University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Summer 2015 Instructor: Sagar Karandikar 2015-08-13

CS61C FINAL

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	ANSWER KEY
First Name	
Student ID Number	
Login	cs61c-
*The name of your **SECTION** TA (please circle)*	Derek | Harrison | Jeffrey | Nathaniel | Rebecca
Name of the person to your Left	
Name of the person to your Right	
*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. **(please sign)***	

Instructions (Read Me!)

- This exam contains 16 numbered pages including the cover page. **The back of each page is blank and can be used for scratch-work but will not be looked at for grading.** (i.e. the sides of pages without the printed "SID: ________" header will not even be scanned into Gradescope).
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 80 minutes to complete this exam. The exam is closed book; you may not use any computers, phones, wearable devices, or calculators. You may use one page (US Letter, front and back) of handwritten notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided. "IEC format" refers to the mebi, tebi, etc prefixes.

	M1-1	**M1-2**	**M1-3**	**M2-1**	**M2-2**	**M2-3**	**F1**	**F2**	**F3**	**Total**
Points Possible	11	20	18	20	15	19	12	20	17	152

SID: ______________

M1-1) Number Representation (11 pts)

Part 1: Quick conversions:

a) What is 0xFFFFFFFC in decimal when read as an unsigned value? **2^32-4**

0xFFFFFFFC is 0b1111 1111 1111 1111 1111 1111 1111 1100, which is the sum of all powers of 2 between 2^2 and 2^31, inclusive, which is the same as (2^32 - 1) - 3. This is because if we're interpreting a binary string of all 1's of length N as an unsigned number, its value is 2^N - 1. Below is an inductive proof:
Inductive Hypothesis: For some N >= 1, an unsigned binary string of all 1's of length N has value 2^N - 1.
Base case: N = 1, (binary string = 0b1) => value is 1.
Inductive Step: Denote the value of the string of all 1s of length X "S_X".
S_N = S_{N-1} * 2 + 1 = (2^(N-1) - 1) * 2 + 1 = 2^N - 1

b) Encode the decimal value -10 in 32 bits of two's complement. Give your answer in hex. **0xFFFFFFF6**

First we encode +10 as a two's complement number. +10 = 0b 0000 .... 0000 1010
Then, we use the two's complement negation trick (flip all bits and add 1) to get -10.
~(+10) = 0b 1111 .... 1111 0101
-10 = ~(+10) + 1 = 0b 1111 .... 1111 0110 = 0xFFFFFFF6

Part 2: We have a new number representation called Balanced Ternary that uses "Signed Digit Representation" in ternary trits. Each digit can individually represent a positive, zero or a negative value. The n^th digit (zero-indexed from the right) gets multiplied by 3^n – just like in regular ternary. Instead of having the three states represent 0, 1 and 2, they represent for **-1, 0, and 1**. **We will use + to denote +1, 0 to denote 0, and – to denote -1.**

Convert the following Balanced Ternary numbers to decimal.

c) 000+ ____________________
1

d) 00+- ____________________
2

e) +--0 ____________________
27 - 9 - 3 = 15

c) 000+ means (+1) * 3^0 = 1
d) 00+- means (+1) * 3^1 + (-1) * 3^0 = 3 - 1 = 2
e) +--0 means (+1) * 3^3 + (-1) * 3^2 + (-1) * 3^1 = 27 - 9 - 3 = 15

f) How many ways can you represent the decimal number 4? Write all of them out.
~~One: 0011~~

This answer should read "One: 00++".

What do you get (expressed in balanced ternary) when you add the following numbers vertically?
(Hint: use the result from (g)-(i) to do (j))

```
g)   +          h)   0+          i)   0-          j)   ++0+ - -
     -               0+               0-               - +0++ -
  __________      __________      __________      _______________
     0               + -              - +              + - + - - +
```

g-j) Use the normal addition algorithm. Note that add("+", "+") = "-" with a carry-over of "+" to the next place, because 1 + 1 = 2 = 3 - 1 = "+-". Similarly, add("-", "-") = "+" with a carry-over of "-" to the next place, because -1 + -1 = -2 = -3 + 1 = "-+"

M1-2) C Pointers, Arrays and Strings (20 pts)

a) Fill in the blanks below for the function `tokenize` which splits up a string into proper C strings whenever it encounters a specified delimiter (a single character).

Example:

```
char * string = "Summer*time";
printf("%s", tokenize(string, '*')[0]);     // prints:  Summer
printf("%s", tokenize(string, '*')[1]);     // prints:  time
```

Inputs:

`char * str` - a NULL terminated character array

`char delimiter` - a character to split on

- You may **assume that the delimiter appears at least once in the input str** and the first/last character in the input string are not the delimiter.

Outputs:

An array of strings split by the provided token (pointers and string data on the heap). See the example above. The delimiter should never appear in an output string and your output should never contain strings of length zero.

