

# CS61C Summer 2012 Final

Your Name: Peter Perfect SID: 0000000

Your TA (Circle): Raphael Paul Brandon Sung Roa

Name of person to your LEFT: Larry Learner

Name of person to your RIGHT: Stephen Student

This exam is worth 92 points and will count for 25% of your course grade.

The exam contains 8 questions on 14 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	25 (46)	
2	10 (22)	
3	7 (14)	
4	5 (12)	
5	10 (20)	
6	14 (26)	
7	10 (20)	
8	10 (20)	
Total	92 (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...* (25 points, 46 minutes)

a) (7 points) True/False:

- T  F Thanks to virtual memory, processes can share the physical memory space. Unfortunately, this means that writing off the end of an array might overwrite data of another process.
  - T  F High reliability guarantees high availability.
  - T F PUE does not measure the power efficiency of Warehouse Scale Computer servers
  - T  F Amdahl's law is restricting the speed increase predicted by Moore's Law
  - T  F We can implement the atomic test-and-set operation in MIPS using `lw` and `sw`.
  - T F The CPI of superscalar processors can be less than one.
  - T F With a 4-entry TLB, 2 physical pages, and 4 virtual pages, the TLB will never be full.
- 

b) (3 points) You are running a service that helps users meet others with similar tastes in food. For each pair of users with at least one food in common, we would like to count how many foods they have in common. We'll do so by chaining two MapReduce jobs together; in other words, the output of the first is the input to the second.

Your input is a list of key value pairs of the form `(user_id, list of foods)`. Given the first map and the second reduce, define in pseudocode `Reduce1` and `Map2`. The final output should be a list of `((user_id1, user_id2), count)` where `user_id1 < user_id2`.

```
Map1(key, value):
    For v in values:
        Emit(v, key)
Reduce1(key, values):
    For id1 in values:
        For id2 in values:
            If id1 < id2:
                Emit((id1, id2), 1)
Map2(key, value):
    Emit(key, value)
Reduce2(key, values):
    For each v in values:
        sum += v;
    Emit(key, sum)
```

1 pt for creating user pairs, 1 pt for `user_id1 > user_id2`, 1 pt for producing right sum (outputting 1)  
-1 pt if assuming multiple keys available in `Reduce1`. There are other variations like emitting the pairs of users in `Map2`.

---

c) (2 points) What's the correct data word given the following SEC Hamming code: 1110011?

There is error in parity bit 1, but that's it. 1 pt if gave code word. **1011**

Your friend thinks he has a better error correction/detection scheme: for  $m$ -bits of data, a copy of those  $m$ -bits will be stored. For example, if the data bits were 1110, the code word would be 11101110.

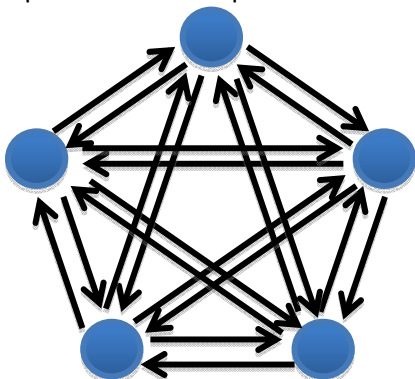
- d) (1 point) What fraction of the code words is valid? Assume  $m$ -bit data words.      $1/(2^m)$
- e) (1 point) What is the minimum Hamming distance between valid code words?     2
- f) (1 point) Consider each of the following independently. Circle all that this scheme can accomplish.
- |                                              |                                              |
|----------------------------------------------|----------------------------------------------|
| i. Detect any single bit error.              | iv. Correct any single bit error.            |
| ii. Detect any double bit errors.            | v. Correct any double bit errors.            |
| iii. Detect any errors of $m$ -bits or less. | vi. Correct any errors of $m$ -bits or less. |

There's a single disk failure in a disk array (you know which disk failed) and you want to read a single page. Assume that for block-striped arrays, the page is contained within a single block.

