CS61C FINAL EXAM

Last Name (Please print clearly)  |  Perfect
---|---
First Name (Please print clearly)  |  Peter
Student ID Number  |  12345678
Circle the name of your Lab TA  |  Alex  Brian  Jinglin  Nate  Reese  Steven
Exam Room  |  The Campanile
Name of the person to your: Left | Right  |  Samantha Student  Larry Learner

All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who haven’t taken it yet. (please sign)

Instructions

- This booklet contains 14 pages including this cover page. The back of each page is blank and can be used for scratch work, but will not be graded (i.e. not even scanned into Gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats, headphones, and watches. Place everything except your writing utensil(s), cheat sheet, and beverage underneath your seat.
- You have 170 minutes to complete this exam. The exam is closed book: no computers, tablets, cell phones, wearable devices, or calculators. You are allowed three pages (US Letter, double-sided) of handwritten notes.
- There may be partial credit for incomplete answers; write as much of the solution as you can.
- Please write your answers within the boxes and blanks provided within each problem!

<table>
<thead>
<tr>
<th>Question</th>
<th>MT1</th>
<th>MT2</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>8 9 9 9</td>
<td>8 9 7 10</td>
<td>8 7 8 8</td>
</tr>
</tbody>
</table>

If you have the time, feel free to doodle on this front page!
Question 1: Number Representation (8 pts)

a) Convert 0x1A into base 6. Don’t forget to indicate what base your answer is in! [1 pt]

\[0x1A = 0b1 1010 = 16 + 8 + 2 = 26 = 4 \times 6^1 + 2 \times 6^0\]

\[= 42_6\]

b) In IEEE 754 floating point, how many numbers can we represent in the interval \([10,16)\)? You may leave your answer in powers of 2. [3 pts]

\[10 = 0b1010 = 1.01 \times 2^3 \text{ and } 16 = 0b10000 = 1.0 \times 2^4\]

Count all numbers with Exponent of \(2^3\) and Mantissa bits of the form \{1b’0, 1b’1, \(21\{1b’X\}\}\) and \{1b’1, \(22\{1b’X\}\}\}, for a total of \(2^{21} + 2^{22}\) numbers.

\[2^{22} + 2^{21} = 3 \times 2^{21}\]

c) If we use 7 Exponent bits, a denorm exponent of -62, and 24 Mantissa bits in floating point, what is the largest positive power of 2 that we can multiply with 1 to get underflow? [2 pts]

Smallest denorm is \(2^{-62} \times 0.0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001 = 2^{-86}\), which is representable. So the next smaller power of 2 is unrepresentable and causes underflow.

\[2^{-87}\]

Local phone numbers in the USA typically have 7 decimal digits, which use the symbols 0 to 9. For example, Jenny Tutone’s phone number is:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Line Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>867</td>
<td>5309</td>
</tr>
</tbody>
</table>

d) How many unique phone numbers can be encoded by this scheme? [1 pt]

\[10^7\]

e) How many bits would we need to represent a phone number if we treated it as a single 7-digit decimal? You may use \(\log()\) and \(\text{ceil}()\) in your answer and the variable \(E\) to represent the correct answer to part (d). [1 pt]

\[\text{ceil}(\log_2(E))\]
Question 2: C Programming (9 pts)

Given an integer array, we want to build a hash table that stores all the values in the array. The hash table will be represented as an array of 256 pointers to buckets. Each bucket is a struct that contains a pointer to an integer array to hold all the values in that particular bucket, as well as the length of that bucket’s array. We want to apply the mathematical transformation $x \rightarrow (x \mod 256)$ to each of the input integer values to get the index of the bucket in the hashtable array to which the input integer belongs. Finally, note that the input array passed to our hashtable building function does not have its size passed in along with it. Instead, the end of the array is marked by a zero element (similar to how strings are terminated with the null-terminating character). There are no zeroes in the array except the final one and that final zero is not considered a value in the array.

