CS61C Summer 2012 Midterm

Your Name: ________________________________ **SID:** ____________________

Your TA (Circle): Raphael Paul Brandon Sung Roa

Name of person to your LEFT: ____________________________________

Name of person to your RIGHT: ____________________________________

This exam is worth 110 points and will count for 20% of your course grade.

The exam contains 9 questions on 13 numbered pages, including the cover page. Put all answers on these pages; don't hand in stray pieces of paper.

Question 0: You will receive 1 point for properly filling out this page as well your login on every page of the exam.

Question	Points (Minutes)	Score
0	1 (0)	
1	15 (26)	
2	10 (14)	
3	10 (18)	
4	9 (16)	
5	10 (18)	
6	17 (26)	
7	12 (22)	
8	26 (40)	
Total	110 (180)	

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.

Signature: ________________________

Question 1: *Potpourri – Hard to spell, nice to smell…* (15 points, 26 minutes)

a) **True/False:**

T F We prefer two's complement over the unsigned representation because two's complement can represent more values.

T F The assembler uses symbol tables to resolve absolute addresses.

T F A program will always execute faster in a RISC architecture than a CISC architecture.

T F The greater the number of memory accesses in a program, the greater the AMAT.

T F Pseudo-instructions do not always use `$at`.

T F Since `$s0` is a "saved register," it does not need to be saved before any function calls.

---

b) Fill in the function below, which returns a new copy of the argument (struct definition not shown):

```
struct something *cpySomething(struct something *old) {

        ________________________________________________________________

        ________________________________________________________________

        return new;
}
```

---

c) Here is what is currently on the heap:

A	**A**	**A**			**C**	**D**	**D**		

The order of allocation/frees: A allocated, B allocated, C allocated, B freed, D allocated

What allocation strategy was used? ______________________________

---

d) What is the average CPI of the program described in the table to the right? Which is better: halving memory access cycles or arithmetic cycles and why?

Instruction Category	Cycles	Frequency
Memory Access	10	0.1
Arithmetic	2	0.4
Branch	3	0.2
Comparison	1	0.3

e) I'm getting the message "`cannot execute binary file`". In one or two sentences, explain what the problem is and how to fix it.

---

f) In our 32-bit single-precision floating point representation, we decide to convert one significand bit to an exponent bit. How many **denormalized numbers** do we have relative to before? (Circle one)

More Fewer

Rounded to the nearest power of 2, how many denorm numbers are there in our new format? (Answer in IEC format)

____

<u>Question 2</u>: *Flippin' Fo' Fun* (10 points, 14 minutes)

Assume that the most significant bit (MSB) of x is a 0. We store the result of flipping `x`'s bits into `y`. Interpreted in the following number representations, how large is the <u>magnitude</u> of `y` relative to the <u>magnitude</u> of `x`? Circle ONE choice per row.

Unsigned	|y| < |x|	|y| = |x|	|y| > |x|	Can't Tell
One's Complement	|y| < |x|	|y| = |x|	|y| > |x|	Can't Tell
Two's Complement	|y| < |x|	|y| = |x|	|y| > |x|	Can't Tell
Sign and Magnitude	|y| < |x|	|y| = |x|	|y| > |x|	Can't Tell
Biased Notation (e.g. FP exponent)	|y| < |x|	|y| = |x|	|y| > |x|	Can't Tell

Question 3: *Doctor Who?!?* (10 Points, 18 Minutes)

The Daleks are invading the Earth again, and we need the help of the Doctor! Find the errors in this code and fix them so that the code correctly prints **"The 10th Doctor and the Blue Police Box"**. There is exactly one coding error for each function and function call pair and can be fixed by changing 5 or 6 lines total. Fill in the corrections in the blanks on the opposite page.

```
void whichDoctor(int* input) {
    input = 10;
}

void doctorChanger(char** input1, char** input2) {
    char* temp = *input1;
    *input1 = *input2;
    *input2 = temp;
}

char* policeBoxGiver(char* input) {
    *input = "The Master";
    return "Police Box";
}

char* colorMaker(void) {
    char* color = malloc(sizeof(char) * 4);
    color[0] = 'B';
    color[1] = 'l';
    color[3] = 'u';
    color[2] = 'e';
    color[4] = 0;
    return color;
}

char* colorFixer(char* input) {
    char temp = *(input+2);
    *(input+2) = *(input+1);
    *(input+1) = temp;
}

int main(void) {
    int * ith = malloc(sizeof(int));
    whichDoctor(ith);
    char* doctor = "Master";
    char* master = "Doctor";
    char* details = "and the";
    char* color = colorMaker();
    colorFixer(color);
    char* box = "David Tennant";
    doctorChanger(doctor, master);
    policeBoxGiver(box);
    printf("The %dth %s %s %s %s",*ith,doctor,details,color,box);
}
```

Line #	**Corrected Code**
______	____
______	____
______	____
______	____
______	____
______	____

Question 4: *Let Me Float This Idea By You* (9 Points, 16 Minutes)

For a very simple household appliance like a thermostat, a more minimalistic microprocessor is desired to reduce power consumption and hardware costs. We have selected a **16-bit** microprocessor that does not have a floating-point unit, so there is no native support for floating point operations (no `float`/`double`). However, we'd still like to represent decimals for our temperature reading so we're going to implement floating point operations in software (in C).

a) Define a new variable type called `fp`:

____

We have decided to use a representation with a **5-bit exponent field** while following all of the representation conventions from the MIPS 32-bit floating point numbers **except denorms**.