- g) (2 point) What's the fewest number of disks you have to read from if the array were:
- |                         |                  |                                     |
|-------------------------|------------------|-------------------------------------|
| i. RAID 1 with 2 disks  | <u>    1    </u> | Data is NOT in the disk that failed |
| ii. RAID 5 with 4 disks | <u>    1    </u> |                                     |
- h) (2 point) What's the greatest number of disks you have to read from if the array were:
- |                                           |                  |                                                            |
|-------------------------------------------|------------------|------------------------------------------------------------|
| i. RAID 4 with 4 disks (including parity) | <u>    3    </u> | Data IS in the disk that failed and needs to be recovered. |
| ii. RAID 5 with 4 disks                   | <u>    3    </u> |                                                            |

Assume we have a 1 GHz processor using 4 KiB pages and an attached hard drive spinning at 6000 revolutions per minute, with a seek time of 3 ms, an average page throughput of 0.4 MB/s (remember, data rates use SI prefixes), and negligible controller overhead.

- i) (2 point) What is the transfer time for a page?
- Throughput for page yields 10.24ms disk latency; avg rotation time is 5ms  
1 pt only for 10.24 ms.     2.24ms
- j) (2 points) Assume the processor polls the disk. Polling operation takes 100 clock cycles and the disk transfers data in 4 B chunks. What percentage of processor time is spent polling?
- 0.4 MB/s  $\rightarrow$   $10^5$  polls/s  $\rightarrow$   $10^7$  clock cycles/s     1%
- k) (2 points) In a 5-state FSM with 4 arrows coming out of each state, as shown below, how many possible 1-bit output functions are there? Answer in IEC format.



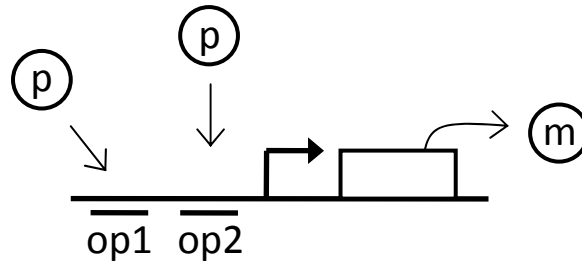
Need  $\lceil \log_2(5) \rceil = 3$  bits for CS  
and  $\lceil \log_2(4) \rceil = 2$  bits for In,  
So total  $2^{(2^5)} = 2^{32}$  functions.  
1 Mebi ( $2^{20}$ ) also accepted (no rows of X's).  
1 point if just have 32 or 20.  
-0.5 for not using IEC.

    4Gibi funcs

**Question 2:** *Biological STATE-ment* (10 points, 22 minutes)

You didn't think you'd get through this test without seeing something bio-related, did you?

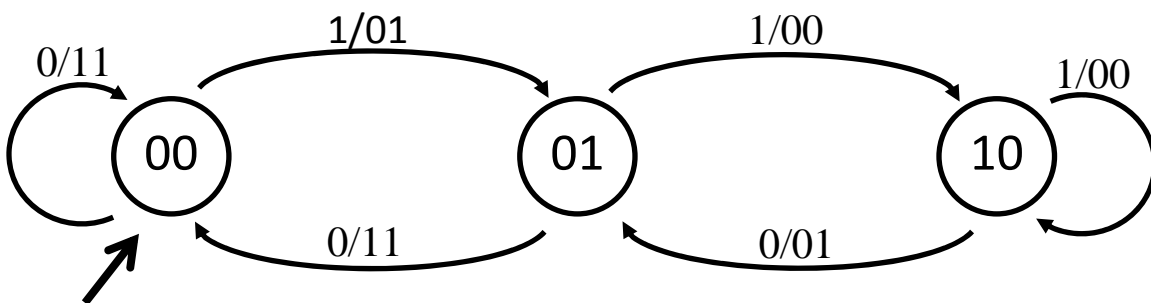
The following diagram shows a DNA construct known as a *promoter*. Proteins (*p*) interact with the promoter by binding and unbinding to its two operating sites (*op1* and *op2*) and affecting how much mRNA (*m*) is being produced. A protein can only bind if there is an open operating site and a protein can only unbind if it was previously bound. Here we assume that *p* acts as a *repressor*, that is, the more *p* are bound to the promoter, the less *m* is produced.



The following FSM represents the behavior of this promoter under the assumption that a protein cannot bind to *op2* unless there is already a protein bound to *op1* and that a protein bound to *op1* cannot unbind while a protein is bound to *op2*.