Requirements:

You may use **one** additional malloc call beyond the one provided in the skeleton.

Assume that you have access to the functions:

`int count_occurences(char * str, char token);` // Counts occurrences of a token in a string

`char * strcpy (char * destination, char * source);` // copies string `source` to buffer `dest`

```
___char **___ tokenize(char * str, char token) {
     char * copy = malloc(sizeof(char) * strlen(str) + ______1_______);
     strcpy(copy, str);

     char ** string_array =  malloc(sizeof(char*)*count_occurences(str, token));

     int index = 0 __________________________________;

     string_array[index] = copy_________________________________;

     while(______*copy ______________________ != '\0'){

          if (________*copy == token && *(copy+1) != token________________){

               *copy = '\0';  ___________________________

               index += 1; _________________

               string_array[index] = ++copy; __________

          } else if (_____*copy == token____________________){

               *(copy++) = '\0'; ___________________________

          } else {______copy++;_____________________________________}
     }
     return string_array;
}
```

See next page for detailed (and corrected) answer.

Essentially, what this code does is copy the input string to the heap, and then replace the delimiter token with null terminating characters, all the while adding pointers (which point to the first character following those replaced tokens) to an array. Note - Original solution had an error on the line which allocates space for `string_array`.

```
char ** tokenize(char * str, char token) { // we want to break the input character pointer into several character pointers and return them.
                                                Recall that in C, "string", simply means a null-character terminated character pointer.
        char * copy = malloc(sizeof(char) * strlen(str) + 1); // 1 comes from our need to allocate space for the null terminating character,
                                                                which strlen does not count as part of the length of the string
        strcpy(copy, str);
        char ** string_array = malloc(sizeof(char*)*(count_occurences(str, token) + 1)); // if there are n delimiters, we want to split the
                                                                                          string into n+1 parts.
        int index = 0;
        string_array[index] = copy; // the first string starts where the input string begins, because we assume that the first character in the
                                        input string is not the delimiter
        while(*copy != '\0'){ // loop until we hit the null terminator
                if (*copy == token && *(copy+1) != token){ //if we have two delimiting characters in a row we want to go to the next
                                                                        case
                        *copy = '\0';
                        index += 1;
                        string_array[index] = ++copy; // recall that ++copy increments pointer and then returns the pointer value
                } else if (*copy == token){
                        *(copy++) = '\0'; // recall that copy++ returns the original pointer value and then increments pointer value
                } else {copy++;}
        }
        return string_array;
}
```

M1-2) C Pointers, Arrays and Strings (continued)

b) What does the following program print if the starting address of x is 1000_{ten}, and the starting address of array arr is 2000_{ten}? All addresses are byte-addressed.

```
#include <stdio.h>
int main() {
     int *p, x;
     int arr[5] = {1000, 2000, 3000, 4000, 5000};
     int *p2;
     p = NULL;
     x = 5000;
     p = &x; //p = 1000, *p = p[0] = 5000
     printf(“%d %d %u %u\n”, x, *p, p, &x);
     p2 = arr; //p2 = 2000
     *(p2+1) = *p; //p2[1] = 5000
     *p = *p2 + *(p2+2); //*p = x = p2[0] + p2[2] = arr[0] + arr[2] = 4000
     p2++; //p2 = arr + 1, so p2 = 2000 + sizeof(int) = 2004
     printf(“%d %d %d %u\n”, x, *p, *p2, p2);
     return 0;
}
```

____5000 5000 1000 1000

____4000 4000 5000 2004

M1-3) What's happening? It's a MIPStery! (18 pts)

The code below does something funky. You can assume that we only need to run this code on a "bare" system – one with no address translation (i.e. no Paging or Base and Bounds).

```
0x000004  main: li   $a0, 1
0x000008        jal      mystery        # ← (A)
0x00000c        addu     $a0, $0, $v0
0x000010        jal mystery         # ← (B)
0x000014        addu     $a0, $0, $v0
0x000018        jal mystery         # ← (C)
0x00001c        addu     $a0, $0, $v0
0x000020        jal mystery         # ← (D)
          …
0x800004  mystery: lui    $t0, 0xffff #used to keep top 16 bits from inst.
0x800008           lui    $t2, %Hi(mystery) $t2 = address of mystery
0x80000c           ori    $t2, %Lo(mystery)
0x800010           addiu $t1, $0, 0 #the instruction at this address is modified
0x800014           andi  $a0, $a0, 0xffff #only using bottom 16 bits of $a0
0x800018           add    $v0, $a0, $t1 #despite appearances, t1 is not always 0
0x80001c           lw     $t3, 12($t2)#load the fourth instruction past 'mystery'
0x800020           and    $t3, $t3, $t0#keep top 16 bits of instr(not imm. bits)
0x800024           or     $t3, $t3, $a0#instruction's imm. is now the (n-1)th fibnumber
0x800028           sw     $t3, 12($t2)#store modified inst. back to 0x80000c
0x80002c           jr     $ra
```

a) What is the value in $v0 **after** returning from the function call in the line marked:

(A) ________ 1   (B) ________ 2   (C) ________ 3   (D) ________ 5

b) What does the function mystery return if we follow the pattern in main above, calling mystery N times? (You should use N in your answer.)

