CS61C Summer 2013 Midterm

Your Name: ________________________________ **SID:** ___________________

Your TA (Circle): Albert Kevin Justin Shaun Jeffrey Sagar

Name of person to your LEFT: ____________________________________

Name of person to your RIGHT: ____________________________________

This exam is worth 95 points and will count for 24% of your course grade.

The exam contains 7 questions on 14 numbered pages. Put all answers in the spaces provided. Some pages are intentionally left blank and will not be graded.

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	10 (16)	
2	27 (48)	
3	15 (30)	
4	8 (16)	
5	22 (45)	
6	12 (25)	
Total	**95 (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: ______________________

PAGE INTENTIONALLY LEFT BLANK

(Any work on this page will not be graded)

<u>Question 1</u>: *Reppin' Yo Numbas* (10 points, 16 minutes)

For this question, we are using 16-bit numerals. For Floating Point, use 1 sign bit, 5 exponent bits, and 10 mantissa bits. For Biased use a bias of $-2^{15}+1$.

a) Indicate in which representation(s) the numeral is **closest to zero**: Two's Complement (T), Floating Point (F), or Biased (B). The first one has been done for you. NaN is not valid in comparisons.

	Numeral:	**Closest to zero:**
1)	`0x0000`	__TF__
2)	`0xFFFF`	_______
3)	`0x0001`	_______
4)	`0xFFFE`	_______
5)	`0x8000`	_______
6)	`0x7FFF`	_______

b) We now wish to **add** the numerals from **top to bottom (1 to 6)**. However, it is possible that we encounter an error when performing these addition operations. For each number representation, state the FIRST error that is encountered and which numeral causes it; if no error is encountered, answer "no error."

Possible arithmetic errors are: OVERFLOW, UNDERFLOW, NaN, and ROUNDING (assume we are rounding using a truncating scheme).

Representation:	**Arithmetic Error:**	**Numeral #:**
Two's Complement	____________________	_____
Floating Point	____________________	_____
Biased	____________________	_____

Question 2: *This Problem is Like a Box of Chocolates* (27 points, 48 minutes)

a) In C, `char` is in fact a *signed* variable type. Assume we have 4 `chars` != 0x00 loaded into a 32-bit `int`, one in each byte. Complete the function below that will negate the `char` in byte `i` of the `int` "in place", with byte 0 being the least significant and byte 3 being the most significant.

```
void negByte(int *data, char i) {
    ________________________________________________;
}
```

---

b) You have been given access to a version of C that does not have the `sizeof` function implemented. However, this version of C still knows how large each type is internally (and a `char` is still one byte). Implement a constant time `sizeof` operation by filling in the blank below. Note that your `sizeof` operation will take in a *variable* of the type you wish to find the size of, rather than the type itself.

```
#define sizeof(type) ______________________________________
```

---

c) For the following MIPS code, fill in the branch immediates (in decimal) AFTER pseudo-instruction replacement. Then fill out the relocation and symbol tables. Recall that the assembler will assign addresses starting at 0x0.

Write addresses in hexadecimal and do not show leading zeros.
The tables are part of an object file and only recognize TAL.

```
        .text
start:  li   $t0, 0x10000
        addi $t1, $t0, 1
        bne $a0, $t0, start → ____
        beq $a0, $t1, next  → ____
next:   la $a0, str
        jal printf
        .data
str:    .asciiz "C.A.L.L. me, maybe?"
```

Symbol Table

Symbol	Address

Relocation Table

Instruction	Address	Dependency

d) Ash Ketchum has six slots in his party, each of which can hold a single Pokémon. Additionally, Ash has access to a PC (personal computer) which holds the rest of the Pokémon he owns. Essentially, his party acts as a "cache" for accesses to the PC (the "memory").

i. Each slot in Ash's party can hold any Pokémon. What kind of cache is this analogous to? (Circle one)

Set-associative   Write-back   Fully Associative   Direct Mapped   Write-through

ii. Ash's party exploits _____________ locality but not _____________ locality.