In = whether a protein attempts to bind (1) or a protein attempts to unbind (0)

Out (2 bits) = amount of mRNA produced when no proteins bound (3), 1 protein bound (1), or 2 proteins bound (0)



a) Provide more intuitive descriptions of the states: (1 point)

State 00: **No proteins bound** \_\_\_\_\_

State 01: **One protein bound** \_\_\_\_\_

State 10: **Two proteins bound** \_\_\_\_\_

b) Fill in the blanks in the Truth Table below. (2 points)

CS1	CS0	In	NS1	NS0	Out1	Out0
0	0	0	0	0	1	1
0	0	1	0	1	0	1
0	1	0	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	0	1
1	0	1	1	0	0	0
1	1	0	X	X	X	X
1	1	1	X	X	X	X

c) Provide the *most simplified* Boolean expression possible for Out0: (4 points)

SOP if 2<sup>nd</sup> to last row is 1:  $Out0 = CS1'CS0'In' + CS1'CS0'In + CS1'CS0In' + CS1CS0'In' + CS1CS0In' = In'(CS1'CS0' + CS1'CS0 + CS1CS0' + CS1CS0) + CS1'CS0'In = In' + CS1'CS0'In = In' + CS1'CS0'In + CS1'CS0'In = In' + CS1'CS0'$

Inversion trick + SOP if last row is 0:  $Out0' = In(CS1 + CS0) \rightarrow Out0 = (In(CS1 + CS0))' = In' + CS1'CS0'$

POS if last row is 0:  $(CS1 + CS0' + In')(CS1' + CS0 + In')(CS1' + CS0' + In') = In' + CS1'CS0'$

0 pt for incorrect, 1 pt for correct expression w/o simplification, 2 pt for partial simplification, 3 pt for off by 1 variable, and 4 pt for simplest expression

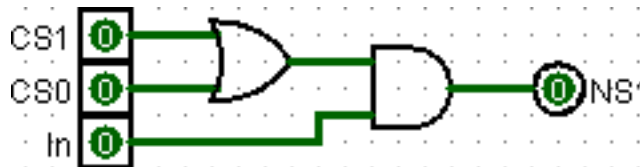
d) For the provided column NS1, draw out the combinational logic using the *fewest* number of gates possible, assuming the “don’t cares” are taken to be 0 and 1 from top to bottom: (2 point)

$CS1'CS0In + CS1CS0'In + CS1CS0In$

$CS'CS0In + CS1In$

$In (CS1 + CS1'CS0)$

$In (CS1 + CS0)$



-1 for each additional gate used

e) Provide one set of values for the “don’t cares” in NS1 that would *guarantee* that the FSM doesn’t stay stuck in the 4<sup>th</sup> state even if it enters it by some glitch. (1 point)

CS1	CS0	In	NS1
1	1	0	0
1	1	1	0

CS = 0b11 is now forced to transition either to 0b01 or 0b00, which are valid states.

### **Question 3:** *Mystery* (7 points, 14 minutes)

Mystery:

```
    la    $t0, L2
    lw    $t1, 8($t0)
    addi  $t2, $0, 1
    sll   $t2, $t2, 16
    add   $t3, $0, $0
    add   $v0, $0, $0
    addi  $t5, $0, 4
    sll   $t5, $t5, 16
L1:    beq  $t3, $t5, L3
        addu $t4, $t1, $t3
L2:    addu $t3, $t3, $t2
        sw   $t4, 8($t0)
        addu $v0, $v0, $a0
        j    L1
L3:    sw   $t1, 8($t0)
        jr   $ra
```

a) Which instruction gets modified during this function call? (1 point)

**addu \$v0, \$v0, \$a0**

0.5 for getting the address of this line

0.5 for saying the right instruction but adding wrong label/other wrong info

b) How many times does the line at label L2 get executed? (2 points)

**4**

c) Describe in a sentence or two what this function does. (4 points)

**Returns the sum of \$a0 to \$a3 in \$v0**

0 for wrong answer

1 for explaining partially how the instruction runs, with no incorrect statements (more than just saying change the line)

1 for explaining exactly what happens with the modification of the function, with wrong number to increment by

2 for explaining exactly what happens with the modification of the function

2 for pointing out the arguments but not making the connection of adding them together, or only having some of the arguments

3 for having sum of arguments but adding something wrong afterwards

4 for correct answer

**Question 4:** *MOESI On Through This Problem* (5 points, 12 minutes)

Fill out the empty fields corresponding to the state of the system.

