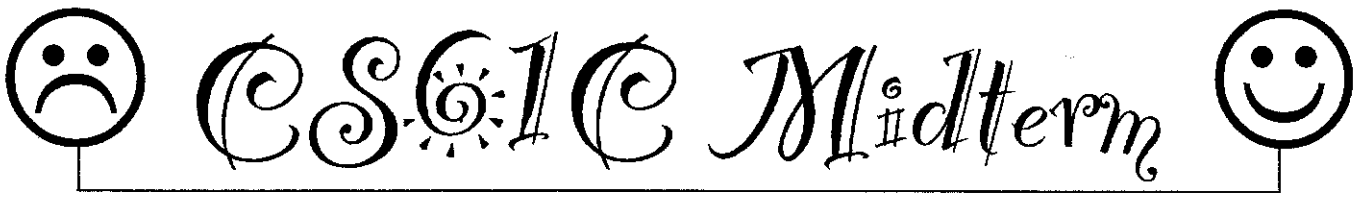
University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2008 Instructor: Dr. Dan Garcia 2008-03-09



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)*	Ben Brian Casey David Keaton Matt Omar
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)***	

a) 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.
- 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**	**14**	**15**	**15**	**15**	**15**	**75**
Score	**1**	**14**	**15**	**15**	**15**	**15**	**75**

Name: Answers Login: cs61c-____

Question 1: Potpourri: hard to spell, nice to smell... (14 pts, 36 min)

Questions (a) and (b) refer to the C code to the right; pretend you don't know about MIPS yet.

```
#define val 16
char arr[] = "foo";
void foo(int arg){
   char *str = (char *) malloc (val);
   char *ptr = arr;
}
```

a) In which memory sections (code, static, heap, stack) do the following reside?

arg stack arr static

*str heap val code

b) Name a C operation that would treat `arr` and `ptr` differently: sizeof sizeof(arr) != sizeof(ptr) ++ (arr++ crashes, ptr++ does not)

You peek into the *text* part of an `a.out` file and see that the left *six* bits of an instruction are `0x02`. As a result of executing this instruction...

c) What's the *most* that your PC could change? Be exact. opcode=0x02 → jump 2^28 – 4

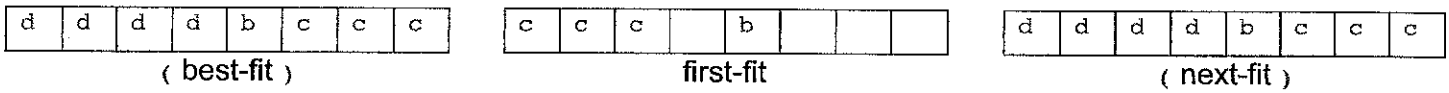
d) What is the *least*? 0

e) Write a `getPC` function, which returns the address of the `jal` instruction **calling it**. (two instructions should be sufficient)

```
getPC: addiu $v0, $ra, -4
       jr $ra
```

f) Which of the *best-*, *first-*, *next-*fit schemes would succeed for **all 5** of the following sequence of `malloc` and `free` requests on a `malloc`-able region of memory only 8 bytes long? Circle those that would and show the resulting contents of memory for each one. E.g., After the "`a=malloc(4)`" call, all schemes should have the *leftmost* 4 boxes labeled "a". A pencil is useful (or draw "a" lightly).

```
a = malloc(4); b = malloc(1); free(a); c = malloc(3); d = malloc(4);
```



(best-fit) first-fit (next-fit)

g) In one sentence, why can't we use automatic memory management in C?

C is weakly typed; any variable could be a pointer.

h) To reduce complexity for your software company, you delete the *Compiler*, *Assembler* and *Linker* and replace them with a single program, `CAL`, that takes all the source code in a project and does the job of all three for *all* the files given to it. Overall, is this a good idea or bad idea? Why or why not?

BAD idea! A change to only one file requires recompiling/reassembling all!