Explain in one sentence:

---

e) Fill in the *recursive* definition of `strcmp()` below. `strcmp` returns 0 if the contents of the strings are identical, otherwise it returns the difference in the ASCII representations of the first non-matching characters (negative number if `s1 < s2` lexicographically).

```
int strcmp(char *s1, char *s2) {

     if ( __________________________________________________________ )

          return ________________________________________________;

     return (*s1 - *s2);
}
```

---

f) MIPS already has a 64-bit architecture, so it's just a matter of time before MIPS128 is released. Let's help them out with their 128-bit instruction format design. Answer the following questions *independently* based on the MIPS32 design taught in class.

i. If we doubled the opcode field size and quadruple the register field sizes, how many instructions *forward* can we reach with a single branch? Answer in IEC.

____

ii. If we quadruple the total number of I and J format instructions, what fraction of memory could we reach with a single jump instruction? Feel free to leave as a power of 2.

____

iii. If we want to keep the same number of R, I, and J instructions, what is the maximum number of registers out architecture can support? Answer in IEC.

____

Question 3: *It's a Bird… It's a Plane… It's Supermalloc!* (15 points, 30 minutes)

Suppose we are writing a program that will be dynamically allocating a LOT of different things (hw2, anyone?). We want to create an easy way to free everything we've ever dynamically allocated. Implement the following scheme:

`void **toFree` – An array that holds a pointer to every space `malloc`'ed in the program.
`int toFreeSize` – Keeps track of the length of `toFree`.
`void* supermalloc()` – Wrapper function for `malloc()`. When called, allocate the requested space, add the pointer to `toFree`, then return the pointer.
`void superfree()` – Wrapper function for `free()`. To be called once just before exiting. Frees ALL dynamically allocated memory used in program.

a) Fill in the missing code below to correctly implement this scheme. You may find the following functions useful. For this problem, assume allocations always succeed.

```
void* malloc (size_t size);
void* realloc (void* ptr, size_t size);    // works when ptr = NULL
void  free (void* ptr);                    // does nothing when ptr = NULL
```

```
#include <stdio.h>
#include <stdlib.h>

void* supermalloc(size_t size, void ***toFree, int *toFreeSize) {
    *toFree = _____________________________________________;
    void *temp = _________________________________; // allocate
    _______________________________________; // update toFree
    ____________________________________; // update toFreeSize
    return temp;
}

void superfree(void **toFree, int toFreeSize) {
    for(int i = 0; i < toFreeSize; i++)
        ____________________________________________;
    ______________________________________________;
}

int main() {
    void **freeAr = ___________________________;
    int freeArSize = __________________________;

    // acts like malloc(4*sizeof(int)), but with additional tracking features
    int *test = supermalloc(4*sizeof(int), &freeAr, &freeArSize);
    superfree(freeAr, freeArSize);
}
```

b) How many total Stack frames are created in the execution of this program? You may assume that all library functions are self-contained (do not call other functions). Note that `sizeof` is an *operator*, not a function.

____

c) What is the maximum Stack frame *depth* (in # of frames) during the execution of this program?

____

Assume we are running this program on a 32-bit machine and that `sizeof(size_t)=4`. Consider the execution right before `supermalloc()` returns (the frame still exists).

d) How many bytes are allocated in each section of memory? Assume all declared variables are stored in memory and that the space for `test` was allocated *before* the call to `supermalloc()`.

   Think CAREFULLY! What needs to be saved on the Stack? Don't worry about Stack frame alignment (assume you allocate just as much space as you need).

   For partial credit, list the names of the variables that are stored on the Stack in the box below.

Stack: ___________ **Heap:** ___________ **Static Data:** __________

Vars on Stack:

PAGE INTENTIONALLY LEFT BLANK

(Any work on this page will not be graded)

<u>Question 4</u>: *U2 Can Write MIPS in Mysterious Ways* (8 points, 16 minutes)

Answer the questions below about the following MIPS function.

```
Mystery:
      addiu $t0, $0,  0
      addiu $t1, $0,  0
      addiu $t7, $0,  32
Lbl1: addu  $t2, $a0, $t0
      addu  $t3, $a0, $t1
      lb    $s0, 0($t2)
      beq   $s0, $0,  Lbl3
      beq   $s0, $t7, Lbl2
      sb    $s0, 0($t3)
      addiu $t1, $t1, 1
Lbl2: addiu $t0, $t0, 1
      j     Lbl1
Lbl3: sb    $0,  0($t3)
      addu  $v0, $a0, $0
      jr    $ra
```

a) What variable type would `$a0` be in the corresponding C program? ________ `a0;`

b) What does the number 32 represent in the 3rd instruction shown?