Operation column: the fields can be {{P1,P2} {reads, writes} {Mem1,Mem2}}

Processor {1,2} columns: MOESI state with corresponding block in cache, if applicable

Snoop bus signal(s): {Px.request(n), Px.send(n), Px.inv(n)} or a combination of them

**Px.request(n)** means processor Px is asking for data memory n

**Px.inv(n)** means processor Px invalidates all other copies of datablock n

**Px.send(n)** means Px is sending datablock n (in response to a request).

System characteristics: Write-back, write-allocate, invalidate others on write, Exclusive responds to read requests, 2 processors, 2-block memory, 1-block cache.

Operation	Processor 1	Processor 2	Snoop Bus
P1 reads Mem1	Exclusive(1)	Invalid	P1.request(1)
P2 reads Mem1	Owned(1)	Shared(1)	P2.request(1) P1.send(1)
P2 write Mem1	Invalid	Modified(1)	P2.inv(1)
P1 write Mem2	Modified(2)	Modified(1)	P1.request(2) P1.inv(2)
P2 read Mem2	Owned(2)	Shared(2)	P2.request(2) P1.send(2)
P2 read Mem1	Owned(2)	Exclusive(1)	P2.request(1)
P2 write Mem2	Invalid	Modified(2)	P2.request(2) P1.send(2) P2.inv(2)

**0.5 pt for each correct statement (-0.5 pt for each additional statement)**

**Question 5:** *This Problem Will BLOW Your Mine!* (10 points, 20 minutes)

One of your staff members loves the game of Minesweeper so much that he’s going to build special hardware dedicated to playing this game. In Minesweeper, there is a rectangular grid of cells (the minefield), under which lie a finite number of mines. **Each cell contains either a mine or the number of mines in the adjacent EIGHT cells (touching on corners counts; 0 is shown as a blank)**. The goal of the game is to uncover the whole field without clicking on/exploding a mine. See the images below:



Start of game: Blank field



Player clicked a mine and lost



Player uncovered field and won

Our goal is to design special hardware to facilitate high performance for setting up the game (generating the field).

- a) We need to store the value of each cell in memory. How many bits are needed at minimum for each cell? **(1 pt)** Numbers 0-8 + mine = 10 things 4

- b) Assume there is a helper function to randomly generate the locations of the mines. Name TWO SEPARATE REASONS why using an `int` array here is preferable to a linked list. **(0.5 pt each)**

Reason 1: **Less memory usage** \_\_\_\_\_

Reason 2: **Faster access: better spatial locality and O(1) array access to specific element** \_\_\_\_\_

**This problem was unfortunately ambiguous as to whether it referred to the mine locations or the mine field, so credit was given to responses referring to SIMD, accessing adjacent cells, etc.**

The procedure for generating the field is as follows. We start with an appropriately-sized array of all zeros. For each mine location, we add 1 to each adjacent cell and a relatively large number (say, 10) to the mine location itself. This way we know that any cell with value  $\geq 10$  contains a mine.

- c) We are designing special hardware that acts like SIMD, but we will optimize the register size for Minesweeper. Given that an `int` is 32 bits, what size should our special “xmm” registers be?

**We need to update 9 cells total (either grouped in 3 rows of 3 entries or 3 columns of 3 entries).**

**Therefore we want a register size of 3 so we don’t have to deal with padding. (1 pt)**

**0.5 pt given for  $32 \cdot 9 = 288$  or  $3 \cdot 4$  (referring to part a).** 96 bits



- d) Assume our new special registers have been typedef-ed so that you can use the variable type `xmm` and that all the SIMD functions have been adjusted to work with the new register size. Only use the SIMD functions provided for you in the `#define` directives. Complete the following function to generate the minefield. **IGNORE EDGE CASES.**

```
#define load  _mm_loadu_ps
#define store _mm_storeu_ps
#define add   _mm_add_ps

/* Field: m by n row-major array stored at mineField */
/* N mines stored in 2*N int array mineLocs (0-indexed coordinates) */
void genField(int *mineLocs, int *mineField, int m, int n, int N) {
    int i, x, y;
    xmm reg1 = _mm_set_ps(1,1,1);
    xmm reg2 = _mm_set_ps(1,10,1);
    xmm reg3;
    for (i = 0; i < N; i++) {
        x = mineLocs[2*i];
        y = mineLocs[2*i+1]_____ ;
        int *mine = mineField + n*y + x; _____
        reg3 = load(mine-1); _____
        store(base, add(reg3, reg2)); _____
        reg3 = load(mine-1-n); _____
        store(mine-1-n, add(reg3, reg1)); _____
        reg3 = load(mine-1+n); _____
        store(mine-1+n, add(reg3, reg1)); _____
    }
}
```