N-th Fibonacci number on the N-th call. This is achieved by modifying instruction at 0x80000c during the n-th call to store n-1th fibonacci number in its immediate bits

c) Write TAL MIPS code to "reset" the function mystery such that after a jal reset_mystery line, the next call to mystery with $a0 = 1 behaves like the function call in (A). You might not need all of the lines provided. In your answer, you can use the %Hi and %Lo directives used above.

reset_mystery:

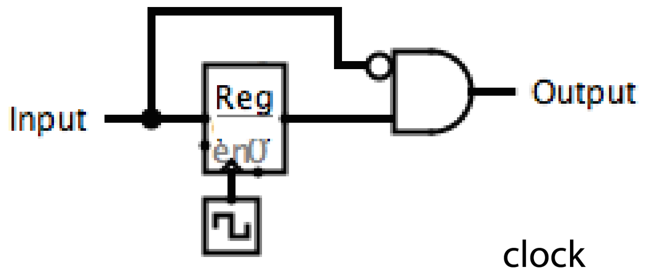
```
lui   $t0, 0xffff
lui   $t2, %Hi(mystery)
ori   $t2, %Lo(mystery)
lw    $t3, 12($t2)
and   $t2, $t2, $t0
sw    $t2, 12($t2)
jr    $ra
```

Meant to zero out bottom 16 bits of the instruction at 0x80000c while keeping top 16 bits. Technically, we should ensure that $t0 is initially 0, by having "add $t0 $0 $0" be the first line after reset_mystery:

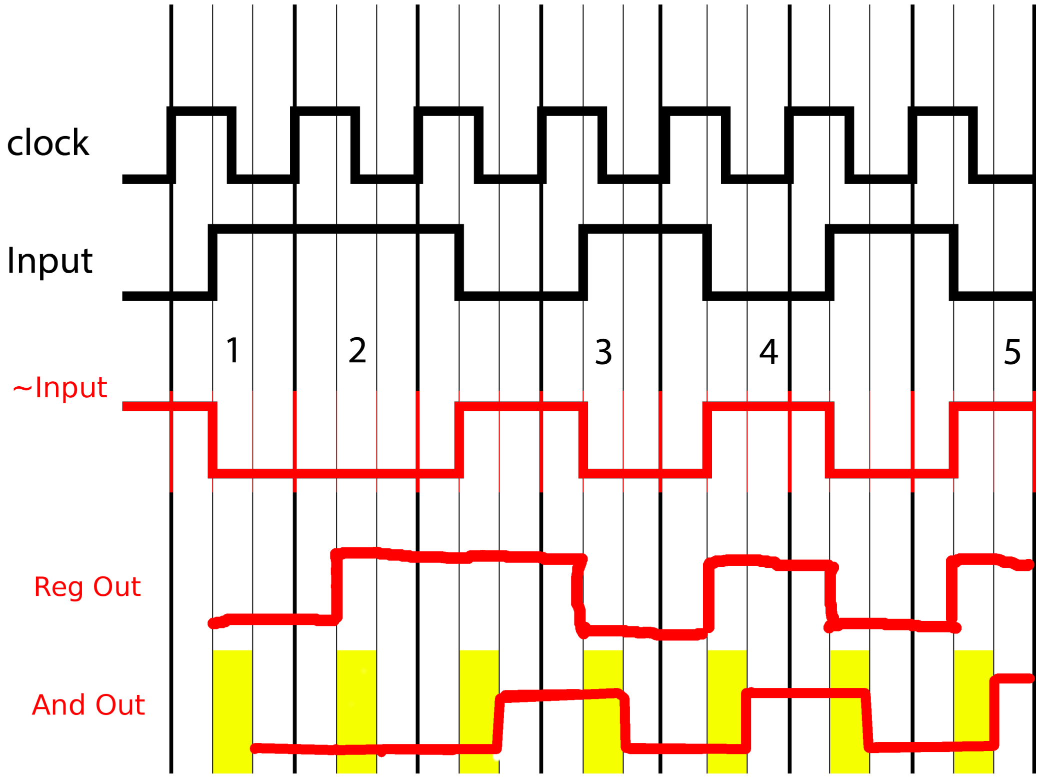
SID: _______________

M2-1) SDS, Gates, Timing, FSM (20 pts)

a) Assume that the clock period is 15ns, the input comes 5ns after each positive clock edge, the register has no hold and setup times but has a clock-to-Q delay of 5ns, and gates have a 5 ns delay (with or without bubbles in the input).



