16 mebithings". Show your work.

79-54-11=14 bits

16 kibithings

e) For *every* line of code on the right, we want to know if any memory is used, and if so, where and how much. If zero, leave it blank.

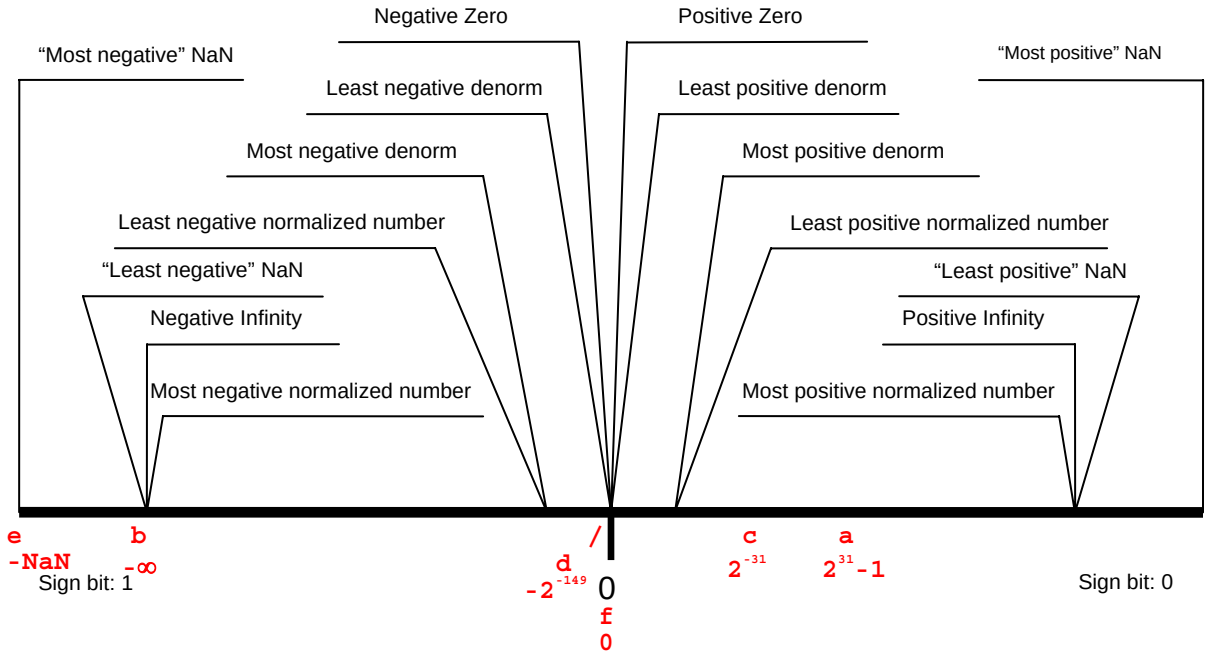
Static	Stack	Heap
4		
	108	
		5

```
1   typedef struct bignum {
2     int len;
3     char *num;
4     char description[100];
5   } bignum_t;
6   bignum_t *res;
7
8   int main() {
9     bignum_t b;
10    b.num = (char *) malloc (5 * sizeof(char));
      // more code below
```

Name: _____Answers_____ Login: cs61c-____

Question 2: If swimming in the 61→6+1→7 Cs, keep "a `float`" (14 pts, 36 min)

The figure below shows the layout of the different types of 32-bit `float` numbers on a not-to-scale real number line (with zero in the center, and NaNs considered to be further from zero than infinity).



For each of the (a-f) values below, draw marks below the number line (like we've done for 0) indicating where the value would fall. **Label the mark with the corresponding letter and actual value below it**.

a) The *sign magnitude* number closest to ∞.

b) The result of casting the `double` number closest to -∞ (but is *not* -∞) into a `float`.

c) 1/(X+1), where X is the largest *ones complement* number.

d) The number represented by the `float` bits `0x80000001`.

e) The number represented by the `float` bits `0x00000000` times the number represented by the `float` bits `0xFFFFFFFF`
(using normal floating point multiply `mul.s`)

f) The difference between the `int` value closest to -∞ and the `float` that can most closely represent that `int`'s value.

g) The default `float` rounding mode often needs to break ties for numbers that fall *between* `float`s it can represent. What is the largest `unsigned int` that falls *exactly* between two `float`s, and what does it round to? (e.g., If this were in decimal, you might write "3.5 → 4"). Show your work below, and put your answer in the box. You may leave your result as a (simplified) expression.