Given this description and the following struct definition, please fill out the hashtable building function:

```c
typedef struct bucket {
    int *values; // pointer to array that holds all ints in this bucket
    size_t size; // length of the values array
} bucket;

bucket **buildHashTable(int *arr) {
    // calloc zero-fills the allocated memory – i.e. makes all bucket pointers null
    bucket **table = calloc(sizeof(bucket *) * 256);

    while ( *arr ) { // loop until the end of the input array
        /* Get the index for this value. You are NOT allowed to use any arithmetic
           (+,-,*,/,%) or bitwise (<<,>>,^,|,&) operations for this purpose. */
        unsigned char index = (unsigned char) (*arr);  
        bucket *bucket;
        if ( !table[index] ) { // bucket has not been initialized yet
            // allocate space for exactly one bucket
            table[index] = malloc(sizeof(bucket));
            bucket = table[index];
            bucket->size = 0;
            bucket->values = malloc(sizeof(int));
        } else { // bucket already exists, just resize it to hold one more element
            bucket = table[index];
            // allocate space for exactly one more value
            bucket->values = realloc((void *)bucket->values, bucket->size + 1);
        }
        // insert value at the end of bucket
        bucket->values[bucket->size] = *(arr);
        // update bucket size
        bucket->size++;
        // move to next element in array
        arr++;
    }
    return table;
}
```
It's the end of the semester and the CS61C staff wants to drop each student's lowest grade. The grade for an assignment is the product of the assignment's score and the assignment's weight (as on gllookup, except with integer values). The struct definition of a linked list of assignments is given below. Based on the C implementation of the lowest_grade function given, fill in the MIPS implementation.

- In variable names, “A” is short for “assignment”
- The struct is tightly packed with sizeof(int) = sizeof(void *) = 4
- The last assignment's next pointer is NULL
- The variable lowest is set to INT_MAX before the first call to lowest_grade

<table>
<thead>
<tr>
<th>Struct:</th>
<th>Function:</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct{</td>
<td>int lowest_grade(A_list *As, int lowest) {</td>
</tr>
<tr>
<td>int score;</td>
<td>if (As) {</td>
</tr>
<tr>
<td>int weight;</td>
<td>int grade = As-&gt;score * As-&gt;weight;</td>
</tr>
<tr>
<td>struct list *next;</td>
<td>lowest = min(lowest, grade);</td>
</tr>
<tr>
<td>} list;</td>
<td>lowest = lowest_grade(As-&gt;next, lowest);</td>
</tr>
<tr>
<td>typedef struct list A_list</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>return lowest;</td>
</tr>
</tbody>
</table>

LOWEST_GRADE:  
# $a0 = As, $a1 = lowest  
addiu $sp $sp -4  # prologue [0.5 pt]  
sw $ra 0($sp)  # epilogue [0.5 pt]  
move $v0 $a1  # initialize return value [1 pt]  
beq $a0 $0 END  # return [0.5 pt]  
lw $t0 0($a0)  # get score [1 pt]  
lw $t1 4($a0)  # get weight [1 pt]  
mul $t0 $t0 $t1  # calculate score * weight [1.5 pt]  
bgt $t0 $a1 RECURSE  
move $a1 $t0  
RECURSE:  
lw $a0 8($a0)  # epilogue [1 pt]  
jal LOWEST_GRADE  
END:  
lw $ra 0($sp)  # return [0.5 pt]  
addiu $sp $sp 4  
jr $ra
**Question 4: Self-Modifying MIPS (9 pts)**

The following function `inst_nibble` overwrites one of its instructions during execution (assume we have write access to Code/Text pages). It flips the bits (0→1, 1→0) of a specified nibble (4 bits) of an instruction, numbered starting at 0 from the right, as shown below.