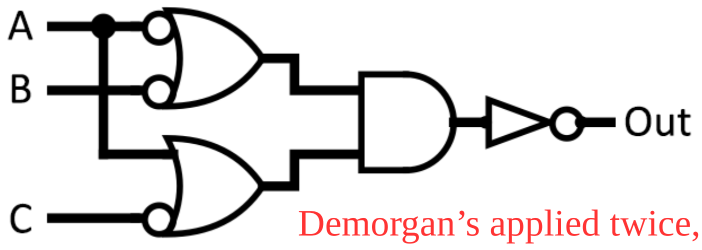
Clock cycle is 15, so distance between vertical lines is 5 ns.. Reg Out is updated at every rising edge, but is delayed 5ns due to CLK-To-Q delay.
To see what the Out is, look at the vertical column of the logical AND funcion applied to the not'd input with the register output.
Note that the AND gate has a delay of 5ns, so it takes 5 ns for Output to be updated given a change in Input or Reg Out



What is the value of the output signal at the numbered markers? **Write X for unknown/undefined**. We have provided space for you to sketch out signals, but we will not grade any sketches.

1) ___________X 2) ___________0 3) ______1_____ 4) ___________1 5) ___________1

b) Write a Boolean expression for Out in terms of inputs A, B and C in the circuit below and simplify the expression so it can be built using only 4 basic gates without bubbles. Basic gates are: OR gates, AND gates, NOT gates.



From the final expression we can see that we need 2 AND gates, 1 OR gate, and 1 NOT gate

Demorgan's applied twice, then once more, then cancel NOT

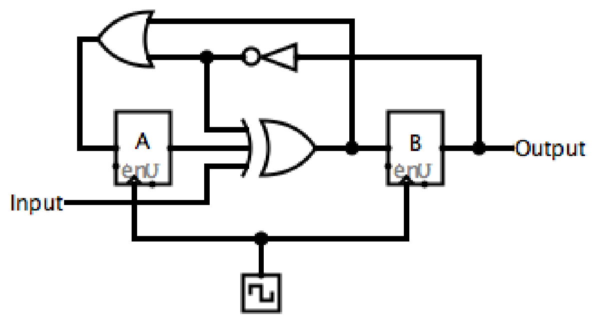
`~((~A + ~B)(A + ~C)) = ~(~(AB) ~(~AC)) = AB + ~AC`

SID: ______________

M2-1) SDS, Gates, Timing, FSM (continued)

Recall: the critical path delay is the longest period of time which data has to travel in a single clock period, meaning the longest path between a pair of inputs, state elements (registers), and/or outputs.

c) The circuit below has the following timing parameters:
Clock period: 20ns
XOR gate delay: 5ns
OR gate delay: 5ns
Inverter delay: 4ns
Register setup time: 2ns
Input comes 1ns after every clock edge



i) What is the maximum possible clk-to-Q delay for the registers?

We're asked to find the an upper bound on the CLK-to-Q delay, which means we have to find the length of the critical path for some general CLK-to-Q delay, and then take the difference between the clock period and the critical path delay without the CLK-to-Q delay to find the maximum CLK-to-Q delay.
In this case, the path from register B to register A through the inverter is the critical path. Starting from B, we incur the CLK-to-Q delay (X), the NOT/INV delay (4 ns), the OR delay (5 ns), and the setup time at A (2 ns). X = 20 -4-5-2=4
Note: The hold time doesn't matter for this calculation, because it overlaps with the CLK-to-Q and in any case, does not impact the upper bound on the CLK-to-Q (it might impact the lower bound on the CLK-to-Q).

```
Register A: 20 – 5 (XOR) – 5 (OR) – 2 (setup A) = 8ns
Register B: 20 – 4 (inv) – 5 (XOR) – 5 (OR) – 2 (setup A) = 4ns
Both registers: min(8,4) = 4 ns
```

ii) Assuming a clk-to-Q delay of 3ns, what is the maximum possible hold time for the registers?

```
Register A: 3 (clk-Q B) + 4 (inv) + 5 (OR) = 12ns
Register B: 1 (input) + 5 (XOR) = 6ns
Both registers: min(12,6) = 6 ns
```

Now we're trying to set an upper bound on the hold time such that we don't incur a hold time violation. Recall that a hold time violation occurs when new data arrives at a register before the hold time has passed since the last rising edge of the clock. So, we're concerned with the shortest path data travels before it reaches a register. In this case, the shortest path is from the input to register B. Data arrives at the input wire 1 ns after the rising edge of the clock, then travels through the XOR gate (5 ns) to register B. By the time 6ns has passed, the hold time must be over, otherwise there is a hold time violation.

