CS61C Summer 2013 Final Exam

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 90 points and will count for 26% 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 for scratch space.

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	23 (48)	
2	9 (20)	
3	15 (30)	
4	12 (20)	
5	17 (36)	
6	13 (26)	
Total	**90 (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* (23 points, 48 minutes)

a) MOESI on Through This Problem

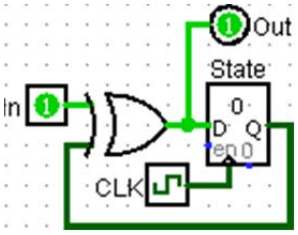
Our computer has two cores, each with a 32B direct-mapped cache with 16B blocks using write-back and write-allocate policies. The MOESI protocol is implemented with invalidation of other caches on write, and the caches are empty at the beginning of the program. `arr` is a block-aligned array of `int`s.

Fill out the status of the blocks in each cache. Indicate any memory locations that are not up-to-date as well. The first two rows have been done for you, using the abbreviations "C" for cache and "B" for block.

After Operation	Core 1 $ State	Core 2 $ State	Out-of-date Mem Locations
C1: read from `arr[0]`	B0: Exclusive B?: Invalid	B?: Invalid B?: Invalid	none
C2: write to `arr[5]`	B0: Exclusive B?: Invalid	B1: Modified B?: Invalid	`arr[5]`
C2: write to `arr[2]`			
C1: read from `arr[3]`			

b) Answer the following questions based on the FSM circuit shown below:

i. Draw the FSM state diagram (assume the initial state shown) in the space below:



ii. Let $t_{\text{setup}} = t_{\text{hold}} = 50$ ps and $t_{\text{XOR}} = 20$ ps. If we run this FSM on a 4-GHz processor and the input arrives t_{hold} after the clock triggers, what are the maximum and minimum $t_{\text{clk-to-q}}$ for the register to ensure proper functionality?

Min: _____ Max: _____

c) Implement the following truth table functions using only NOR gates (fewer is better)

A	B	F1	F2	F3
0	0	1	0	1
0	1	1	1	0
1	0	0	1	1
1	1	0	1	1

F1:

F2:

F3:

d) Hamm It Up

i. What's the correct data word given the following SEC Hamming code: `0101000`?

_____

You are analyzing a new error detection/correction scheme for 4 bits of data ($d_1d_2d_3d_4$), where you store *two* parity bits ($d_1d_2d_3d_4p_1p_2$) with p_1 = xor(d_1,d_2,d_3,d_4) and p_2 = xor(d_1,d_2,d_3,d_4,p_1).
For example, if the data bits were 1011, the code word would be 101110.

ii. What fraction of all code words is valid? _____

iii. What is the minimum Hamming distance between valid code words? _____

iv. What is the *maximum* number of errors we can *detect*? _____

e) There's Loot to be Had… Let's RAID It!

You are using RAID with 8 equally-sized disks and block-striping in 32-bit chunks.

i. Which RAID level(s) can you *not* use? _____

ii. With which viable RAID level(s) can you to store the *most* data? _____

iii. With which viable RAID level(s) can you store the *least* data? _____

iv. If using RAID 5, how many disk reads and writes are there for writing 8 bytes of data within the same stripe?

Rd: _____ Wr: _____

f) Solve for the *maximum controller overhead* to meet the following specifications:

We need disk latency under 18 ms while reading 800 B of data. The hard drive spins at 6000 rev/min with a seek time of 2.5 ms and transfer rate of 80 KB/s (SI prefix). Don't forget units!