____

c) Give the following labels more intuitive/functional names:

`Lbl1` ____________ `Lbl2` _____________ `Lbl3` _____________

d) Describe in ONE sentence what the MIPS function *accomplishes*. You are not allowed to use any register names, so don't describe it instruction-by-instruction.

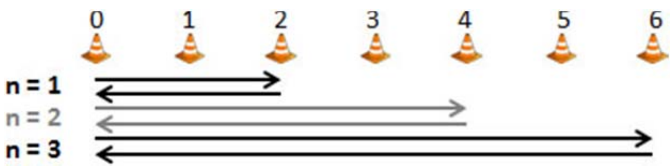
e) Your friend tells you that using the function above broke the rest of his code! In one sentence EACH, describe two DIFFERENT ways to fix `Mystery`:

Method 1: ____

Method 2: ____

Question 5: *Dying For Some Cache* (22 points, 45 minutes)

In sports, the exercise known as *suicides* is where an athlete makes successively longer sprints to and from the same starting position. The following function counts the number of times the athlete passes evenly-spaced cones (including cone 0 but not including the end cone). Each array entry `cones[i]` is the `i`-th cone from the start and the athlete makes n runs, with each successive run being `stride` cones longer than the last. The example shown below is for `stride`=2 and n=3:



`Returned array:`
`cones[] = {6,6,4,4,2,2}`

```
int *run_suicides(int stride, int n) {
      int i, j, *cones = (int *) calloc(sizeof(int)*n*stride);
      for (i = 1; i <= n; i++) {
            for (j = 0; j < i*stride; j++)      cones[j]++;
            for (j = i*stride-1; j >= 0; j--)   cones[j]++;
      }
      return cones;
}
```

Assume our system has the following parameters:

- 16MiB address space
- Block size of 16B
- Cache with 8 slots: write-back and write allocate

SHOW YOUR WORK FOR PARTIAL CREDIT!!!

a) If the cache is fully associative with random replacement,

TIO Breakdown: ____:____:____

Bits to implement cache: ______________

b) If the cache is 2-way set associative with LRU replacement,

How many blocks map to each set? Answer in IEC. ______________

Minimum LRU bits for whole cache? ______________

For the following questions, assume that `calloc()` returns a *block-aligned* address and sets the allocated memory to zero in sequential order starting from `cones[0]`. Our cache is 2-way set associative with LRU replacement. Consider ONLY the hit and miss rates for the loops (lines 3-6).

SHOW YOUR WORK FOR PARTIAL CREDIT!!!

c) If n=1, what is the minimum miss rate?

____

d) If n=1, what is the maximum `stride` value before the hit rate drops below its maximum?

____

e) Explain in 1-2 sentences how switching to no-write allocate affects your answer to part (c):

f) If n=2 and `stride=32`, what is the miss rate?

____

g) Consider each of the changes listed below independently. Circle the one(s) that would DECREASE the miss rate in part (f):

Halve n	Double `stride`	Decrease associativity
Halve cache size	Double block size (same cache size)	Write-through policy

PAGE INTENTIONALLY LEFT BLANK

(Any work on this page will not be graded)

<u>Question 6</u>: *AMATter of Performance* (12 points, 25 minutes)

We wish to implement the following function, which returns the name of the last node in a linked list:

```
struct node {
   char *name;
   struct node *next;
};
```

```
// returns name of last node in linked list
// assume head != NULL
char *lastNodeName(struct node *head) {
   while (head->next)
     head = head->next;
   return head->name;
}
```

a) We compile the function into True Assembly Language (TAL). Complete the implementation below. You are not allowed to introduce any additional labels.

```
lastNodeName:
        lw   ______________________
        beq  ______________________
        lw   ______________________
        j    ______________________
Ret:    lw   ______________________
        jr   $ra
```

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

- **Unified** L1$ with a hit time of 3 cycles and a hit rate of 90%
- Miss Penalty to main memory of 100 cycles
- Base CPI of 5 (in the absence of cache misses)

b) Calculate our machine's AMAT:

____

c) We decide to add a L2$ to reduce our AMAT to 5. We know the global miss rate is 1%. What's the worst L2$ Hit Time that will still meet our AMAT goal?

____

d) Back to only L1$: what is the CPI_{stall} for `lastNodeName` if it is called on a linked list of length N?

e) In 1 sentence, describe a 1-line change to the code in part (a) that would decrease our CPI_{stall}:

BACK OF EXAM

(Any work on this page will not be graded)