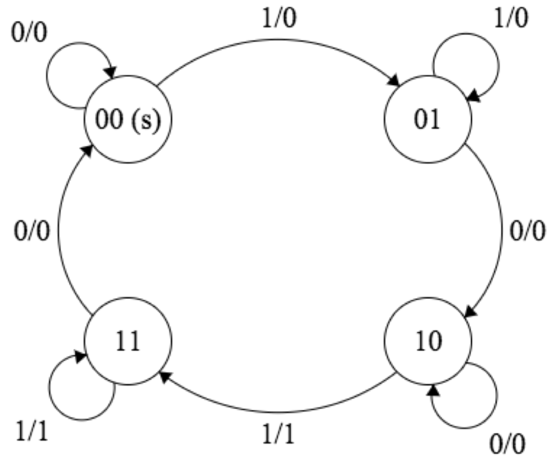
M2-1) SDS, Gates, Timing, FSM (continued)

d) Using as few states as possible, **draw an FSM diagram** that satisfies the following:

- We are reading a buffer bit-by-bit containing data surrounded by a "header" and a "footer".
- The header is defined to be the bit sequence "10". Once we have seen "10", then we will read in bit-by-bit and always output the bit we read in last.
- The footer is also defined to be the bit sequence "10". If we are in the body, once we have seen a "10" we are at the end of the data section.
- The FSM is initialized to detect headers. Whenever we are not within the body of the data or the footer, it always outputs 0. Note that this buffer is infinitely long and should continue to detect a header after finishing a footer.

Use labels/words for your states instead of 1's and 0's to make your life easier.
Note the following sample input -> output scheme:
Cycle 01: 0 -> 0 # garbage value
Cycle 02: 1 -> 0 # header
Cycle 03: 0 -> 0 # header
Cycle 04: 0 -> 0 # data
Cycle 05: 0 -> 0 # data
Cycle 06: 1 -> 1 # data
Cycle 07: 1 -> 1 # footer
Cycle 08: 0 -> 0 # footer
Cycle 09: 1 -> 0 # garbage value
Cycle 10: 1 -> 0 # garbage value
Cycle 11: 1 -> 0 # garbage value
Cycle 12: 1 -> 0 # header
Cycle 13: 0 -> 0 # header
etc…

Place your solution in this box. Be sure to label the start state ((s) in the solution)



indicate which

SID: ______________

M2-2) MIPS Pipeline (15 pts)

The code on the left is written for a single-cycle MIPS CPU. **Put a star to the left of lines that are involved in hazards** (both the lines that cause and are affected by hazards), reorganize the code order, and insert nops so that the code runs in as few cycles as possible on a 5-stage pipelined MIPS CPU. Assume that the CPU has complete forwarding in the pipeline with load, branch and jump delay slots; we have a branch comparator in the decode stage; but we do not have the ability to send bubbles in the pipeline. You might not need all of the provided lines.

```
Start:                      We can move this line to right after the branch, because
                            it needs to get executed but doesn't affect/depend on the
    and  $t4 $t1 $t2        other instructions before beq

    ori  $t5 $s0 0xFFFF

    add  $t0 $t1 $t2

<=  add  $s0 $t2 $t3        Data hazard on next line, which is unavoidable so
                            we need to put in a nop
    beq  $s0 $t0 Somewhere  Control hazard - a control hazard occurs whenever
<=                          there is a branch, because we don't know whether we
    addi $s2 $0  0          are branching until after the Decode stage, so we
<=                          always execute the instruction immediately after the
Loop:                       Branch, so we sometimes need to insert a nop.

    lw   $s1 4($s0)         Data hazard on next line
<=
    addi $s1 $s1 1          Data hazard on next line, which can be avoided by
                            shifting in unrelated addi
<=
    sw   $s1 0($s0)         This will definitely be executed if put after jal

    addi $s2 $s2 1

    jal  DoStuff            Control hazard
<=
    addi $s0 $s0 4
<=
    blt  $s1 $s2 Loop       Control hazard, needs something to come after it
<=
    beq  $s1 $s2 Start      Control hazard
<=

    jr $ra <=
```

Data hazards occur when registers which are written to are then used as operands in some other instruction which follows soon after. However, because the CPU has forwarding, as soon as data leaves the ALU (or Memory if a load), it can be used by another instruction in its Execute stage (or ID stage if it's a branch.