<table>
<thead>
<tr>
<th>bit</th>
<th>31</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>nibble numbering:</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

```assembly
inst_nibble:
1   la $v0, inst
2   lw $v0, 0($v0)
3   andi $a0, $a0, 0xF     # $a0 holds bit mask
4   sll $a1, $a1, 2        # $a1 holds nibble number
5   sllv $a0, $a0, $a1     # add $a1 to $a0
6   xor $v0, $v0, $a0      # $a0 flips bits
7   la $t0, inst
8   sw $v0, 0($t0)
9   inst:      addiu $v0, $0, 0   # machine code: 0x24020000
10  jr $ra
```

Answer the following questions assuming we are using a single-cycle CPU (i.e. no hazards).

**a)** What is the maximum number of instruction fields we can change simultaneously with a single call to `inst_nibble`? [1 pt]

Some nibble positions cross instruction field border (e.g. nibble 6 with opcode and rs/jaddr). [2]

**b)** Identify the *line number* of the function that accomplishes the following things:

- Retrieves the instruction to change 2 [0.5 pt]
- Flips the bits 6 [0.5 pt]
- Gets the address of the instruction to overwrite 7 [0.5 pt]
- Overwrites an instruction 8 [0.5 pt]

**c)** What value is returned in `$v0` after `inst_nibble` is called with `$a0=0xF, $a1=3`? Answer in hexadecimal. Pay attention to the code comments. [4 pt]

Machine code for Line 9 given as 0x24020000. With `$a0=0xF, $a1=3`, this gets changed to 0x2402F000. Since this is an I-type instruction (addiu), this only changed the immediate. Now the instruction `addiu $v0, $0, 0xF000` is stored and executed at Line 9. This sets `$v0` to the sign-extension of 0xF000, which is 0xFFFFFFFF.

**d)** If we wanted to use line 6 (`xor`) as our base instruction (i.e. data to manipulate), update the line 2 instruction to do this. **Hint:** How many lines of code is the `la` instruction? [2 pt]

Each instruction is 4 B. `la` becomes `lui, ori` (2 instr), so `xor` is actually 4 instructions above `addiu`, so need to offset by -16.
Question 5: Synchronous Digital Systems (8 pts)

Circuit Specs:
- 10 ns logic gate delay
- 5 ns setup time
- 0 ns hold time
- 5 ns CLK-to-Q delay

a) Give a Boolean expression for \( Y \) using the 3 basic gates (AND, OR, NOT). Simplify to use as few gates as possible. [1 pt]
\[
Y = (\overline{B} + \overline{C})D = \overline{B} \overline{C} D \quad \text{(DeMorgan’s – 3 gates)}
\]

b) Assume the values of A, B, C, D all change at time \( t = 0 \). After what time will both outputs \( X \) and \( Y \) reflect the final Boolean values for these inputs? Include units. [2 pt]

Longest path is from B or C to X, through 4 gates.

\[ t = 40 \text{ ns} \]

We add registers as follows and the inputs and outputs are now connected to registers. The circuit specs remain the same as before.

c) What is the minimum clock period we can use that ensures correct behavior? Include units. [2 pt]

Critical path is from \( A \) to \( X \), so 3 gates + CLK-to-Q delay + setup time.

\[ t = 40 \text{ ns} \]

d) Assume the clock period is 100 ns and the inputs to the registers that control the values of A, B, C, D cause them to change after the clock trigger at time \( t = 0 \). Assuming the new inputs are then held constant, after what time will the output \( Y \) reflect the final Boolean value for these inputs? Include units. [1 pt]

Longest path through left and right registers to \( Y \), so 2 clock periods + CLK-to-Q delay of right-most register.

\[ t = 205 \text{ ns} \]

e) With a clock period of 100 ns, what is the maximum hold time that will not cause a problem? [2 pt]

Shortest path is from right-most register to \( Y \) (connected to register).

\[ t_{\text{hold,max}} = 5 \text{ ns} \]
Question 6: Datapath (9 pts)

We wish to make a modification to the MIPS ISA such that the add instruction does not cause a general exception on overflow, but instead jumps to one of 32 specific exception handlers we can store in a special handler location.