$2^{32} - 2^{7} \rightarrow 2^{32}$

The largest unsigned int is $2^{32} - 1$, which falls between floats 2^{32} and $2^{32} - 2^{8}$. Thus, halfway between those two floats is $2^{32} - 2^{7}$, whose bits look like [0 | bits-encoding-exponent-32 | 11...11] 1, and rounding to 'even' means rounding away from odd, which is rounding up.

Name: _____Answers_____ Login: cs61c-____

Question 3: Goodness, Grandma, what `bignums` you have! (15 pts, 36 min)

Part A: After your extensive C `bignum` experience, you were hired by Lawrence Berkeley Labs to make an arbitrary precision math package. The scientists use scientific notation and keep track of significant figures. The scientists have written a function called `sci_bignum_cmp` as shown below. Unfortunately, the implementation has at least one bug. In the boxes at the bottom, briefly explain all of them and give sample values for `a` and `b` (i.e. 1.23 x 10^4) that causes `sci_bignum_cmp()` to reveal the bug. You may not necessarily use all the boxes.

As an example, to store the number 1.23 x 10^4, the `sign` would be the char `'+'`, the `significand` would be the null-terminated string `"123"`, `num_sigfigs` would be `3`, and the `exponent` would be `4`. There is an implicit decimal point after the first `significand` digit.

```
#define POS '+'
#define NEG '-'
typedef struct sci_bignum {
   char sign;                  // POS or NEG
   char *significand;          // null-terminated string of decimal digits ('.' implicit)
   unsigned int num_sigfigs;   // equal to strlen(significand)
   int  exponent;
} sci_bignum_t;

// Compare a to b; return <0 if a < b, 0 if a == b, or >0 if a > b (just like strcmp)
int sci_bignum_cmp(sci_bignum_t *a, sci_bignum_t *b) {
    if (a->exponent != b->exponent) {
        return (a->exponent < b->exponent ? -1 : 1);
    } else if (a->sign != b->sign) {
        return (a->sign < b->sign ? -1 : 1);
    } else {
        for(int i = 0; a->significand[i]; i++) {
            if(a->significand[i] != b->significand[i])
                return (a->significand[i] < b->significand[i] ? -1 : 1);
        }
    }
}
```

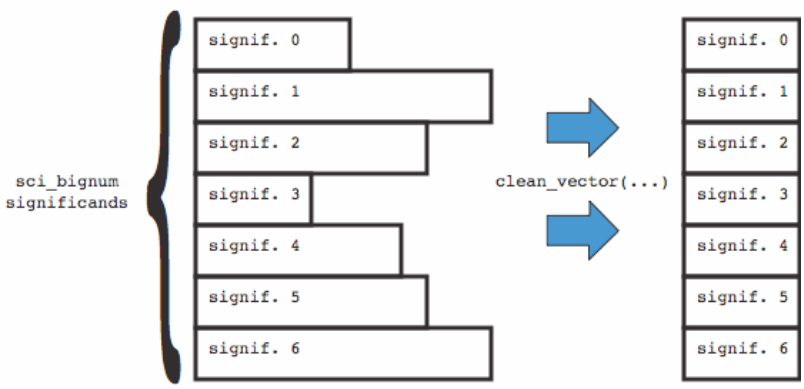
Bug Description	Values for `a` and `b` that reveal the bug	What *a correct* `sci_bignum_cmp` should return	What *this* buggy `sci_bignum_cmp` returns / does
exponent compared before sign	a = -1.23 x 10^4 b = 1.23 x 10^{-4}	-1	1
Wrong return value when exponents are equal, but a->sign != b->sign since char '+' < '-'	a = -1.23 x 10^4 b = +1.23 x 10^4	-1	1
Wrong return value when two negative numbers passed in	a = -1.2 x 10^4 b = -1.4 x 10^4	1	-1
No return value when a == b (should be 0)	a = 1.23 x 10^4 b = 1.23 x 10^4	0	$v0 not set so returns garbage. Might not compile

Name: _____Answers_____ Login: cs61c-____

Question 3: Goodness, Grandma, what `bignums` you have! (cont'd)

Part B: The scientists also use vectors quite a bit, so they wrote the following C `struct`:

```
typedef struct sci_vector {
      sci_bignum_t *elts;
      unsigned int num_elts;
} sci_vector_t;
```