Ex1: addi t0 x0 1 → data goes from end of Execute to beginning of next Execute,
add t1 t0 t0 requires no instructions between the two of them

Ex2: lw t0 0(a0) → data goes from end of Memory to beginning of next Execute,
add t1 t0 t0 requires one instructions between the two of them

Ex3: lw t0 0(a0) → data goes from end of Memory to beginning of next Decode,
beq t0 t0 label requires two instructions between the two of them

```
Start:
    ori  $t5 $s0 0xFFFF
    add  $t0 $t1 $t2
    add  $s0 $t2 $t3
    nop
    beq  $s0 $t0 Somewhere
    and  $t4 $t1 $t2
    addi $s2 $0  0
Loop:
    lw   $s1 4($s0)
    addi $s2 $s2 1
    addi $s1 $s1 1
    jal  DoStuff
    sw   $s1 0($s0)
    blt  $s1 $s2 Loop
    addi $s0 $s0 4
    beq  $s1 $s2 Start
    nop
    jr $ra
```

M2-3) Cache Performance/AMAT (19 pts)

Each subproblem in this question can be done without the previous subproblems.
We have caches with the following configurations:
Cache-size: 1KiB, Block-size: 128B, Memory-size: 4 GiB, Write-policy: Write back and write-allocate

```
int rand(x,y);  // defined elsewhere, returns a random integer in the range [x,y)
#define repcount 4
int A[1024*1024]; // block aligned
int B[8*1024]; // block aligned
...
// ⇐ Start (assume cache is cold)
for(int rep = 0; rep < repcount; rep++)
     for (int i = 0; i < 1024; i++)
          A[8*i] = B[8*i] + A[256*i + 8*i + rand(0, 32)] + A[8*i];
// ⇐ Stop
```

For the following scenarios calculate the hit-rate and specify which types of misses are encountered in the code between the lines marked start and stop. Assume all variables stay in registers and memory reads happen left to right in each line.

Index bits = log_2(# sets) = log_2(#blocks/#ways)

a) i) Direct Mapped: # Tag/Index/Offset bits: [_ 22________ : _3________ : _7________]

ii) What is the best-case hit rate? Also, state the types of misses.

- best case: A and B aren't within the same index at the start
- 32 ints / block
- 4 ints / block are accessed with the A[8*i] and B[8*i] accesses
- A[256*i + 8*i + rand(0,32)] always accesses a new block which shares same index as A[8*i] (since 256*i will make the addresses always differ by a multiple of the cache size)
- rand() doesn't affect anything
- B[8*i] hits 3/4 of the time, because 4 consecutive accesses in same block
- A[256*i + 8*i + rand(0, 32)] always maps to same set as A[8*i] so neither read will ever hit (256*i means incrementing by 2^10 Bytes, a multiple of cache size)
- A[8*i] write always hits (directly after read)
- 1/4*(1) + 1/4*(3/4) + 0*(1/4)+ 0*(1/4) = 7/16
- We encounter compulsory, capacity (because entire working set does not fit in the cache, and conflict misses (because if fully associative we would hit A[256*i + 8*i + rand(0,32)])

iii) What is the worst-case hit rate? Also, state the types of misses.

- worst case: A and B are within the same index at the start
- Accesses keep replacing blocks except the write (since it directly follows the read to A[8*i]
- All misses except write
- Hit rate: 1/4*1 + 1/4*0 + 1/4*0 + 1/4*0 = 1/4
- We encounter compulsory, capacity, and conflict misses.

SID: ________________

b) i) 2-way Set Assoc: # Tag/Index/Offset bits: [_ 23_______ : _2_________ : _7________]
ii) What is the worst-case hit-rate? Assume we use LRU replacement. Also, state the types of misses.

- worst case: A and B are within the same index at start
- Consider what cache looks like at start:
- Load B[8*i], then load A[256*i + 8i+ rand(0, 32)], A[8*i] then kicks out B[8*i], the write on A[8*i] then hits, the load of B[8*i] then replaces A[256*i + 8i+ rand(0, 32)], A[256*i + 8i+ rand(0, 32)] then replaces A[8*i], then the cycle repeats
- Thus the only hits are on the write
- Hit rate: 1/4*1 + 1/4*0 + 1/4*0 + 1/4*0 = 1/4
- We encounter compulsory, capacity, and conflict misses.

c) Calculate the AMAT in cycles if the L1 local hit rate is 60%, with a hit time of 1 cycle, the L2 global hit rate is 20% with a hit time of 10 cycles and the main memory has a hit time of 200 cycles.

1 + 0.4*(10 + (1 - .2/.4) * 200) = 45

We always have to check L1 Cache, if block isn't there we need to check L2, if block isn't there we need to go to main memory.

AMAT = cache hit time + L1 local MR * L1 miss penalty
L1 miss penalty = L2 hit time + L2 local MR * L2 miss penalty
L2 miss penalty = main memory hit time
L2 local HR * L1 local MR = L2 global HR
L2 local MR = 1 – L2 local HR

F1) MapReduce (12 pts)

You have been hired as a computer scientist for a retail company where your boss is trying to optimize the location of his products in the store by analyzing the products bought by customers. He decides that he wants you to take their existing code and add on MapReduce in order to analyze giant logs of information more quickly.

a) Right now, the machine that you work on finishes the old program in 30 seconds. Your boss gives you extra funds for a new machine that will be able to take existing code and have it finish in 10 seconds. You notice that 5/6 of the code is parallelizable. What is the minimum number of threads your new machine has to have in order to achieve this speedup?

