CS61c Summer 2014 Midterm Exam

Read this first: This exam is marked out of 100 points, and amounts to 30% of your final grade. There are 7 questions across 9 pages in this exam. The last question is extra credit – it will not be included in our calculation of the final class curve.

You may use one double sided 8.5x11" page of handwritten notes, as well as the MIPS green sheet provided to you as references throughout this exam. Otherwise your only resource is what you know or can deduce during the examination period. Answers should be placed inside the spaces provided on the test, and should be no longer than is necessary to communicate the solution. Misplaced or exceedingly verbose solutions may be marked down.

You may find this exam difficult. If you do, then *do not panic*. Remember that different questions have different weights, and some may be more difficult than others. Answer those questions and subquestions in which you are most confident first, then move on to more difficult questions. We recommend that you skim over the entire exam before attempting to answer any questions. Also be sure to read all questions carefully before answering them, and do not hesitate to ask for clarifications if you are unclear on what a question is asking.

Your Name: _____

Your TA: Andrew David Fred Hokeun Kevin

Login: _____

Login of the person to your left: _____

Login of the person to your right: _____

Question	Points	Time (minutes)	Score
1	18	32	
2	20	36	
3	19	34	
4	14	25	
5	17	30	
6	12	21	
7	1	2	
Total	101	180	

Q1: Silly Rabbit, Trits Are for Kids

A new memory technology with three distinct states is exploding into the technology industry! Let's see if we can't develop some new number representations to take advantage of this new development.

(a) First, define a rule, analogous to what we use for binary numerals, for determining the unsigned value of a ternary numeral $d_n d_{n-1} \ldots d_0$, and use this rule to convert 2102_3 into decimal:

$$\text{unsigned}(d_n d_{n-1} \ldots d_0) = \sum_{i=0}^{n} 3^i \cdot d_i$$
$$\text{unsigned}(2102_3) = 65$$

(b) Next we'd like to define an analogue to two's complement for ternary numerals, which we'll call three's complement. Three's complement numbers should be as evenly distributed between positive and negative as possible (favor negative if necessary), should have a zero at 0_3, and should increase in value when incremented as an unsigned value (except in the case of overflow). Define a rule for negating a three's complement number.

"flip" all of the trits (replace 2s with 0s, 0s with 2s), and add 1

(c) What is the most positive possible three's complement 8-trit number? Using this result, specify a rule for determining if a three's complement number is positive or negative.

$1 \ldots 1_3$. A number is negative if it has unsigned magnitude greater than $1 \ldots 1_3$

(d) There are two different two's complement numbers who are their own inverse. Specify these numbers.

0, 0b10...0

(e) Which numbers in three's complement are their own inverse?

0

(f) What arithmetic operation is a shift left logical equivalent to with three's complement numbers?

Multiplication by a power of three.

(g) What arithmetic operation is a shift right arithmetic equivalent to with *two's complement* numbers?

Division by a power of two, followed by a floor

Q2: The Smorgasbord

(a) Complete the following C function according to its comment. You may assume that the architecture on which this code is run uses two's complement to represent signed integers.

```
/* Returns an unsigned integer whose first IDX low order bits
 * are the same as in LO, and whose high order bits are the same
 * as HI's high order bits. Assume 0 <= IDX < sizeof(int). */
unsigned splice(unsigned lo, unsigned hi, int idx) {
    return (lo & (1 << idx) - 1) | (hi & -1 << idx);
}
```

(b) Specify whether each of the following comparisons would evaluate to true or false, interpretting the hex numbers as IEEE floats:

T/**F** `0xFF800001 < 0xDEADBEEF`
T/**F** `0xF000AC03 < 0x0F34DD7C`
T/**F** `0xDEADBEEF < 0xFFFFFFFF`
T/**F** `0x02000000 < 0x017BEEEF`
T/**F** `0x00000000 == 0x80000000`

(c) We select a 32-bit number uniformly at random from the set of all 32-bit numbers. What is the probability that the bit pattern we select is a real number when interpreted as an IEEE float (infinities are *not* real numbers)?

$$\frac{255}{256}$$

(d) Write, in hexadecimal, a MIPS instruction which will cause an infinite loop if ever executed, anywhere, in any MIPS program.

