University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Summer 2010 Instructor: Paul Pearce 2010-07-16

☹ 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)*	Eric Tom Noah Alex
Name of the person to your Left	
Name of the person to your Right	
All the work is my own and I have collaborated with no one. 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 **12** numbered pages including the cover page and MIPS reference guide. 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. 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 login on every sheet of paper.
- You have 180 minutes to complete this exam. The exam is closed book, no computers, PDAs, calculators. You are allowed 1 page of notes, front and back.
- A MIPS reference sheet has been provided as the last page of this handout. You should rip it off.
- 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	15	20	25	15	20	25	(+59 to review) = 180
Points	**1**	**39**	**20**	**50**	**20**	**25**	**35**	**190**
Score	**1**	**39**	**20**	**50**	**20**	**25**	**35**	**190**

Login: cs61c-__**Answers**__

Question 1: Where's the kitchen sink? ………………………………. (39 pts, 15 min)

Part 1: CS61C trivia

True/False: Circle the correct answer in the right-hand column.		
a) You must use the *addu* instruction to add unsigned numbers.	T	**F**
b) You do not need to save volatile registers if your code doesn't call any subfunctions.	**T**	F
c) You do not need to save volatile registers if they won't be modified by any subfunctions.	T	**F**
d) We add a bias to floating point exponents to increase the range of values we can represent.	T	**F**
e) The instructions *srl* and *sra* behave identically on positive (2's complement) numbers.	**T**	F
f) There are situations where using first-fit will cause less fragmentation than best-fit.	**T**	F
g) The size of a structure that contains only 2 `int`s and 1 `char` will be 9 bytes.	T	**F**

Fill in the blank: neatly write your answer in the right-hand column

h) How many things can you represent with *N* bits?	2^N
i) Suppose you are given N bits. How many more bits would you need if we wished to triple the number of things we wanted to represent?	**2**
j) Assuming the following C code declaration: `char str[] = "Hello World";` What will `sizeof(str)` return?	**12**
k) Assuming the following C code declaration: `char *str = "Hello World";` What will `sizeof(str)` return?	**4**
l) Assuming the following C code declaration: `char str[] = "Hello\0World";` What will `strlen(str)` return?	**5**

(Continued on next page)

Question 1: Where's the kitchen sink? (Continued) ………………… (39 pts, 15 min)

Part 2: Number representation

So far we have studied 4 different methods for representing integers using 32-bits. These methods can be generalized to any number of bits.

Fill in the bit patterns for the following *4-bit* numbers. If there are multiple bit patterns for a given number, write them all. If no bit pattern exists to represent the given number, write "N/A" in the box ***(don't leave it blank!).*** The first one has been done for you already.

	Unsigned	Sign & Magnitude	One's Complement	Two's Complement
0	0000	0000, 1000	0000, 1111	0000
-1	N/A	1001	1110	1111
15	1111	N/A	N/A	N/A

Now fill in the **decimal** (base 10) value for the following *4-bit* numbers. The first one has been done for you already.

	Unsigned	Sign & Magnitude	One's Complement	Two's Complement
Number with bit pattern 0b1100	12	-4	-3	-4
Number closest to +∞	15	7	7	7
Number closest to -∞	0	-7	-7	-8

Part 3: Compiling/Linking/Loading

Fill in the blanks to specify during what stage each action occurs. Use abbreviations **CO**=Compiling, **AS**=Assembly, **LI**=Linking, **LO**=Loading.

__LI__ Jump labels are resolved

__AS__ Short branch labels are resolved

__LO__ The operating system handles this stage

__CO__ Code is translated from C->MAL

__AS__ Code is translated from MAL->TAL

Login: cs61c-__**Answers**__

Question 2: Did somebody say "Free Lunch"?! .... (20 pts, 20 min)

Consider the following 10-bit floating-point format. It 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.). It simply allocates its bits differently. Please answer the following questions, and show all your work in the space provided. We went ahead and got you started.

S	EEE	MMMMMM

Number represented by **0x00**: ______0______

Bits in the Mantissa: ______6______

a) Exponent Bias: ______3______

b) Implicit exponent for denormalized #'s: ______-2______

c) # of Numbers between (2 ≤ n < 8): ______128______

d) Largest number x such that x + .5 = .5: ______2^{-8} = 1/256______

(Continued on next page)

Question 2: Did somebody say "Free Lunch"?! (Continued) .... (20 pts, 20 min)

S	EEE	MMMMMM

(Repeated so you don't need to flip back and forth)

e) Difference between the two smallest positive values: ______ 2^{-8} = 1/256 ______

f) Difference between the two largest non-   positive values: ______ 2^{-3} = 1/8 ______

g) Number of NaN's: ______ 126 ______

h) Using the above format, what is the bit pattern for the floating-point number closest to 14.4? _____0b0110110011______