**Key points:** (7 pt total)

For coordinates (notice `x` pulls every *other* element), need `y = mineLocs[2*i+1]`.

Store/load take *addresses* as arguments, not data (so `&mineLocs[mine-1]` okay, but not `mineLocs[mine-1]`).

Recognize that 3 separate load/add/stores required (one with `reg2`, two with `reg1`).

Row-major array (`n*y + x`, not `m*x + y`).

SIMD instructions act on continuous addresses *starting* at the given address, not around it (so need `-1` in address).

**Question 6: Watch Your (Time) Step!** (14 points, 26 minutes)

Consider the following difference equation (i.e. differential equation that is discretized in time):

$$x[n+1] - x[n] = -\alpha \cdot x[n]$$

$$x[n+1] = (1-\alpha) \cdot x[n]$$

Here we restrict ourselves to integer values of alpha and use the following integration function:

```
# $a0 -> addr of array to store x[i] (assume x[0] is already set)
# $a1 -> length of array/integration
# $a2 -> alpha
integrate:
1      beq    $a1,$0,exit
2      sub    $t0,$0,$a2    # $t0 = -alpha
3      addi   $t0,$t0,1     # $t0 = 1-alpha
4      lw     $t1,0($a0)    # $t1 = x[n]
5      mult  $t0,$t1
6      mflo  $t1            # $t1 = (1-alpha)*x[n]
7      sw    $t1,4($a0)    # x[n+1] = (1-alpha)*x[n]
8      addiu $a0,$a0,4
9      addiu $a1,$a1,-1
10     j     integrate
11    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, etc.). The default behavior is to stall when necessary. Multiplication and branch checking are done during EX and the HI and LO registers are read during ID and written during WB.

- a) As a reminder,  $T_c$  stands for “time between completions of instructions.” Given the following datapath stage times, what is the ratio  $T_{c, \text{single-cycle}}/T_{c, \text{pipelined}}$ ? **1000/400=5/2**  
1pt all of nothing

IF	ID	EX	MEM	WB
200 ps	100 ps	400 ps	200 ps	100 ps

- b) Count the number of the different types of potential hazards found in the code above:

Structural:   0                    Data:   6                    Control:   3    