Specifically, the syntax will look like this: \texttt{add \$rd \$rs \$rt \textless handler number\textgreater}, where the first three arguments are registers and the handler number field is an immediate between 0 and 31, inclusive. With this syntax, we can continue to use the R-type instruction format for this changed add instruction by putting the handler number in the \texttt{shamt} field. The specific behavior for the instruction is as follows:

\begin{verbatim}
if (\langle\$rs + \$rt does NOT cause overflow\rangle) {
    // no overflow. Add as normal
    \$rd = \$rs + \$rt;
} else {
    // overflow occurred, do not add, instead
    // jump to exception handler
    // store PC + 4 in \$ra to return from exception
    \$ra = PC + 4;
    // Set PC to handler location. All handlers are
    // stored at some offset past 0x0000F000
    PC = concat(0x0000F, \langle handler_number\rangle, 0b0000000);
}
\end{verbatim}

Based on the MIPS datapath diagram shown below, choose the modifications required in the dashed boxes (labeled (a), (b), and (c)) to execute the new instruction correctly.
a) For box (a), circle the number below for the circuitry required: [2 pt]

b) For box (b), circle the number below for the circuitry required: [2 pt]

c) For box (c), circle the number below for the circuitry required: [2 pt]

d) Based on the MIPS datapath diagram shown on the previous page, fill in the control signal values necessary to execute the add instruction correctly. [3 pt]

<table>
<thead>
<tr>
<th>add</th>
<th>RegDst</th>
<th>ExtOp</th>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>ALUCtrl</th>
<th>MemToReg</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>“ADD”</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Question 7: Pipelining (7 pts)**

We use a 5-stage pipelined processor with the following specifications:

- Branch comparison occurs during the execution stage.
- Separate I$ and D$
- Memory and register writes occur on each rising clock edge and reads at the falling edge.
- When necessary, the processor will insert stalls.

We run the following MIPS code that encrypts the string (address stored in $a0) by adding the encryption value ($a1) to each character (except the null terminator):

```
encrypt:
1  lbu  $t0, 0($a0)  N  0  0
2  beq  $t0, $0, end  D, C [1 pt]  4  3 [0.5 pt]
3  addiu $t0, $t0, $a1  D  1  0
4  sb   $t0, 0($a0)  D [0.5 pt]  2 [0.5 pt]  0
5  addiu $a0, $a0, 1  N [0.5 pt]  0  0 [0.5 pt]
6  j encrypt  C [0.5 pt]  1 [0.5 pt]  1
7 end: jr $ra
```

a) Fill in the blanks in the table above.

- In the Hazard(s) column, write “S” for structural, “D” for data, “C” for control, or “N” for none if that line potentially causes that type of hazard with no optimizations. You may use more than one per box.
- In the two Stalls columns, write how many stalls would need to be inserted (either before or after that instruction depending on the hazard) with and without forwarding.

**Trickiest one is Line 2 (beq). Two stalls from data hazard with lbu, two stalls afterwards for control hazard (branch decision made in EX). Because the data hazard is a load hazard, forwarding only removes 1 stall.**

b) We add branch delay slots to reduce the number of stalls (assume no forwarding). Circle ONE instruction below that can be safely moved into the branch delay slot assuming forwarding: [2 pt]

```
Line 1  Line 3  Line 5  Line 6
Line 2 needs Line 1 $t0 value. Moving Line 5 would cause lbu and sb to use different addresses. Moving Line 6 breaks program logic (would never enter loop). Moving Line 3 causes $t0 to be modified one extra time, but doesn’t affect program logic (goes unused because you don’t reach the sb when you branch).
```

c) Our pipelined CPU has a throughput of 1/(200 ps). What is its latency? Don’t forget units. [0.5 pt]

Max stage latency is 200 ps and have 5 stages. 1000 ps
Question 8: Caches (10 pts)

We are using a 20-bit byte addressed machine. We have two options for caches: Cache A is fully associative and Cache B is 4-way set associative. Both caches have a capacity of 16 KiB and 16 B blocks.

a) Calculate the TIO address breakdown for Cache A: [1 pt]