0x1000ffff

(e) Write a single command which will commit all changes to currently tracked files along with the log message "VCS is pretty cool" inside of a git repository.

`git commit -am "VCS is pretty cool"`

(f) A program spends 15% of its time inside of a function `foo`, 10% inside of `bar`, and 5% in `baz`. Your colleague makes a breakthrough that doubles the speed of `foo`. How much speedup would you need to achieve in `bar` in order to match the performance gains from improving `foo`? How much speedup would you need in `baz`?

4x for `bar`, impossible for `baz`

(g) Suppose a bit of register memory costs 1 cent. What is the total cost of the saved registers (as opposed to volatile) on a MIPS chip?

$32 \cdot (1 + 1 + 8) = 320$ cents OR $32 \cdot (1 + 1 + 1 + 1 + 8) = 384$ cents

if you included `$fp` and `$gp`

Q3: C to This Mess, Will You?

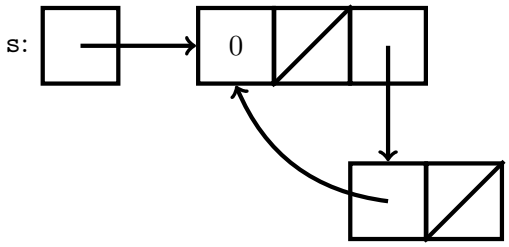
A possible definition for general graph structures is presented below

```
typedef struct node {
    char breadcrumb;          /* For dfs traversals */
    char *name;               /* Node contents */
    struct node **neighbors; /* Nodes that I can reach.
                              * Null terminated. */
} node;
```

(a) Assuming `V` has type `(node **)`, that we are operating on a 32-bit architecture, and that structs are tightly packed, specify the value of each of the following expressions:

`sizeof(&V[0])` = 4
`sizeof(*V)` = 4
`sizeof(V[1][0])` = 9

(b) Draw a box and pointer diagram of a *minimal, cyclic* graph (i.e. a graph with a loop, with a few nodes/edges as possible), starting from `node *s`. For this part of the problem use 0 for any `breadcrumb`s and `NULL` for any `name`s.



(c) How many bytes were necessary for the graph structure from (b)?

21 bytes

Continued on next page

(d) We would like to write a procedure for freeing a `malloc()`'d graph, along with the data referenced inside the graph (e.g. we want to free `name` inside of each node as well). Complete the following two functions so that they do so, remembering that *graphs can contain cycles*. You may assume that any dynamic memory allocations you make in your solution succeed (i.e. there is no need to check calls to `malloc()` and friends), and you may also assume that when `free_graph()` is called all of the nodes in the graph will have their `breadcrumb`s set to 0.

```
void free_graph(node *start) {
    node **nodes = NULL;
    size_t num_nodes = 0;
    fg_helper(start, &nodes, &num_nodes);
    for (size_t i = 0; i < num_nodes; i += 1) {
        free(nodes[i]->name);
        free(nodes[i]->neighbors);
        free(nodes[i]);
    }
    if (nodes) free(nodes); //oops, missing from original exam
}

void fg_helper(node *nd, node ***nd_arr, size_t *n) {
    size_t i = 0;
    if (nd->breadcrumb)
        return;
    nd->breadcrumb = 1;
    *nd_arr = realloc(*nd_arr, sizeof(node *) * (*n + 1));
    (*nd_arr)[(*n)++] = nd;
    while (nd->neighbors[i]) {
        fg_helper(nd->neighbors[i++], node_arr, n);
    }
}
```

Q4: It's a Bloody MIPStery

Consider the following MIPS function `mystery`:

```
1  mystery:
2      beq   $a1, $0,  L3
3      sll   $a1, $a1, 2
4      addu  $a1, $a0, $a1
5      move  $t8, $a0
6  L1:
7      addiu $t8, $t8, 4
8      beq   $t8, $a1, L3
9      move  $t9, $t8
10 L2:
11     beq   $t9, $a0, L1
12     lw    $t0, -4($t9)
13     lw    $t1,  0($t9)
14     slt   $t2, $t1, $t0
15     beq   $t2, $0,  L1
16     sw    $t1, -4($t9)
17     sw    $t0,  0($t9)
18     addiu $t9, $t9, -4
19     #PRINTF HERE
20     j     L2
21 L3:
22     jr $ra
```

(a) What would the type signature of `mystery` be in a corresponding C program?