_____

PAGE INTENTIONALLY LEFT BLANK

(Any work on this page will not be graded)

Question 2: *MIPStifying* (9 points, 20 minutes)

Answer the questions below about the following MIPS function. Answer each part separately, assuming each time that `mystery()` has not been called yet.

```
      mystery:
1           andi  $a0, $a0, 3
2           ori   $t0, $0,  1
3           sll   $t0, $t0, 6
4     Lbl1: beq   $a0, $0,  Lbl2
5           sll   $t0, $t0, 5
6           addi  $a0, $a0, -1
7           j     Lbl1
8     Lbl2: la    $s0, Lbl3
8           lw    $s1, 0($s0)
9           add   $s1, $s1, $t0
10          sw    $s1, 0($s0)
11    Lbl3: add   $v0, $0,  $0
12          jr    $ra
```

a) Which instruction (number) gets modified in the above function? _____

b) Write an equivalent *arithmetic* (not logical) C expression to instruction 1. `a0` = _____

c) Which instruction field gets modified when mystery is called with `$a0 = 3`? _____

d) How many times can `mystery(2)` be called before the behavior of `mystery()` changes?

_____

e) How many times can `mystery(0)` be called before the behavior of `mystery()` changes?

_____

f) A program calls `mystery` with the following sequence of arguments: 0, 1, 2, 3, 4, 5.
What MIPS instruction gets stored in memory?

_____

Question 3: *To Be Without Parallel… Means You're Slow* (15 points, 30 minutes)

a) SIMD and OpenMP

Four CEOs are playing the board game Monopoly, where the object of the game is to own properties and gain profits from them. You are in charge of keeping track of the CEOs' finances and wish to parallelize this task. All memory accesses are valid. Assume `sizeof(int) = 4`.

```
int balance[4];            // global array of balances
Property props[NUMPROPS];  // global array of properties
```

A property is defined by the following struct:

```
typedef struct {
    int owner;    // the index of the CEO who owns this property
    int profit;   // collectable money
} Property;
```

Every round we must give each CEO the amount of money that he has earned from each of his properties. To do this, we add the property's profit into the CEO's balance and then set the profit to zero for the next round using the following function:

```
void collect_profits() {

    for(int i = 0; i < NUMPROPS; i++) {
        balance[props[i].owner] += props[i].profit;
        props[i].profit = 0;

        _____________________________________________________________________;

        _____________________________________________________________________;
    }
}
```

i. Perform a 2-fold unrolling of the loop by filling in the blank spaces above. You may edit the looping conditions if you need to (cross out and write in changes). Assume `NUMPROPS` is a multiple of 2.

ii. `_mm_loadu_si128` loads 128-bits of data into a vector. If we wish to use **three** of these instructions to load from `props[]` every iteration of our loop, how many fold must we unroll the original loop? Answer n if performing an n-fold unrolling. _____

iii. You slap a `#pragma omp parallel for` statement on the original `for` loop. Circle the effect on execution below and provide a one phrase/sentence explanation. No credit without explanation.

Correct; Faster | Correct; Slower | Almost always Incorrect | Segfault

Explanation:

iv. Now you additionally add a `#pragma omp critical` statement around all writes to `balance[]`. What is the new effect on execution? No credit without explanation.

Correct; Faster | Correct; Slower | Almost always Incorrect | Segfault

Explanation:

b) MapReduce

Suppose that given a large social network dataset, you wish to generate recommendations for yourself by looking at the "Likes" of your friends. You wish to exclude your own likes so that the recommendations are useful. Unfortunately, your input dataset consists of everyone on the social network, not just your friends. Assume that you have access to a global person_id for yourself, **YOUR_ID**.

Conceptually, you can think of the `map` as filtering the overall dataset to just you and your friends, while the `reduce` will filter out common "Likes" between you and your friends.