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

b) Below is the initial state of one set (four slots) in Cache B. Each slot holds 2 LRU bits, with 0b00 being the most recently used and 0b11 being the least recently used. Circle ONE option below for two memory accesses that result in the final LRU bits shown and **only one block replacement**. [2 pt]

<table>
<thead>
<tr>
<th>Initial</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Slot</td>
</tr>
<tr>
<td>0001 1110</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

(1) 0x019D0, 0xAD9D0   (2) 0xAC9E0, 0x129E0
(3) 0xAD9D0, 0x019D0   (4) 0x129E0, 0xAC9E0

For the code given below, calculate the hit rate for Cache B assuming that it starts cold. [3 pt]

```c
#define ARRAY_SIZE 8192
int int_arr[ARRAY_SIZE];        // &int_arr = 0x80000
for (int i = 0; i < ARRAY_SIZE / 2; i++) {
    int_arr[i] *= int_arr[i + ARRAY_SIZE / 2];
}
```

Access pattern is R\textsubscript{i}, R\textsubscript{i+ARRAY_SIZE/2}, W\textsubscript{i}. Array index jump is 4096*4 = 2\textsuperscript{14} B away, so maps into same set same set (I+O=12<14).
N=4, so both blocks can fit in cache at once. Indices are not revisited and each block holds 16 B / 4 B = 4 indices, so first index is MMH, other 3 are HHH, so HR = 10/12 = 5/6.

d) For each of the proposed changes below, write U for “increase”, N for “no change”, or D for “decrease” to indicate the effect on the hit rate of Cache B for the loop shown in part (c): [2 pt]

- Direct-mapped __D__
- Increase cache size __N__
- Double ARRAY\_SIZE __N__
- Random block replacement __D__

e) Calculate the AMAT for a multi-level cache given the following values. Don’t forget units! [2 pt]

<table>
<thead>
<tr>
<th>L1$ HT</th>
<th>L1$ MR</th>
<th>L2$ HT</th>
<th>GMR</th>
<th>MEM HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ns</td>
<td>20%</td>
<td>25 ns</td>
<td>5%</td>
<td>500 ns</td>
</tr>
</tbody>
</table>

\[ HT_{1} + MR_{1} \times HT_{2} + GMR \times HT_{MEM} = 4 + 5 + 25 \]

\[ HT = 34 \text{ ns} \]
Question 9: Virtual Memory (8 pts)

This election season, the US will computerize the voting system. There were approximately $2^{27}$ voters in 2012. There are four candidates in the running and so each voter will submit letter A, B, C, or D. The votes are stored in the char votes[] array.

The following loop will count the votes to determine the winner. We are given a 1 MiB byte-addressed machine with 4 MiB of VM and 128 KiB pages. Assume that votes[] and candidates[] are page-aligned and i is stored in a register.

```c
#define NUM_VOTERS 134217728 // 2^27
int candidates[] = {0,0,0,0}; // initialize to 0s
for (int i = 0; i < NUM_VOTERS; i++) { // Loop 1
    if (votes[i] == 'A') candidates[0]++;
    if (votes[i] == 'B') candidates[1]++;
    if (votes[i] == 'C') candidates[2]++;
    if (votes[i] == 'D') candidates[3]++;
}
```

a) How many bits wide are the following? [2 pt]

<table>
<thead>
<tr>
<th>VPN</th>
<th>5</th>
<th>Page Offset</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td>3</td>
<td>Page Table Base Register</td>
<td>20</td>
</tr>
</tbody>
</table>

b) We are given a fully-associative TLB with 4 entries and LRU replacement policy. One entry is reserved for the Code. In the best case scenario, how many votes will be counted before a TLB miss occurs? [2 pt]

Best case: TLB already has code page, candidate page, and 2 votes pages loaded. One page is $2^{17}$B. votes is a character array so each page holds $2^{17}$ votes. $2 * 2^{17} = 2^{18}$ votes