Fill in the following functions. Not all blanks need to be used. <u>You can call these functions and assume proper behavior regardless of your implementation</u>. Assume our hardware implements the C operator ">>" as *shift right arithmetic*.

b)

```
/* returns -num */
fp negateFP(fp num) {
      return ____________________________________________________________;
}
```

c)

```
/* returns the signed value of the exponent */
int getExp(fp num) {

      ____________________________________________________________________

      return ____________________________________________________________;
}
```

d)

```
/* multiplies floating point num by 2^n, while detecting over/underflow */
/* remember, there are no denorms */
fp multPow2(fp num,int n) {

      ____________________________________________________________________

      if(____________________________________________) exit(1);  #overflow

      if(____________________________________________) exit(-1); #underflow

      ____________________________________________________________________

      return ____________________________________________________________;
}
```

<u>Question 5</u>: *Who Says Less is Better?* (10 points, 18 Minutes)

We're going to take a page out of the ARM book and design a new instruction set architecture with just **16 32-bit registers**. This means that we only need 4-bit register fields in our instructions.

a) How many extra bits do we have now for other fields in the following formats?

R: ____________ J: ____________

b) For R-format instructions, would you give the extra bits to `opcode`, `shamt`, or `funct`? __________
Explain your choice in a sentence or two (no credit without explanation):

For I-format instructions, we naturally give the extra bits to the `immediate` field, resulting in the following format:

`[ opcode (6) | rs (4) | rt (4) | immediate (18) ]`

c) What fraction of our address space can we now reach with a branch instruction?

____

d) Assume our PC currently contains the address `0x08000000`.
What is the LOWEST address (in hex) we can reach with a branch? ____________________

e) Write out the Verilog pseudocode (as in the OPERATION column on the MIPS Green Sheet) for `beq`.
Make sure you specify what `BranchAddr` is.

<u>Question 6</u>: *Cache in While You Can* (17 points, 26 Minutes)

Consider a single 4KiB cache with 512B blocks and a write-back policy. Assume a 32-bit address space.

a) If the cache were direct-mapped,

# of rows? __________ # of offset bits? __________

b) If the cache were 4-way set associative,

# of tag bits? __________ # of index bits? __________ # of bits per cache slot? __________

Consider an array of the following `location` structs:

```
typedef struct {
      ... // some undefined number of other struct members
      int visited;
      int danger;
} location;

location locs[NUM_LOCS];
```

Here's a piece of code that counts the number of places we've visited. Assume this gets executed somewhere in the middle of our program, that `count` is held in a register, and the size of the array is greater than 4 KiB.

```
for(int i = 0; i < NUM_LOCS; i++)
      if(locs[i].visited) count++;
```

c) What's the fewest possible number of bytes written to main memory? _____________

d) What's the greatest possible number of bytes written to main memory? _____________

Now consider if we store the `visited` and `danger` information in individual arrays instead:

```
int visited[NUM_LOCS];
int danger[NUM_LOCS];
```

e) This way, the cache can exploit better ____________________ for the above task.

We can expect a ____________________ (higher or lower) miss rate

because of the change in the number of ____________________(type of cache miss) misses.

Consider the following code with `NUM_LOCS > 2^10`.

```
for(int i = 0; i < NUM_LOCS; i++)
      if(visited[i] && danger[i] > 5) count++;
```

Two memory accesses are made per iteration: one into `visited`, the other into `danger`. Assume that the cache has no valid blocks initially. **You are told that in the worst case, the cache has a miss rate of 100%.** Consider each of the following possible changes to the cache individually.

f) Mark each as **E**, if it eliminates the chances of this worst-case scenario miss rate, **R** if it reduces the chances, or **N** if it's not helpful.