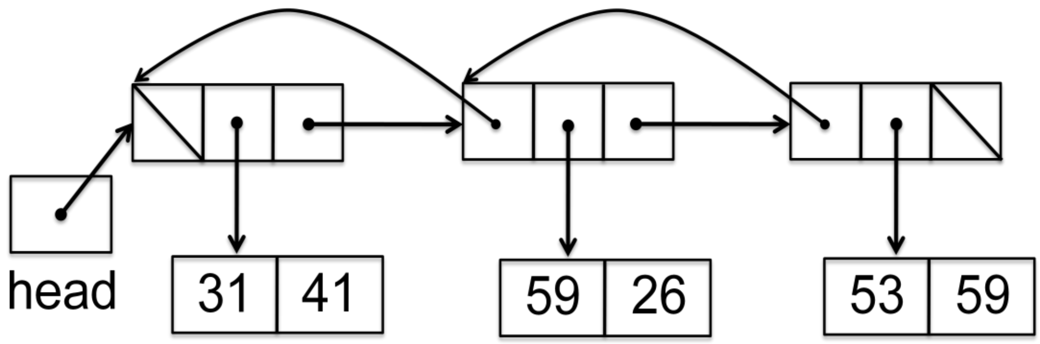
Login: cs61c-__Answers__

Question 3: Don't lose your `head` ....(50 pts, 25 min)

We've created a simple doubly linked list out of nodes as defined below. Each node contains a pointer to a struct pair. The structures are defined as follows.

```
struct pair {
  int x;
  int y;
};

struct node {
  struct node *prev;
  struct pair *datum;
  struct node *next;
};
```



The first node has its prev field set to NULL. The last node has its next field set to NULL.

a) Fill in the code below to implement `insert_before`. The function inserts a node in the linked list before `given_node` and sets up the node's `datum`. Should `new_node` be the new head of the linked list, adjust `head` accordingly (a handle to the head is passed as `head_h`). The function returns a pointer to the new node, or `NULL` if the operation cannot be completed. Assume **valid non-NULL** parameters.

```
struct node *insert_before(struct node *given_node, struct pair p, struct node **head_h) {
    // Allocate space
    struct node *new_node = (struct node*) malloc (sizeof(struct node));

    if (new_node == NULL)   // Check for errors

          return NULL;

    // Setup datum

    new_node->datum = (struct pair*) malloc (sizeof(struct pair));
    if(new_node->datum == NULL)
    {
        free(new_node)
        return NULL;
    }
    *(new_node->datum) = pair;

    new_node->next = given_node->next;

    new_node->prev = given_node->prev;

    if (given_node == *head_h)

          *head_h = new_node;

    else
          given_node->prev->next = new_node;

    given_node->next->prev = new_node;

    return new_node;
```

}

Question 3: Don't lose your `head` (Continued)………………………. (50 pts, 25 min)

b) The following `free_list` function takes a pointer to `head` (recall head points to the first element in the doubly linked list, and `head_h` is a handle to the head) and frees all memory that was allocated for the list. Once the list is freed, `free_list` must set `head` to `NULL`. This function is **BUGGY.** Assume the memory for both the nodes and the datums was allocated from the heap. You should also assume `free_list` is correctly passed the address of `head` and all datums are non-`NULL`.

```
void free_list(struct node **head_h) {

    struct node *curr = head_h;

    while (curr != NULL) {
        free(curr);
        curr = curr->next;

    }
}
```

Describe all the bugs in the given `free_list` function in the space below. Number each bug.

1) ***head_h**
2) **Free then deref**
3) **No datum free**
4) **No head update**

c) Implement a correct iterative version of `free_list` which corrects the bugs found in part b. Please re-read the function requirements from part b. You may not need all the space provided.

```
void free_list(struct node **head_h) {

    struct node *curr = *head_h;


    while (curr != NULL) {

      free(curr->datum);
      struct node *tmp = curr->next;
      free(curr);
      curr = tmp;


    }

    *head_h = NULL;


}
```

Login: cs61c-__**Answers**__

Question 4: Who needs a compiler? ……………………………………(20 pts, 15 min)

While trying to compile a program for a MIPS processor, the compiler crashes. Before crashing, it compiled everything but the following lines of code.

```
// var and i are signed integers
// ptr is a pointer to a sufficiently large array of integers
i = 0;
while (var != 0) {
      var = var * 2;
      i++;
}

if (i >= 0) {
      var = *ptr;
}
else {
      var = *(ptr+i);
}
```

Finish the job of the **compiler** by translating the preceding C to MIPS code.