You have access to the following special methods:

```
list1 = removeAll(list1, list2)   // list1 now contains only elements that were in list1
                                  // but not in list2

list_ret = sortOnValues1(list_arg)   // assuming each element in list is a tuple, returns
                                     // copy of list_arg sorted on the 1st element of the tuple
```

Feel free to access members of lists and tuples using **array syntax**.

i. The input to the `map` function is (key = person_id, value = (friend_list, likes)). How many times will a friendship between two people show up in the input data? _____

ii. Fill in the MapReduce functions below using Java-like pseudocode:

```
map(key, value){
    friend_list = value[0];
    likes = value[1];
    person_id = key;
    for (friend: friend_list) {
        if (person_id == YOUR_ID) {

            emit (( YOUR_ID, ____________________), (0, likes));

        } else if (_____________________________________) {

            emit ((____________________, ____________________), (1, likes));
        }
    }
}
reduce(key, values){

    emitValues = ____________________________________________________________;

    emit(key, removeAll(_____________________, _____________________));
}
```

Your output will be of the form:

key = (friendpair$_i$), value = recommendations

PAGE INTENTIONALLY LEFT BLANK

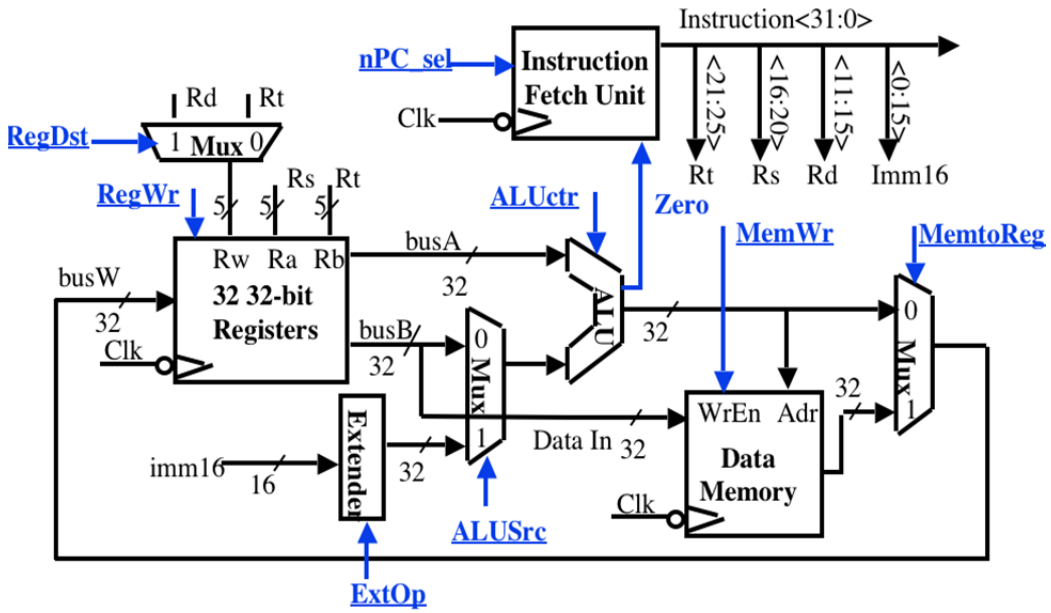
(Any work on this page will not be graded)

Question 4: *Off the Beaten Datapath* (12 points, 20 minutes)

Add the instruction `stones` (store one smaller) to the single cycle datapath, which stores a 1 at the address of the smaller value between two specified registers. **Ignore pipelining.**

That is,

```
unsigned x, y;
char *p = NULL;
if (x < y)
   *(p+x) = 1;
else
   *(p+y) = 1;
```



a) Write out the assembly syntax and RTL for this instruction. Don't forget about the PC!

Syntax: **RTL:**

b) Change as *little as possible* in the datapath above **(draw your changes on the figure)** to enable `stones`. List all your changes below (be concise!). Your modification may use MUXes (define what select bits refer to what inputs), wires, constants, and up to one new control signal, but nothing else. You may not need all of the provided boxes. You cannot modify the ALU (there is no `min` operation).

