University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Summer 2009 Instructor: Jeremy Huddleston 2009-07-20

CS61C MIDTERM

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

<table>
<thead>
<tr>
<th>Last Name</th>
<th>Standard Grading</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td></td>
</tr>
<tr>
<td>Student ID Number</td>
<td></td>
</tr>
</tbody>
</table>

Login cs61c-

Login First Letter (please circle) a b c d e f
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) James Josh Paul

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 12 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.
- You have 80 minutes to complete this exam. No computers, PDAs, calculators, or other electronics. You may reference two books and a collection of personal notes (within reason, we have the final say!)
- 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. "ICE format" refers to the mebi, tebi, etc prefixes.
- If the question asks for a response as a numerical expression, you should express the answer in as reduced a form as possible (i.e. show what you would enter into a calculator if you had one available)
- Assume variable size and padding is consistent with the MIPS architecture we've been studying in class.
- If a sign is ever omitted, the value is assumed positive.

<table>
<thead>
<tr>
<th>Question</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minutes</td>
<td>1</td>
<td>14</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>80</td>
</tr>
<tr>
<td>Points</td>
<td>2</td>
<td>25</td>
<td>19</td>
<td>39</td>
<td>39</td>
<td>26</td>
<td>150</td>
</tr>
</tbody>
</table>

Score
Question 0: Hi, I’m your midterm. Who are you? (2 Pts, 1 Min)

Take a minute to write your name and login on every page of this exam. Make sure you are not missing any pages.

2 pts for success

Question 1: Just a lil bit... just a lil bit... (25 Pts, 14 Min)

a) How many different things can we represent with N bits?

\[ 2^N \] 2 points

\[ 2^N - 1 \] 1 point

b) How many more bits (than N) would we need if we want to double the number of things we can represent?

1 + 2 pts

c) How many more bits (than N) do we need if we want to instead triple the number of things we can represent?

9 + 2 pts

d) Fill in the table below to show how the following bits can be interpreted in different ways. If a particular field has no solution, answer “N/A”.

<table>
<thead>
<tr>
<th>bit pattern</th>
<th>1111</th>
<th>1111</th>
<th>0010</th>
<th>0100</th>
<th>0110</th>
<th>0000</th>
<th>0000</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>an unsigned hexadecimal number</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>an IEEE 32bit float (normalized binary scientific notation)</td>
<td>-1.0000000</td>
<td>1.0000000</td>
<td>1.117</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 ASCII characters</td>
<td>N/A</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>4 sign and magnitude bytes (in decimal)</td>
<td>-127</td>
<td>-1</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 one’s complement bytes (in decimal)</td>
<td>0</td>
<td>-1</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 two’s complement bytes (in decimal)</td>
<td>-1</td>
<td>-1</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Special cases:

1) -4 if you complemented the n’s complement positive numbers
2) -7 if you didn’t follow the above w/ sign magnitude, too.
Clint and John are trying to organize a collection of all their old cowboy buddies. These 21st century cowboys are always on the road and don’t like to be tied down, so there’s no need to store street address, just cell numbers. Use the code they came up with (below) to answer the questions that follow.

```c
#define NUM_FRIENDS 100
#define MAX_PHONE_LEN 13

struct entry {
    char *name;
    char phone_number[MAX_PHONE_LEN];
    int favorite;
};

typedef struct entry entry_t;

unsigned int num_buddies = 0;
entry_t *buddies[NUM_FRIENDS];

void add_buddy(char *name, char *phone_number, int favorite) {
    int i = num_buddies++;
    static int initialized = 0;

    if(!initialized) {
        for(int j=0; j < NUM_FRIENDS; j++)
            buddies[j] = NULL;
        initialized = 1;
    }

    buddies[i] = (entry_t *)malloc(sizeof(entry_t));
    buddies[i]->name = name;
    strcpy(buddies[i]->phone_number, phone_number, MAX_PHONE_LEN);
    buddies[i]->favorite = favorite;

    printf("\n", sizeof(name));
    printf("\n", sizeof(entry_t));
    printf("\n", sizeof(buddies));
    printf("\n", sizeof(*buddies));
    printf("\n", sizeof(buddies[i]->name));
    printf("\n", sizeof(buddies[i]->phone_number));

    printf("%u\n", *name);
    printf("%u\n", buddies[i]->name == name ? 1 : 2);
    printf("%u\n", buddies[i]->phone_number == phone_number ? 3 : 4);
    printf("%u\n", *buddies[i]->phone_number == *phone_number ? 5 : 6);
}

int main() {
    add_buddy("Dean", "5105551234", 1);
    ...
}
```
a) Fill in the table below with the value printed as a result of each `printf` statement in the code. Assume `add_buddy` is being called for the first time as shown in `main()`.