According to Ahmdal's

```
30/10 = 1/[(1-(5/6) + ((5/6)/N)]
N = 5
```

b) You now move on to working on MapReduce. Your boss, who loves hats, complains that sales for hats are not great. Previously, he tried putting different colors of hats with items of similar type but sales did not improve.

So, he proposes to take advantage of spatial information and wants to move the hats according to the aisle number (10 in total) of other items that were bought by those who bought hats.

For example, if people have bought items in aisle 2 AND they bought a red hat, your boss wants to consider moving red hats to aisle 2 because customers who buy items in aisle 2 has a higher probability to buy a red hat.

You are given a customer log which contains:
`(customer_id, list(item_object))` where `item_object` is defined as:

```
item_object {
      int color      // Color of product encoded as an integer
      int product_id // The ID of the product; the ones digit of item_id
}                    // corresponds to aisle number it is in (only 10 aisles)
```

Your output should be in the form of N key-value pairs `(hat_color, aisle_number)` if there are N unique hat colors bought in the customer log. These pairs should tell your boss where the different colored hats should be put.

Also, **assume you are given the following functions**:
1) `isHat(product_id)` that takes a product ID and returns whether that product is a hat or not.
2) `indexMax(array)` that takes an array and outputs the **index** of the largest value in the array.
3) `set1.add(value1)` adds some value (`value1`) to a `set()` named `set1`

You can access fields in an `item_object` using dot notation (e.g. `item_obj.color`)

Write **pseudocode** in the provided MapReduce functions on the following page so that at the end of your reduce function, you will provide the best color hat that should be placed in each aisle.

F1) MapReduce (continued)

```
map(customer_id, item_list) {
      hats = set()                  // empty set
      other_items = set()           // empty set
      for item in item_list:

            aisle_number = _____item.product_id % 10______________

            if (isHat(item.product_id)):________

            _____hats.add(item.color)_____________

            else:_________________________________

            _____other_items.add(aisle_number)
      for hat in hats:
            for item in other_items:

                  emit(_____hat, item______)
}


reduce(key, values_list) {
      totals = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

      for ____item in values_list____________:

            totals[item] += 1_________________________

      emit(_______key, indexMax(totals)________)
}
```

Want to emit one key value pair for every combination of (hat color, aisle of non-hat item) that a particular customer bought.

Then, there is an implicit GroupByKey, where the key is hat color, and the value is a list of aisles in which customers who bought a hat of that color bought some other non-hat item. We want to know which aisle was the most common among customers who bought a hat of that color.

SID: _______________

F2) Virtual Memory (20 pts)

For the following questions, assume the following (IEC prefixes are on your green sheet):

- **You can ignore any accesses to instruction memory (code)**
- 16 EiB virtual address space per process // 2^{64} B
- 256 MiB page size // 2^{28} B
- 4 GiB of physical address space // 2^{32} B
- Fully associative TLB with 5 entries and an LRU replacement policy
- All arrays of doubles are **page-aligned** (start on a page boundary) and do **not** overlap
- All arrays are of a size equivalent to some nonzero integer multiple of 256 MiB
- All structs are tightly packed (fields are stored contiguously)
- All accesses to structs and arrays go out to caches/memory (there is no optimization by reusing values loaded into registers)

```
---------------------------------------------------------------------------
typedef struct { *double dbl; double fun;} doubleFun

void dblCpy(doubleFun* measurer, double* dblsToCpy) {
     measurer->fun = 0;
     for (uint32_t i = 0; i < ARRAY_SIZE; i+=4) {
          measurer->dbl[i] += dblsToCpy[i];
          measurer->fun += dblsToCpy[i];
     }
     measurer->fun /= ARRAY_SIZE;
}
...
/* Now, the code goes on to call the function dblCpy. Assume that space
for the array pointed to by measurer->dbl was allocated at some time in
the past and that all elements in the array were set to 0. The arrays
dblsToCpy and measurer->dbl are each of length ARRAY_SIZE.*/
... // dblCpy function call here
---------------------------------------------------------------------------
```

a) Fill the following table: Recall that #VA offset bits == #PA offset bits

Virtual Page Number Bits: 36	Virtual Address Offset Bits: 28
Physical Page Number Bits: 4	Physical Address Offset Bits: 28

b) Assume the TLB has just been flushed. What TLB hit to miss ratio would be encountered if sizeof(double) * ARRAY_SIZE = 256 MiB and we run the above code? Show your work.