The scientists want the elements of a given vector to have the *same number of significant digits*. In other words, each element in a vector should be truncated to the smallest `num_sigfigs` in the vector. They want you to help them write a function (by filling in the blanks) that will "clean up" its argument vector by modifying all its `sci_bignum`s to have the appropriate number of significant digits. Any excess allocated space should be freed so there's no wasted memory. Avoid memory leaks.

```
#define MIN(a, b) ((a)<(b)?(a):(b))

void clean_vector(sci_vector_t *vec) {
    unsigned int min_sigfigs = 0xFFFFFFFF; // Initialize to biggest unsigned int

    /* get min significant digits */
    for (unsigned int i = 0; i < vec->num_elts; i++)
                                  min_sigfigs,          vec->elts[i].num_sigfigs
        min_sigfigs = MIN( _____________________________ , _____________________________);

    /* truncate all elts to have min_sigfigs */
    for (unsigned int i = 0; i < vec->num_elts ; i++) {
                                vec->elts + i;
        sci_bignum_t *b = ____________________________ // convenient reference
                  b->num_sigfigs                  min_sigfigs
        if (_____________________________ > _____________________________) {
                                (char *) malloc((min_sigfigs + 1)*sizeof(char))
            char *new_significand = (______) malloc(_____________________________);
            b->significand[min_sigfigs] = '\0';
            ____________________________________________________________ // for strcpy

            strcpy(new_significand, b->significand);
            free(b->significand);
            ____________________________________________________________________________
            b->significand = new_significand;
            ____________________________________________________________________________
            b->num_sigfigs = min_sigfigs;
            ____________________________________________________________________________

            ____________________________________________________________________________

            ____________________________________________________________________________

        }
    }
}
```

Name: _____Answers_____ Login: cs61c-____

Question 4: `fun` with MIPS ... more naughty bits! (15 pts, 36 min)

What follows is an inefficient MIPS function. Read it carefully, and answer the questions below. The definition of `div` can be found in your green sheet, column ↑ (`div a,b` ∏ `lo=a/b, hi=a%b`).

```
fun:  mov   $v0, $0
      li    $s0, 1
loop: beq   $a1, $0, end
      addiu $a1, $a1, -1
      sll   $s0, $s0, 1
      div   $a0, $s0
      mfhi  $s1
      or    $v0, $v0, $s1
      j     loop
end:  jr    $ra
```

```
// Precondition: y < 31

unsigned int fun(unsigned int x,
                 unsigned int y)
{
   return x & ((1 << y) - 1);




}
```

a) Briefly, explain what `fun` returns (assuming `y < 31`).
***Don't** describe the algorithm; explain how the return value relates to `x` and `y`.*

> The lowest y bits of x

b) Write optimized C code for `fun` in the box (make it as compact and efficient as possible). That is, think of all the C tricks you know and try to author it in the fewest characters possible.

c) Uh oh, we've broken some calling conventions! What should we add to the beginning (before `mov $v0, $0`) and end (before `jr $ra`) of `fun` to correct this? Help!

BEGIN	END
addiu $sp $sp -8 sw $s0 0($sp) sw $s1 4($sp)	lw $s0 0($sp) lw $s1 4($sp) addiu $sp $sp 8

Name: _____Answers_____ Login: cs61c-____

Question 5: He's a unix. He's definitely a unix. He's dead! (15 pts, 36 min)

Assume (for simplification) that `main` returns a value to its caller (Unix) through standard MIPS procedure calling conventions. We wish to see how long our command-line inputs arguments are:

```
unix% count_argument_characters
0

unix% count_argument_characters I love cs61c!
11
```

Implement `count_argument_characters` in MAL MIPS. Follow the hints given by the comments; you may not need to use all the lines.

```
            li $v0, 0
main: _____________________     # ans=0
           $a0, $a0
word: addiu _____________, -1   # Decrement
           $a0, $0
      beq _____________, done   # We're done!
       sll         $a0   $2
      ______ $t0, ____, ____    # change $t0
       addiu $t1, $a1, $t0
      _____________________     #
           lw $t1, 0($t0)
      _____________________     #
           lb $t2, 0($t1)
letr: _____________________     #
                $0
      beq $t2, _______, word    # end of word
         addiu $v0, $v0, 1
      _____________________     #

      addiu $t1, $t1, 1         # increment $t1

      _____________________     #
             letr
      j ____________________    # keep processing

done: jr $ra
```