(i)	
(ii)	
(iii)	
(iv)	

c) We now want to set all the control lines appropriately. List what each signal should be, using 0, 1, X, or an intuitive name. Include any new control signals you added.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	

Question 5: *Tread Carefully, Thread Carefully* (17 points, 36 minutes)

Examine the function prototype and MIPS implementation below.

```
// sets *value = (*value) * 2^pow using shifting instructions
int multMemPow2(int *value, unsigned int pow);
```

```
      multMemPow2:
1           lw   $v0, 0($a0)    # load value
2     loop: beq  $a1, $0, exit  # exit condition
3           sll  $v0, $v0, 1    # multiply by 2
4           addi $a1, $a1, -1   # decrement counter
5           sw   $v0, 0($a0)    # store result
6           j    loop
7     exit: jr   $ra
```

We are using a 5-stage MIPS pipelined datapath with separate I$ and D$ that can read and write to registers in a single cycle. Assume no other optimizations (no forwarding, no branch prediction, etc.). The default behavior is to stall when necessary. Branch checking is done during the Execute stage.

For parts (a)-(c), let pow=1. When we ask for clock cycles to execute `multMemPow2`, we mean from the instruction fetch of `lw` up to and including the write back of `jr`.

a) How many instructions are executed in `multMemPow2`?

_____

b) How many clock cycles does it take to execute `multMemPow2`?

(we will assign partial credit based on table on opposite page) _____

c) Consider the following optimizations *separately*. How many FEWER cycles are taken for the addition of each optimization?

   i. Forwarding _____

   ii. Branch Prediction of always taken _____

d) Suppose we introduce only jump delay slots and want to move a loop instruction into the new jump delay slot after instruction 6 (`j loop`). For the following candidate instructions, answer **C** for "changes behavior," **S** for "causes additional stall(s)," or **G** for "good choice":

   Instr 3: _____ Instr 4: _____ Instr 5: _____

For the following questions, assume we are executing the `multMemPow2` simultaneously on TWO processors in the same shared-memory machine with **`*value=1 and pow=2`**.

e) List ALL possible values of `*value` after execution: _____

Stanley Stanfurd thinks he can fix the data race problem by replacing the `lw` with `ll` and `sw` with `sc`. Assume `sc` compares against the value from the last `ll` call.

f) List ALL possible values of `*value` after execution of this new version: _____

You may find the following space useful for scratch work; we will check for assigning partial credit:

PAGE INTENTIONALLY LEFT BLANK

(Any work on this page will not be graded)

Question 6: *It's Virtual Insanity!* (13 points, 26 minutes)

Our 32-bit uniprocessor machine has 1 GiB of RAM with 1 KiB pages, a fully-associative TLB that holds 8 entries and uses LRU, and a direct-mapped, write-back *data* cache with 32 B blocks and 32 slots. The *instruction* cache is 256 B and fully-associative with 32 B blocks.

a) What is the maximum number of valid entries in the page table for a single process? Answer in IEC.

_____

b) What is the TLB Reach of our system? _____

Examine the following function. Assume the entire program's code takes the entirety of one page and `sizeof(int)=sizeof(int *)=4`.

```
void addConst(int *ptr, char c) {
    for(int i = 0; 1; i+=4)
        ptr[i] += c;
}
```

c) If `ptr[]` lives in disk and `ptr[0]` is page-aligned, what is the TLB hit rate for data accesses only?

_____

d) If `ptr[]` lives in disk and `ptr[0]` is page-aligned, what fraction of D$ misses are also TLB misses?

_____

e) If `ptr[0]` is in physical memory, what is the *minimum* value of `i` that could cause a **page fault**?

_____

f) If `ptr[0]` is in physical memory, what is the *minimum* value of `i` that could cause a **protection fault**?

_____

g) If `ptr[0]` is in physical memory, what is the *maximum* value of `i` that causes the first **cache miss** in the loop?

_____

h) If `ptr[0]` is in physical memory, what is the *maximum* value of i that causes the first **TLB miss** in the loop? You may leave your answer as a product.

_____

BACK OF EXAM

(Any work on this page will not be graded)