Name: Answers Login: cs61c-____

Question 2: Player's got a brand new bag... (15 pts, 36 min)

We want to add an inventory system to the adventure game so that the player can collect items. First, we'll implement a *bag* data structure that holds *items* in a linked list. Each `item_t` has an associated `weight`, and each `bag_t` has a `max_weight` that determines its holding capacity (see the definitions below). In the left text area for `item_node_t`, define the necessary data type to serve as the nodes in a **linked list** of items, and in the right text area, add any necessary fields to the `bag_t` definition.

```
typedef struct item {
    int weight;
    // other fields not shown
} item_t;
```

```
typedef struct item_node {
    // (a) FILL IN HERE

    item_t *item;
    struct item_node *next;

} item_node_t;
```

```
typedef struct bag {
    int max_weight;
    int current_weight;
    // add other fields necessary
    // (b) FILL IN HERE

    item_node_t *contents;

} bag_t;
```

c) Complete the `add_item()` function, which should add `item` into `bag` **only** if adding the item would not cause the weight of the bag contents to exceed the bag's `max_weight`. The function should return `0` if the item *could not* be added, or `1` if it succeeded. Be sure to update the bag's `current_weight`. You do not need to check if `malloc()` returns `NULL`. Insert the new item into the list wherever you wish.

```
int add_item(item_t *item, bag_t *bag) {
    if ( item->weight + bag->current_weight > bag->max_weight ) {
        return 0;
    }
    item_node_t *new_node = (item_node_t *) malloc( sizeof(item_node_t) );

    // Add more code below…

    new_node->item = item;
    new_node->next = bag->contents;
    bag->contents = new_node;
    bag->current_weight += item->weight;

    return 1;
}
```

(d) Finally, we want an `empty_bag()` function that frees the bag's linked list but **NOT** the memory of the items themselves and **NOT** the bag itself. The bag should then be "reset", ready for `add_item`. Assume that the operating system immediately fills any freed memory with garbage. Fill in the functions below.

```
void empty_bag(bag_t *bag) {
    free_contents( bag->contents );

    // FILL IN HERE
    bag->current_weight = 0;
    bag->contents = NULL;

}
```

```
void free_contents( item_node_t *c ) {

    // FILL IN HERE

    if (c == NULL) return;
    free_contents(c->next);
    free(c);

}
```

Name: Answers Login: cs61c-____

Question 3: You won't mind this question one bit! (15 pts, 36 min)

We wish to implement a **bit** array, where we can read and write a particular *bit*. Normally for read/write array access, we would just use bracket notation (e.g., `x=A[5]; A[5]=y;`), but since a bit is *smaller* than the smallest datatype in C, we have to design our own `GetBit()` and `SetBit()` functions. We'll use the following typedefs to make our job easier:

```
typedef uint8_t  bit_t;      // If it's a single bit, value is in least significant bit.
typedef uint32_t index_t;    // The index into a bit_t array to select which bit is used
```

E.g., imagine a 16-bit bit array: `bit_t A[2]; A[1]=0x82; A[0]=0x1F;` Internally, `A` would look like this:

	8				2				1				F			
Array A:	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1
Bit index:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

`GetBit(A,0)` would return `1`, as would `GetBit(A,1)`, `GetBit(A,2)`, `GetBit(A,3)`, and `GetBit(A,4)`. `GetBit(A,5)` would return `0`, as would `GetBit(A,6)`, `GetBit(A,7)`, and `GetBit(A,8)`. Etc.

a) How much space would the largest **usable** bit array take up? "Usable" means we could read and write every bit in the array. Express your answer in IEC format. E.g., 128 KiB, 32TiB, etc. **512 MiB**

b) Write `SetBit` in C. *You may not need to use all the lines.*

```
void SetBit(bit_t A[], index_t n, bit_t b) {   // b is either 0 or 1
    A[n/8] = ( A[n/8] &  ~(1 << (n%8)) ) | (b << (n%8)); // The one liner. Or …

    uint32_t byte_index = n/8;    // (1) Find out which byte we want
    uint8_t  bit_index  = n%8;    // (2) Find where within that byte is the bit

    A[byte_index] &= ~(1 << bit_index); // (3) Reset that bit in the byte

    A[byte_index] |= b << bit_index;    // (4) Assign that bit in the byte
}
```

c) Write `GetBit(bit_t A[], index_t n)` in MAL; `$v0` should be `1` if the bit is on, and `0` if it's off. Hint: it might help if you start from the `srlv` and work backwards.

```
        srl   $t0, $a1, 3            # $t0 = byte_index = n/8
GetBit: _____ $t0,_________________  #
        addu  $t1, $a0, $t0          # $t1 = A + byte_index
        _____ $t1,_________________  #
        lbu   $t2, 0($t1)            # $t2 = byte = *(A+byte_index) = A[byte_index]
        _____ $t2,_________________  #
        andi  $t3, $a1, 7            # $t3 = bit_index = n%8
        _____ $t3,_________________  #
        srlv  $v0,$t2,$t3            # $v0 = byte >> bit_index (slide bit to lsb slot)
        srlv  $v0,$t2,$t3            # "srlv rd,rt,rs" means (in C): rd = rt >> rs
        andi  $v0,$v0,1              # $v0 &= 1 (mask out the lsb bit)
        _____ _________________      #

        jr    $ra                    # $v0 better be either a 0 or 1
```

Name: Answers Login: cs61c-____

Question 4: Did somebody say "Free Lunch"?! (15 pts, 36 min)

Consider two competing 5-bit floating point formats. Each contains the same fields (sign, exponent, significand) and follows the same general rules as the 32-bit IEEE standard (denorms, biased exponent, non-numeric values, etc.), but allocates its bits differently.