We want to improve our machine by expanding the TLB to hold 8 entries instead of 4. We also revised our for loop, which replaces Loop 1. Assume i and vote are stored in registers.

```c
for (int i = 0; i < NUM_VOTERS; i++) { // Loop 2
    char vote = votes[i];
    if (vote == 'A') candidates[0]++;
    if (vote == 'B') candidates[1]++;
    if (vote == 'C') candidates[2]++;
    if (vote == 'D') candidates[3]++;
}
```

c) Now how many votes can be counted before a TLB miss in the best case scenario? [2 pt]

Even though we have 1 access per vote instead of 4 with the new for loop, this does not change the fact that in the best case, we will only miss in the TLB if we go through all our pre-loaded pages. Since our TLB can now hold 6 pages for the votes pages, we can get $6 * 2^{17}$ votes before a miss.
d) In the worst case scenario, how many TLB misses would occur if this improved loop ran to completion? In other words, what is the highest number of TLB misses possible when running this loop? [2 pt]

In the worst case scenario, we start out with a cold TLB with nothing that is usable/valid when the loop begins. We will miss to fetch the code page and then miss to load in the candidate page. Those are 2 misses, and those pages will remain in the TLB throughout the execution of the loop since this TLB is LRU and the code and candidate page will constantly be accessed. This leaves us with misses for each page we need to fetch for the votes array. Our votes array is $2^{27}$ bytes and our page size is $2^{17}$ bytes. $2^{27}/2^{17} = 2^{10}$ pages, which we all cause a TLB when they are first fetched into the TLB. Thus, we have $2^{10} + 2$ TLB misses.
Question 10: Performance Programming (7 pts)

a) Circle ONE option below that best describes the result of running the following parallel code snippet. [2 pt]

```c
// given 512-element integer array A, count the occurrences of 61
int count = 0;
#pragma omp parallel for
{
    for (int i = 0; i < 512; i++){
        if (A[i] == 61) count++;  
    }
}
```

(1) Always correct  (2) Incorrect because of false sharing
(3) Incorrect because of data race  (4) Incorrect, because of data dependency

Data race on shared count variable (count++ → count=count+1).

b) To tackle data dependency, we can enforce the correct order of execution to ensure data dependency is always met. Consider the following hypothetical situation, we have three functions:

```c
void harvey();  void patterson();  void garcia();
```

These functions fill the three arrays H[], P[], and G[] sequentially. For example, if you call harvey() twice, then H[0] and H[1] will be filled. harvey() and patterson() work independently, but garcia() depends on the data in H[] and P[]. Specifically, to compute G[i], garcia() requires H[i] and P[i] to be already computed. Fill in the blanks below to ensure the program works correctly.

```c
omp_set_num_threads(3);  // allocate three threads
int h_idx, p_idx, g_idx = 0,0,0; // track indices
#pragma omp parallel
{
    int tid = omp_get_thread_num();
    while (g_idx < 1000) {  // stop when garcia() is done
        if (tid == 0){
            harvey();
            h_idx++;  [0.5 pt]
        } else if (tid == 1){
            patterson();
            p_idx++;  [0.5 pt]
        } else {
            while( g_idx >= h_idx || g_idx >= p_idx );  [3 pt]
            garcia();  [0.5 pt]
            g_idx++;  [0.5 pt]
        }
    }
}
```
Question 11: MapReduce (8 pts)

a) You try to speed up some code up by parallelizing with 8 threads. What fraction of your code needs to be parallelizable to achieve a 4X speedup? [1 pt]

\[
4 = 1/(1 - F) + F/8; \quad 4 - 4F + F/2 = 1; \quad 3 = 7F/2
\]

b) You decide to figure out whether each of your friends is active on Facebook as a “poster” (makes posts) or a “liker” (clicks Like) using MapReduce on Facebook activity data. You score each “user type” as follows:

- poster-score = (number of likes) points for each post
- liker-score = 1 point for each like

Each friend has a poster-score and a liker-score. In your output of the reduce function, you should use the ratio \(\text{poster-score}/\text{liker-score}\) to indicate what type of user this friend is. You **MUST** differentiate between poster-score and liker-score by emitting **negative numbers** for poster-score from the map function. Make sure your output from reduce is a **positive number**.