<table>
<thead>
<tr>
<th>Code</th>
<th>Printed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(name));</code></td>
<td>4</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(entry_t));</code></td>
<td>24</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(buddies));</code></td>
<td>400</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(*buddies));</code></td>
<td>4</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(buddies[i]-&gt;name));</code></td>
<td>4</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, sizeof(buddies[i]-&gt;phone_number));</code></td>
<td>13</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, *name);</code></td>
<td>68</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, buddies[i]-&gt;name == name ? 1 : 2);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, buddies[i]-&gt;phone_number == phone_number ? 3 : 4);</code></td>
<td>4</td>
</tr>
<tr>
<td><code>printf(&quot;%u\n&quot;, buddies[i]-&gt;*phone_number == *phone_number ? 5 : 6);</code></td>
<td>5</td>
</tr>
</tbody>
</table>

| pt each |
b) Match the expressions on the left with their final location in memory on the right (assume that add_buddy has been called at least once as shown in main).

E  struct entry
B  num_buddies
A  strlcpy
D  *buddies
C  name
D  buddies[0]->phone_number
B  initialized
A  add_buddy
B  buddies

A. Code (text)
B. Static (global, data)
C. Stack
D. Heap
E. None / Other
Question 3: Insert wit here (39 Pts, 20 Min)

We have a simple doubly linked list that consists of witty comments (as C strings). The structure appears below with an example:

```c
struct node {
    struct node* prev;
    char *payload;
    struct node* next;
};
```

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

a) Fill in the code below to implement the `insert_after` function. This function inserts a node in the linked list after the `given_node` and copies the given string `data` into the `payload` field. The function returns a pointer to the `new_node` or NULL if the operation can not be completed. You can assume that the input is always valid.

```c
struct node* insert_after(struct node* given_node, char * data) {
    // Allocate space
    struct node *new_node = malloc(sizeof(struct node)) + 1;
    if (new_node == NULL) // Check for errors
        return NULL + 1;

    // Setup the payload
    new_node->payload = (char *) malloc(sizeof(char) * (strlen(data) + 1));
    if (new_node->payload == NULL)
        free(new_node), +1
    return NULL + 1;

    // Setup Links
    new_node->next = given_node->next; new_node->prev = given_node;
    if (given_node->next != NULL + 1)
        given_node->next->prev = new_node;
    given_node->next = new_node;

    return new_node + 1;
}
```
b) The `free_list` function takes a pointer to the head of a doubly linked list and frees all memory that was allocated for the list. You may assume it is correctly given the head of the doubly linked list.

```c
void free_list(struct node* head) {
    free(head->payload);
    free(head);
    free_list(head->next);
}
```

Describe all the bugs in the given `free_list` function. Please enumerate ("number") your answer.

1) Calls `free(head)` before `free_list(head->next)`, cannot dereference head after freeing.

2) No base case on recursion, will free a null pointer eventually. -2 for head many solutions. -1 or -2 for answers that were wrong, but would break things.

Implement a version of `free_list` which corrects all the bugs.

```c
void free_list(struct node* head) {
    if (head == NULL) return;
    free(head->payload);
    free_list(head->next);
    free(head);
}
```

These were the two most common solutions, others were accepted.
**Question 4: Assemble this! (39 Pts, 20 Min)**

a) Below is a section of MAL code. Convert this code into its TAL equivalent using the table below and label the address in memory where the TAL instructions start. The address for `start` is 0x1020.

```mal
start:    subiu $a0, $v0, 0xDEADBEEF
          move $a1, $s0
          jal check_value
          beqz $v0, start  # branch if $v0 is zero
```

<table>
<thead>
<tr>
<th>Start Address</th>
<th>MAL</th>
<th>TAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1020</td>
<td>subiu $a0, $v0, 0xDEADBEEF</td>
<td>lui $at, 0x DEAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ori $at, $at, 0xBEEF</td>
</tr>
<tr>
<td></td>
<td>move $a1, $s0</td>
<td>suba $a0, $v0, $at</td>
</tr>
<tr>
<td>0x102c</td>
<td></td>
<td>or $a1 $a0 $s0 $s0</td>
</tr>
<tr>
<td>0x1030</td>
<td>jal check_value</td>
<td>jal check_value</td>
</tr>
<tr>
<td>0x1034</td>
<td>beqz $v0, start</td>
<td>beq $v0 $0 start</td>
</tr>
</tbody>
</table>

b) What toolchain stage (compiler, assembler, linker, loader) would be responsible for finishing the conversion from MAL to TAL for each of the 4 instructions?
c) Convert the following TAL instructions into their machine level representation in hex. Show your work for full credit.

i) lw $t0 4($sp)  Hex: 0x 8FA80004

\[ \begin{array}{ccccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

ii) addu $s0 $t1 $a0  Hex: 0x 01248021

\[ \begin{array}{ccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

iii) beq $t0 $t1 -4  Hex: 0x 1109FFFC

\[ \begin{array}{ccccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

5 pts each
Question 6: Gi’me, gi’me more, gi’me more, gi’me, ... (26 Pts, 15 Min)

Amy recognized that rounding errors occurred when adding together two of her floating point numbers. Bob, one of her friends, suggested that she could increase the precision and express more numbers by increasing the size of the significand/mantissa field.