Implementation "LEFT": | S | EE | FF |

scratch space (show all work here)

Exponent Bias: $2^{\#Es-1}-1 = 2^{2-1}-1 = 1$

Denorm implicit exponent: $-(Bias-1) = 0$

Number of NANs: 6

What	Number	Bit Pattern
Smallest non-zero pos denorm	2^0x0.01= ¼	0b00001=0x1
Largest non-infinite pos value	2^1x1.11=3.5	0b01011=0xB
Negative Infinity	−∞	0b11100=0x1C

Mark every representable number in the range [+0,1] as a vertical line on the number line below. We've already done it for +0.

+0 1/8 ¼ ½ ¾ 1

0 ¼ ½ ¾ 1

Implementation "RIGHT": | S | EEE | F |

scratch space (show all work here)

Exponent Bias: $2^{\#Es-1}-1 = 2^{3-1}-1 = 3$

Denorm implicit exponent: $-(Bias-1) = -2$

Number of NANs: 2

What	Number	Bit Pattern
Smallest non-zero pos denorm	2^{-2}x0.1=1/8	0b00001=0x1
Largest non-infinite pos value	2^3x1.1=12	0b01101=0xD
Negative Infinity	−∞	0b11110=0x1E

Mark every representable number in the range [+0,1] as a vertical line on the number line below. We've already done it for +0.

+0 1/8 ¼ ½ ¾ 1

0 1/8 ¼ 3/8 ½ ¾ 1

Which implementation is able to represent more *integers*, LEFT or (RIGHT) ? (circle one)

Name: Answers Login: cs61c-____

Question 5: Three's a Crowd... (15 pts, 36 min)

Breaking news! We have just developed hardware that has 3-states: {false=0, true=1, and maybe=2}! Now we can store all our numbers in base 3. The race is on to develop a good encoding scheme for integer values.

Decimal	Ternary
5	12_{three}
26	222_{three}
27	1000_{three}

a) To warm up, first do some simple conversions between decimal and unsigned ternary. We've done one for you.

b) Suppose we have N ternary digits (*tets*, for short). What is the largest unsigned integer that can be stored?

$3^N - 1$

Ok, now that we've got unsigned numbers nailed down, let's tackle the negatives. We'll look to binary representations for inspiration.

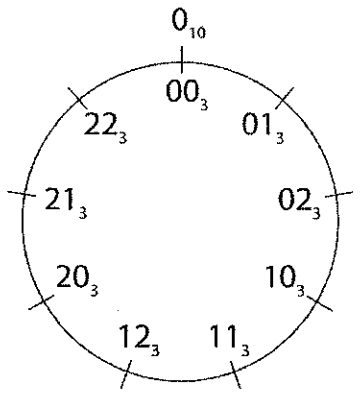
c) Name two disadvantages of a *sign and magnitude* approach in ternary. Suppose a leading 0 means positive, and a leading 1 means negative, similar to what we did in the binary days.

There are only two signs, but three digits (we waste 1/3 of our numbers)

Wierd gap behavior -> now we jump to 2's mystery zone from 11…11 + 1.

d) Maybe *three's complement* will be more promising. To make sure we understand what that means, let's begin with a very small example – say a 2-tet number. Fill in the following number ring of tet-patterns with the values we'd like them to represent (just as in two's complement, we want all zeros to be zero, and want a balanced number of positive and negative values).

Going clockwise from 00_{three} : 0, 1, 2, 3, 4, -4, -3, -2, -1



e) Recall that for an N-bit *two's* complement number, the bit-pattern of the largest positive number looks like 011…11. For an N-tet *three's* complement number, what does the tet-pattern of the largest positive number look like?

11…11

f) Provide (in pseudocode) an algorithm for *negating* an N-tet three's complement number.

```
Flip all tets across 1.  (i.e., 0 -> 2, 1-> 1, 2 -> 0)
Add 1. (same as 2's complement!)
```