`var` is stored in `$t0`, `i` is stored in `$t1`, and the pointer `ptr` is in `$t2`. Ignore register conventions for this problem. We have given you the first instruction and some labels to help guide you. **You must comment your code!** You may not need all the given space. **Do not use mult!**

```
             addu $t1, $0, $0          # Set i = 0

while:       beq $t0, $0, endWhile     # While var != 0
             sll $t0, $t0, 1           # Var *= 2
             addiu $t1, $t1, 1         # i++
             j while                   # loop back and retest

endWhile:    slt $t3, $t1, $0          # t3 = 1 if t1 < 0, 0 otherwise
             bne $t3, $0, else         # if t1 is not >= 0, goto else

if:          lw $t0, 0($t2)            # var = *ptr;
             j done                    # DO NOT execute the else case

else:        sll $t3, $t1, 2           # Get byte offset
             addu $t4, $t2, $t1        # Add in the offset
             lw $t0, 0($t4)            # var = *(ptr + i);

done:
```

Login: cs61c-____

Question 5: Mad about MIPS .... (25 pts, 20 min)

The factorial of a number n is defined as n*(n-1)*(n-2)*…*1. Factorials can be computed with the recursive definition

f(n) = n * f(n-1), where the base case is f(1) = 1

Implement the factorial function below recursively in TAL MIPS. You may assume that the argument is an unsigned integer > 0, and the result can fit in 32 bits so you do not have to handle overflow.

Assume you have a correctly implemented MIPS function `mult` which returns (in `$v0`) the value `$a0*$a1`, where `$a0` and `$a1` are treated as unsigned integers. Follow the hints given by the comments. We have given you some code and some labels to help guide you. **You must comment your code!** You may not need all the given space**.**

```
factorial:  ________addiu $sp, $sp, -8________   # Setup

            ________sw $ra, 0($sp)____________   # save the return address, since this
                                                   function calls a subfunction
            ________sw $a0, 4($sp)____________   # we need the original argument after
                                                   we call factorial recursively
            ________addiu $v0, $0, 1__________   # set return value for base case
                                                   (note we also use this register
                                                    for the base case comparison)
            ________bne $a0, $v0, rec_________   # Handle base case (if n != 1 then
                                                   do recursion)
            __________________________________

            __________________________________

            j done                               # else: this is base case -> return 1

rec:        __________________________________   # Recursive case

            ________addiu $a0, $a0, -1________   # compute argument for recursive call

            jal factorial                        # Call factorial

            ________lw $a0, 4($sp)____________   # get the original argument back.
                                                   (NOTE WE CANNOT ASSUME $A0 WILL
                                                   STILL BE VALID, EVEN THOUGH WE
                                                   WROTE THIS FUNCTION!)

            ________add $a1, $v0, $0__________   # set up second argument to multiply:
                                                   the result of our recursive call
            __________________________________

            jal mult                             # Call mult (compute n * fact(n-1))

            __________________________________   # the result we want to return is
                                                   in $v0 already
            __________________________________

done:       ________lw $ra, 0($sp)____________   # restore $ra

            ________addiu $sp, $sp, 8________    # restore stack

            jr $ra                               # return
```

Login: cs61c-__**Answers**__

Question 6: It's all MIPS to me... (35 pts, 25 min)

A) You are the assembler. Convert the following MAL MIPS code to TAL. Assume the labels are located at the addresses specified, and **adding additional instructions does not affect these addresses. If a MAL instruction is already TAL, simply rewrite it in the TAL column.**

Address	MAL	TAL
0x10000000	`entry: subiu $sp, $sp, 4`	**`addiu $sp, $sp, -4`**
0x10000004	`lbu $t0, 6($sp)`	**`lbu $t0, 6($sp)`**
0x10000008	`move $v0, $a0`	**`add $v0,$0, $a0`**
0x1000000C	`ble $v0, $0, entry`	**`blez $v0 entry`**
0x10000010	`j label`	**`j label`**
0x10000014	`sltiu $t0, $t1, 0x8000`	**`ori $at, $0, 0x8000`** **`slt $t0, $t1, $at`** **`#because sltiu sign extends`** **`#0x8000 cannot be represented in`** **`#16 bits`**
... ...	`# Some TAL instructions` `# which you can ignore`	There is nothing to translate to TAL here!
0x2000000C	`label: # Some instruction`	

(This space left intentionally blank. Feel free to doodle.)

Problem continued on next page.

Question 6: It's all MIPS to me (Continued) .... (35 pts, 25 min)

B) Assemble the following TAL code to its machine language representation. **You must show all your work (below) to receive credit**.

```
0x78000000   start:     addu   $t0, $t9, $s1       =    0x__03314012_______________
0x78000004   loop:      lw     $v0, -8($sp)        =    0x__8FA2FFF8_______________
0x78000008              beq    $v0, $0, done       =    0x__10400004_______________
0x7800000C              nop                        =    0x00000000
0x78000010              nop                        =    0x00000000
0x78000014              nop                        =    0x00000000
0x78000018              nop                        =    0x00000000
0x7800001C   done:      j      loop                =    0x__0A000001______________
```

Show your work here.

```
addu  $t0, $t9, $s1
```

```
lw    $v0, -8($sp)
```

```
beq   $v0, $0, done
```

```
j     loop
```