- More sets, same block size, same associativity ______
- Double associativity, half block size, same total cache size ______
- Everything stays the same but use a write-through policy instead ______

Question 7: *Can't Make Copies Fast Enough* (12 points, 22 Minutes)

We are revisiting our friend the Fast String Copy from lecture! Recall that the function prototype in C is as follows:

```
char *strcpy(char *dst, char *src);
```

Consider the following MIPS implementation of this function:

```
        jal    strcpy  # begin function call
        ...
strcpy:
        addi   $v0,$a0,0
loop:   lb     $t0,0($a1)
        sb     $t0,0($a0)
        addiu  $a0,$a0,1
        addiu  $a1,$a1,1
        beq    $t0,$zero,exit
        j      loop
exit:   jr     $ra
```

Suppose we are running code on a machine with the following cache parameters:

- **Unified** L1$ with a hit time of 2 cycles and a hit rate of 95%
- Miss Penalty to main memory of 200 cycles
- Base CPI of 1.5 (in the absence of cache misses)

a) Calculate our machine's AMAT:

b) What is the CPI of a single call to `strcpy` with `src = ""` (the function call includes the `jal`)?

c) We decide to add a L2$ to reduce our AMAT to 6. Our L2$ has a hit time of 20 cycles. What's the worst Local Hit Rate that will still meet our AMAT goal?

d) In addition to speeding up our architecture, we want to speed up our code, so we decide to eliminate the return value (presumably the caller retains a copy of the destination pointer). In this case, the `strcpy` function above can be rewritten in just 6 instructions. Write out this implementation in the blanks below, introducing any necessary labels (don't worry about any label name clashes with `strcpy`).

```
strcpy2:

        ____________________________________

        ____________________________________

        ____________________________________

        ____________________________________

        ____________________________________

        ____________________________________
```

e) If we call `strcpy` and `strcpy2` on the same `src` string of length `n+1=N` (`N` *includes* `'\0'`), what is the ratio of instructions executed in `strcpy` to instructions executed in `strcpy2` (including the `jal`)? Leave your answer in terms of `N`.

f) Is the ratio in part (e) the same as the relative performance between these two functions? In a sentence or two, explain why or why not.

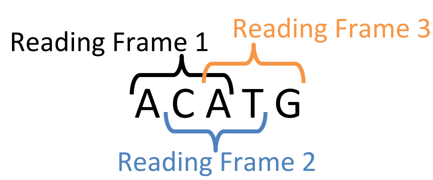
Question 8: *Putting the Science in Computer Science* (26 points, 40 minutes)

DNA can be called the "alphabet of life." From a *very* simplified view, DNA within a cell produces amino acids, which in turn produce proteins, which are the building blocks for most of your body. Here we'd like to write some code for examining a strand of DNA.

a) DNA is made up of *nucleotides*, which we write shorthand as A, C, G, and T. DNA is in base 4 (quaternary)! Fill in the table below, using the DNA nucleotide symbols in alphabetical order (A < C < G < T).

Decimal	DNA
	CAT
50	

An amino acid is encoded by three nucleotides. Because DNA is found in long strands, the following 5 nucleotides can be read 3 different ways:



The sequence **ATG** (as seen in the 3rd reading frame) signals the beginning of a protein ("start codon").

b) Fill in the blanks on the opposite page for the **recursive** function `find_start` in MIPS that returns the position of the first start codon found in the given strand of DNA. Assume each nucleotide is stored as a `char` in memory. *Blanks do not necessarily need to be filled*. Maximum points awarded for using the *fewest* amount of registers and memory.

[Answer the following AFTER looking at the code]

Assume we call `find_start` from main with `char dna[] = "GCATGC";`.

c) How many total frames are created on the Stack (not including `main`)? ______________

d) What is the maximum depth of the Stack (in # of frames, not including `main`)? ______________

e) What will the line `j ret` look like once this file is run through the assembler? __________________

f) Where will the label `ret` show up? (Circle one)

Symbol Table   Relocation Table   Both   Neither

C function prototype:
```
/* dna: start address of DNA strand */
/* pos: search position from start of strand */
int find_start(char *dna, int pos);
```

this is a recur. function don't forget dumby

```
find_start:
        addiu $sp,$sp,-4___   # PROLOGUE
        sw $ra, 0($sp)   #don't need to set up par. since dna stays the same for jal call
        __________________

        jal    strlen         # call strlen(dna); Assume strlen doesn't
                                  # change $a0 or $a1
        slti $t0, $v0, 3_____ # make sure we don't read past the end of
        __________________        # the array

        __________________
        beq    $t0,$0,chk     # 'chk' for check if start codon
        addi   $v0,$0,-1      # return -1 (start codon not found)
        j      ret            # 'ret' for return
chk:    Text______________
        lb     $t1,0($t0)
        addi   $t2,$0,65       why do you add 65?
        bne    $t1,$t2,rec    # 'rec' for recurse
        lb     $t1,1($t0)
        addi   $t2,$0,84__
        bne    $t1,$t2,rec
        lb     $t1,2($t0)
        addi   $t2,$0,_71_
        bne    $t1,$t2,rec
        add $v0, $t2, $0______ # return current position
        j      ret
rec:    lb $t1, 4($a1)_____   # recurse at next position
        addi $t1, $t1, 1__
        Text______________
        jal    find_start
        Text______________
        Text______________

ret:    __________________    # EPILOGUE
        sw $ra, 0($sp)____
        addiu $sp,$sp,4___
        jr     $ra
```

BACK OF EXAM

(Any work on this page will not be graded)