Amy decided to change the floating point spec to see if this would work. She chose to move two bits from the exponent field to the significand field:

<table>
<thead>
<tr>
<th>1</th>
<th>6</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>EEEEE</td>
<td>MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM</td>
</tr>
</tbody>
</table>

Answer the following questions below using these modified field sizes. Everything else follows the normal IEEE floating point standard. Where you may have a choice in algorithm, choose the *default* mode for IEEE floating point. If a question asks for a floating point value, write it in normalized binary scientific notation.

a) What is the exponent bias for Amy’s new configuration? (3 pts)

\[
\text{Exponent bias} = 2^{\text{exp-1}} - 1 = 2^6 - 1 = 2^5 - 1 = 31
\]

-1 Math Error / off by 1 Error
-3 Solution that doesn’t make any sense

b) What is the smallest positive number that Amy can accurately represent in normalized form? (5 pts)

\[
\begin{align*}
S & \text{ EEEEE } M \ldots \ldots M \\
0 & \text{ 0000000000000 } \\
(-1)^0 \times 2^{1-31} \times 1.0 & = (-1)^0 \times 2^{-30} \\
& = 2^{-30}
\end{align*}
\]

-1 off by 1 Error
-5 bad solution (doesn’t know difference between normalized & denormalized + one more)
c) What is the implicit exponent for denormalized numbers in Amy's new configuration? 

\[-(\text{bias} - 1) = 1 - \text{bias} = 1 - 31 = -30\]

- 1 off by 1 error
- 3 bad solution (positive implicit exp) after nonsense

\[- \text{bad solution (positive implicit exp)}\]

\[-3\text{ missing either } 2^{-30} \text{ or } 2^{-25}\]

- 5 bad solution

d) What is the smallest positive number that Amy can accurately represent? 

\[-(1)^{0} \times 2^{-30} \times 2^{-25} = 2^{-55}\]

\[-5\text{ missing either } 2^{-30} \text{ or } 2^{-25}\]

- 5 bad solution

e) What is the smallest positive number (in Amy's scheme) to which you can add the number 5.0 and have the result be unchanged? (ie: what is the smallest x such that x + 5.0 = x) 

5.0 = 101₂

\[-1\text{ off by 1, with subnormal bits for the significand}\]

\[-2^{\text{-8}}\text{ other issues}\]

\[\text{11/9 enough guard to want a sign bit}\]

-10 blank/none sense
Reference: Where have all the cowboys gone?

Clint and John are trying to organize a collection of all their old cowboy buddies. These 21st century cowboys are always on the road and don't like to be tied down, so there's no need to store street address, just cell numbers. Use the code they came up with (below) to answer the questions that follow.

```c
#define NUM_FRIENDS 100
#define MAX_PHONE_LEN 13

struct entry {
    char *name;
    char phone_number[MAX_PHONE_LEN];
    int favorite;
};
typedef struct entry entry_t;

unsigned int num_buddies = 0;
entry_t *buddies[NUM_FRIENDS];

void add_buddy(char *name, char *phone_number, int favorite) {
    int i = num_buddies++;
    static int initialized = 0;

    if(!initialized) {
        for(int j=0; j < NUM_FRIENDS; j++)
            buddies[j] = NULL;
        initialized = 1;
    }

    buddies[i] = (entry_t *)malloc(sizeof(entry_t));
    buddies[i]->name = name;
    strlcpy(buddies[i]->phone_number, phone_number, MAX_PHONE_LEN);
    buddies[i]->favorite = favorite;

    printf("%u\n", sizeof(name));
    printf("%u\n", sizeof(entry_t));
    printf("%u\n", sizeof(buddies));
    printf("%u\n", sizeof(*buddies));
    printf("%u\n", sizeof(buddies[i]->name));
    printf("%u\n", sizeof(buddies[i]->phone_number));

    printf("%u\n", *name);
    printf("%u\n", buddies[i]->name == name ? 1 : 2);
    printf("%u\n", buddies[i]->phone_number == phone_number ? 3 : 4);
    printf("%u\n", *buddies[i]->phone_number == *phone_number ? 5 : 6);
}

int main() {
    add_buddy("Dean", "510551234", 1);
    ...
}
```