You may use the function `length(list)`, which returns the length of a list. Fill in the blanks below:

```python
map( name, posts ) {
    post_score = 0  [0.5 pt]
    for post in posts:  # post contains list of friends who Liked it
        post_score += length(post)  [1 pt]
        for friend_name in post:
            emit( friend_name , 1 )  [1 pt]
    emit( name , -post_score )  [1.5 pt]
}
```

```python
reduce( key, scores ) {
    PS = 0  # poster score
    LS = 0  # liker score
    for score in scores:
        if score > 0:  [1 pt]
            LS += score
        else:
            PS = -score  [1 pt]
    emit( key , PS/LS )  [1 pt]
}
```
Question 12: Special Topics Potpourri (8 pts)

a) Availability: For a datacenter, the mean time to failure (MTTF) is 1000 hr.
   
i. What is the annualized failure rate (AFR) of our datacenter? [0.5 pt]
      \[ \text{AFR} = \frac{8760}{\text{MTTF}} \]
      \[ \frac{8.76}{1} \]

   ii. If we want two 9s of availability, what is the maximum mean time to repair (MTTR) we can have? Leave your answer as a fraction. [1 pt]
      \[ 0.99 = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}; \quad 0.99 \times \text{MTTR} = 0.01 \times \text{MTTF} \]
      \[ \frac{1000}{99} \]

a) Error Correcting Codes (ECC): We use Hamming ECC for two bits of data.
   
i. What is the correct data word associated with the invalid code word 0b10111? [1 pt]
      Errors in parity groups p1, p2; correct to 0b10011
      \[ \text{0b01} \]

   ii. What fraction of all possible code words are valid? Leave your answer as a fraction. [1 pt]
      4 valid (corresponding to 00, 01, 10, 11) of \( 2^5 = 32 \) code words
      \[ \frac{4}{32} = \frac{1}{8} \]

b) Redundant Array of Inexpensive Disks (RAID): [2.5 pt]
   
i. A power surge fries one of our disks. Which RAID type would not be able to recover from this?
      Data striping but no parity data
      RAID 0

   ii. What is the data transfer bottleneck in RAID 4 that is “fixed” in RAID 5?
      RAID 5 does interleaved parity blocks
      parity disk

   iii. We use 5 disks for RAID. If we need to read 32 B of block-aligned data, how many disks do we need to access in each of the following situations? Don’t include the parity check (handled by the disk controller).
      Data striped across all 5 disks
      RAID 0 with byte striping: \[ \underline{5} \]

      Data striped across all disks except parity disk
      RAID 3: \[ \underline{4} \]

      32 B fits within single block (single disk)
      RAID 5 with block size of 64 B: \[ \underline{1} \]

c) Input/Output (I/O): Apple has just released a new USB device that does [something revolutionary]! Now we need to decide whether to interface with it using polling or interrupts. Consider a CPU running at 2 GHz.
   
i. Our polling routine can only hold 32 B of data at a time. If the device produces data at a rate of 64 KiB/sec, how fast do we need to poll the device (polls/sec) so not to miss any data? [1 pt]
      \[ \frac{64 \text{ KiB}}{s} = 2^{16} \frac{B}{s}; \quad 32 \frac{B}{\text{poll}} = 2^5 \frac{B}{\text{poll}} \]
      \[ 2^{11} = 2048 \text{ polls/sec} \]

   ii. If the device interrupts 1000 times a second with a transfer overhead of 2000 clock cycles, what percentage of the processor is used by the device? [1 pt]
      \[ 1000 \frac{\text{interrupts}}{s} \times 2000 \frac{\text{cycles}}{\text{interrupt}} = 2 \times 10^5 \frac{\text{cycles}}{s}; \quad 2 \text{ GHz} = 2 \times 10^9 \frac{\text{cycles}}{s} \]
      \[ 0.001 = 0.1\% \]

SID: ________________