`void mystery(int *arr, size_t n);`

(b) Explain, in no more than 10 words (seriously, more than that and we won't grade it), what `mystery` does.

It's an insertion sort

(c) Suppose we wanted to modify the code to include an invocation of the library function `printf` at the line marked `#PRINTF HERE`, but not modify any other lines of code in the function. List the registers which would need to be saved onto the stack before calling `printf` and restored afterwards. If no registers need to be saved/restored, write "none".

`$t8, $t9, $a0, $a1, $ra`

(d) Suppose we wanted to modify the code so that the type of its first argument was `(char*)`. Specify, *in ascending order*, the line numbers which would need to be changed to do so.

3,7,12,13,16,17,18

Q5: Money, it's cache, stick it in your computer and maaake it fast.

For this problem we will be working on a 32-bit byte addressed MIPS machine, with a single two-way set-associatve 16KiB cache, a write-back policy, and 64B blocks.

(a) Specify the T:I:O breakup for our system: 19:7:6

(b) How many bits of hardware are there in each row of the cache? 64*8 + 21 = 533

A *linked list of arrays* is a data structure which seeks to provide a compromise between using arrays for collections of data and using linked lists for collections of data. A possible definition for such a data structure is presented below, along with two different versions of a copy function:

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

Assuming that we have a list of arrays containing 960 (i.e. $15 \cdot 64$ `int`s) elements which we wish to copy

(c) What is the best-case hit rate for version 1 of the code? 31/32 Version 2? 59/60

(d) What is the worst-case hit rate for version 1 of the code? 7/8 Version 2? 0

The subquestions below are three *independent* variations on the original cache settings.

(e) If our cache were 4-way set associative, what would the best case hit rate be for version 2? 59/60

(f) If our cache were fully associative, what would the worst case hit rate be for version 2? 14/15

(g) If our cache were only 1KiB, what would the best case hit rate be for version 1? 31/32

Q6: AMATter of Performance

Consider the following C function, which returns the sum of all the values contained within the tree:

```
typedef struct bt {
    unsigned val;
    struct bt *left, *right;
} bt;
```

```
unsigned sum(bt *root) {
    if (!root)
        return 0;
    return root->val
           + sum(root->left)
           + sum(root->right);
}
```

(a) Finish compiling `sum` into 32-bit TAL MIPS in the space below.

```
sum:
    addiu $sp, $sp, -12
    sw    $ra, 8($sp)
    sw    $s1, 4($sp)
    sw    $s0, 0($sp)
    xor   $v0, $0, $0
    beq   $a0, $0, ret
    addu  $s0, $a0, $0
    lw    $s1, 0($s0)
    lw    $a0, 4($s0)
    jal   sum
    addu  $s1, $s1, $v0
    lw    $a0, 8($s0)
    jal   sum
    addu  $v0, $s1, $v0
ret:
    lw    $ra, 8($sp)
    lw    $s1, 4($sp)
    lw    $s0, 0($sp)
    addiu $sp, $sp, 12
    jr $ra
```

Continued on next page

Suppose that when we run our code we find the following hit rates and hit times:

- I$ always hits in 1 cycle
- L1$ hits 90% of the time, and hits in 1 cycle
- L2$ hits 75% of the time, and hits in 10 cycles
- DRAM always hits, and hits in 100 cycles
- Ideal CPI (CPI without cache misses) of 2

(b) What is the AMAT within our data cache?

1 + 0.1*10 + 100*0.1*0.25 = 2 + 2.5 = 4.5

(c) What is the actual CPI from our test run? Remember, our I$ always hit, but our D$ did not always hit.

$\text{CPI}_{\text{actual}} = 2 + 0.5 \cdot 3.5 = 3.75$

(d) We want to improve our AMAT to no more than 3 cycles by expanding our L2$. At least how high must our L2$'s local hit rate be to achieve this goal?

90%

Q7: One for the Road

"Syntactic sugar causes cancer of the semi-colons" – Alan Perlis