None                                  @ 2-3, 3-5, 4-5,                  @ 1, 10, 11  
                                         5-6, 6-7, 9-1

1 pt for Structural  
1 pt for 3 Control or 0.5 for 2 Control  
3 pts for Data: 0.5 pt for each

For the following questions, examine a SINGLE ITERATION of the loop (do not consider the `jr`).

- c) With no optimizations, how many clock cycles does our 5-stage pipelined datapath take in one loop iteration (end your count exactly on completion of `jr`)? 24

= 10 instructions + 4 for draining pipeline + 2\*(# data hazards - 2) + 2 for beq control hazard.

Beq hazard is 2 stalls because branch comparison done in EX stage.

# data hazards minus 2 because 3-5 and 9-1 hazards taken care of by other stalls.

2pts - Answer depends on data hazards in (b).

- d) How many clock cycles LESS would be taken if we had FORWARDING? 7

= 2\*(# of data hazards - 1) - 1 because of load hazard

Again, you can ignore the 3-5 hazard here.

2pts - Answer depends on data hazards in (b).

- e) Assuming that we introduce both FORWARDING and DELAY SLOTS, describe independent changes to `integrate` that will reduce the total clock cycles taken. Changes include replacing or moving instructions. You may not need all spaces given.

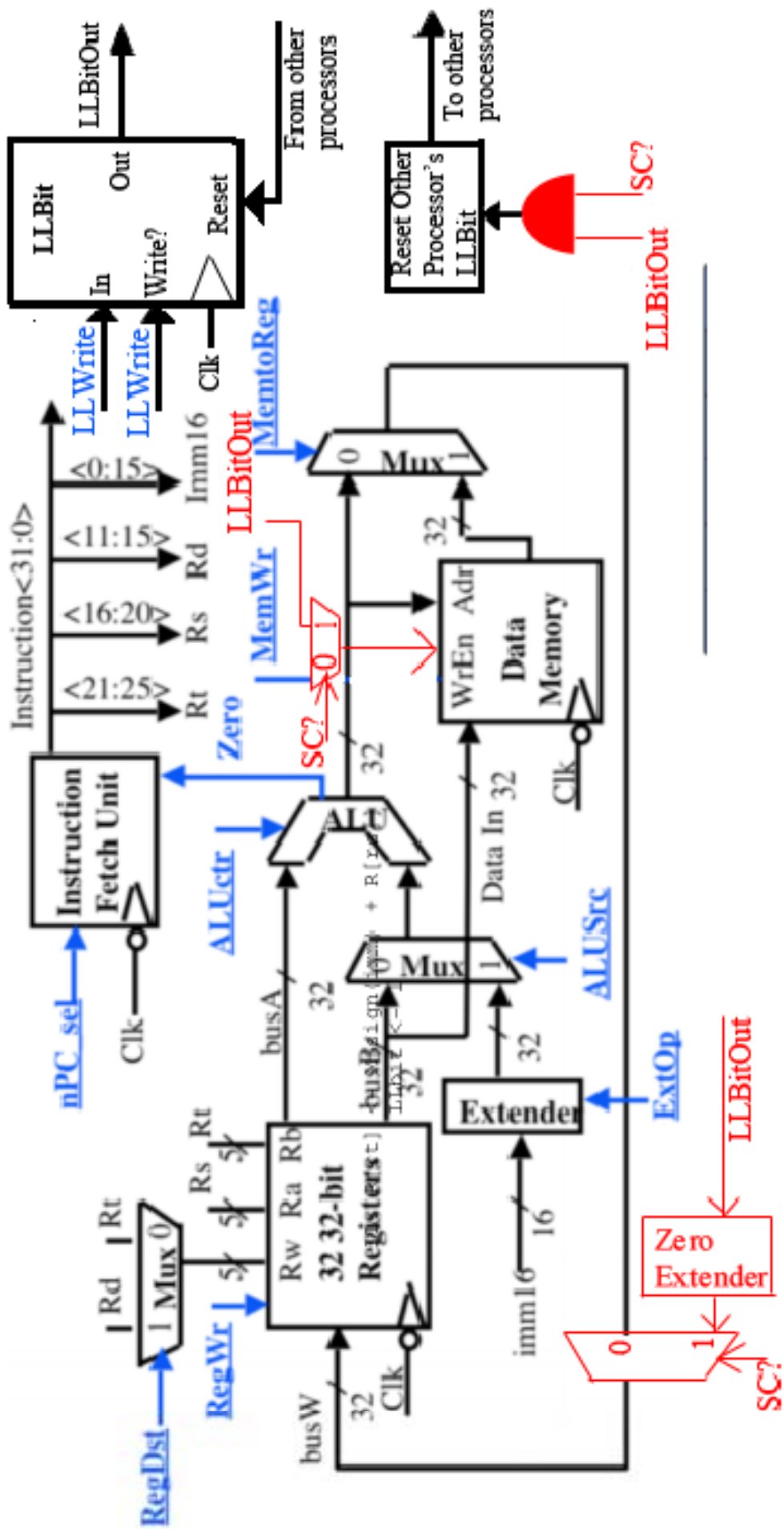
Instr	Change
<u>3</u>	Move after 4 (or move instr 4 before instr 3)
<u>8/9</u>	Move after 10
Also accepted:	
<u>9</u>	Move after 4
<u>8</u>	Move after 10

2 pts for filling load-use stall

2 pts for filling jump delayed-branch slot

-0.5pt for unnecessary changes

-1pt for changes that changed what the code does



**Question 7: Conditional Wiring** (10 points, 20 minutes)

Assume our MIPS machine is implemented as a single-cycle datapath. Your job is to provide new datapath and control elements that can implement **store-conditional** instruction (the implementation of load-link is provided for you). In particular, the load-link and store-conditional should be implemented as:

```

LL: R[rt] <- M[sign(imm) + R[rs]]
    LLbit <- 1

SC: if LLbit:
    M[sign(imm) + R[rs]] <- R[rt];
    Reset other processor's LLbit
    R[rt] <- {31 zeros, LLbit}

```

**Modify** the picture on the opposite page and **list** the changes you made here (you may not need all the given lines). You cannot modify the pre-existing datapath components besides the wires:

1. **\_Need to mux the MemWr and LLBitOut into WrEn with select bit SC?** \_\_\_\_\_
2. **\_Need to AND SC? and LLBitOut before Reset Others box** \_\_\_\_\_
3. **\_Need to wire LLBitOut to a zero-extender** \_\_\_\_\_
4. **\_Need to mux the busW with zero-extended LLBitOut into the write port of register file with control signal SC?** \_\_\_\_\_
5. **\_Need control signal SC?** \_\_\_\_\_
6. \_\_\_\_\_
7. \_\_\_\_\_
8. \_\_\_\_\_

(using an AND instead of a mux (-0.5 pt))

(before reset other proc's, uses AND(LLBitOut, SC?) is OKAY)

(just saying {31 zero's, LLBitOut} instead of extender is okay)

(wiring LLBitOut right into ResetOther's (-0.5 pt))

(using a control signal as an entry, OK)

(modifying datapath structure (-0.5 pt))

(-2 pts for no drawing)

Now **state** what values of these control signals should be {0,1, X for “don’t care,” or any other intuitive names}, plus for any new signals you may have created.

	RegDst	RegWr	nPC_sel	ExtOP	ALUSrc	ALUctr
LL	rt	1	PC+4	sign	imm	add
SC	<b>rt</b>	<b>1</b>	<b>PC+4</b>	<b>sign</b>	<b>imm</b>	<b>add</b>

	MemWr	MemtoReg	LLWrite	SC?		
LL	0	1	1	<b>0</b>		
SC	<b>X</b>	<b>X</b>	<b>0</b>	<b>1</b>		

(-0.5 pt for each incorrect box)

**Question 8:** *It's Not My Fault* (10 points, 20 minutes)

Consider the following OpenMP snippet:

```
int values[size];
#pragma omp parallel
{
    int i = omp_get_thread_num();
    int n = omp_get_num_threads();
    for(int j = i * (size / n); j < (i + 1) * (size / n); j++)
    {
        values[j] = j;
    }
}
```

All cores share the same physical memory and we are running 2 threads. This is the sole process running. Each page is 1 KiB, and you have 2 pages of physical memory. The code snippet above starts at virtual address 0x400, and the values array starts at 0x800. The size, n, i, and j variables are all stored in registers. The functions `omp_get_thread_num` and `omp_get_num_threads` are stored in virtual addresses 0x440 to 0x480. The replacement policy for the page table is Least Recently Used.

At the start of the `pragma omp parallel` call the page table looks as follows:

	Virtual Page Number	Valid	Dirty	Physical Page Number
0x000	0	0	0	0
0x400	1	1	0	0
0x800	2	1	1	1
0xc00	3	0	0	1

a) How many page faults will occur if `size = 0x080`? 0

Traversing 0x200 of memory (addresses 0x800 to 0xa00), which all sit in virtual page 2, which is already Valid according to the table. Instruction accesses are also all in a Valid page.

2 pts for correct answer

b) What is the minimum number of page faults that will occur if `size = 0x200`? 1  
 What is the maximum? 512

Now traversing 0x800 of memory. With two threads and loop split evenly, best case is sequential accesses (1 thread finishes before other starts), which causes 1 page fault. Worst case is ping-ponging accesses (with instruction fetches in-between) across threads (starting with Thread 2).

2 pts for correct minimum page fault number

2 pts for correct maximum page fault number

c) How could you reduce the maximum page faults for part (b)? Choose the valid options amongst the following:

- increase virtual address space
- decrease number of threads (1.5 pts)
- increase physical address space (1.5 pts)
- increase number of threads
- use SIMD instructions (1 pt)
- add a 4-entry TLB
- change page table replacement policy to Random

-1 pt for every incorrect answer