$6(2^{23})$ TLB hit: 3 TLB misses

*Our loop has 2^{28} / $(2^3 * 2^2)$ = 2^{23} iterations. For each iteration, there are 6 memory accesses. For the loop entirely, there are $6(2^{23})$ total memory accesses.
*Before the loop, there is one memory access. After the loop there are two memory accesses (read & write measurer-> fun).
*Thus, there are $6(2^{23})+3$ total memory accesses. We require three pages, one for dblsToCpy and two for doubleFun all that miss initially when loading up. Thus, our hit-miss ratio is $6(2^{23})$: 3.

F2) Virtual Memory (continued)

c) In the best-case scenario, how many iterations can be executed with no TLB misses? Use IEC prefixes when reporting your answer. Show your work.

2^{24} iterations = 16 MiB.

In the best case scenario, all 5 slots in the TLB are full of valid info. We have 2 pages worth of measurer->dbl, 1 page for dbl->fun, and 2 pages for dblsToCpy. This means that our arrays are 2^{29} B (2 pages) in length. We index by every 4 elements of 8-byte doubles each time to obtain 2^{29} / 2^{2} / 2^3 = 2^{24} iterations.

[EXTRA ROOM TO SHOW YOUR WORK BELOW.]

You have to put your final result under the problem. Any final result below this text will not be graded.

SID: _______________

F3) Potpourri (17 pts)

a) What is the first positive integer divisible by 5 that is not representable by floating point representation? (Hint: 2^{4n} divided by 5 gives a remainder of 1 for any integer value of n)

2^{24}+9 see next page for explanation

b) What is the fraction of integers in the range $[2^{26}, 2^{27})$ that are not representable in single-precision IEEE 754?

7/8

c) Encode the following data value using Hamming ECC: 0b10010001100. **Give your answer in hex.**

Hamming-Encoded value in Hex: __________ 0x590C

d) Suppose that we have a program that runs in 100 ns with 4 processors, but takes only 50ns when run on 8 processors. What term describes the behavior of this program as we increase the number of processors, extrapolating from the given data?

Strong scaling.

e) Which of the following choices correctly describes the given code? (Circle one). Explain your choice.

```
omp_set_num_threads(8);
#pragma omp parallel
{
      int thread_id = omp_get_thread_num();
      for(int count=0; count < ARR_SIZE/8*8; count+=8){
            arr[(thread_id%4)*2 + (thread_id/4) + count] = thread_id;
      }
}
```

A) Always correct, slower than serial
B) Always correct, speed depends on caching scheme
C) Always correct, faster than serial
D) Sometimes incorrect
E) Always incorrect

B)

Explain:

We have false sharing. Data is being written to adjacent areas of memory (in the integer array arr, so multiple threads will be updating the same cache Block at omce, leading to false sharing as the mmu has to keep all the block up to date in every cache.

a) All integers of magnitude <= 2^24 - 1 are representable, because we can make any binary string of length 24 (implicit one along with 23 mantissa bits), and then shift the decimal place anywhere, including all the way to the right of those 24 significant bits (make exponent = 23 + 127 = 150 to shift decimal to the right of 23 bit mantissa).
However, this is not to say that no integers greater than or equal to 2^24 can be represented. Clearly, we can also choose to shift the decimal point further to the right than 23 places. For example, we can make our exponent 151, and shift the decimal place 24 places to the right, and also set our significand to be 000...01, we can make the value 2^24 + 2. We note that 2^24 + 1 cannot be represented. In fact, we can represent no odd number greater than 2^24, because that would require a 1 to end up in the 2^0 position after shifting the decimal 24 positions or more to the right, but that's impossible given that the mantissa is 23 bits long. So, the first multiple of 5 we cannot represent is the first even multiple of 5 (multiple of 10) that's greater than or equal to 2^24. This is 2^24 + 9.

b)
2^26 has exponent 153, 2^27 has exponent 154, and both have mantissa of all 0s. You can represent all the numbers between 2^26 and 2^27 which have exponent 153 and any significand. These are all the integers which have 0s in the 3 least-significant bit positions if the integer is written as an unsigned binary number. Those integers which would have a 1 in at least one of the 3 least-significant positions are not representable, because there are not enough significand bits. Thus, 1/8 of the integers between 2^26 and 2^27 are representable.

c)
We proceed according to the Hamming encoding scheme/table. 11 bits means we need 4 parity bits in the following positions.
P P 1 P 0 0 1 P 0 0 0 1 1 0 0
1st partity bit needs to make P + 1 + 0 + 1 + 0 + 0 + 1 + 0 even, so P1 = 1
substitute P1
1 P1 P0 01 P0 00 11 00
2nd parity bit needs to make P + 1 + 0 + 1 + 0 + 0 + 0 + 0 even, so P2 = 0
substitute P2
101 P001 P000 1100
3rd parity bit needs to make P + 0 + 0 + 1 + 1 + 1 + 0 + 0 even, so P4 = 1
substitute P4
1011001 P0001100
4th parity bit needs to make P + 0 + 0 + 0 + 1 + 1 + 0 + 0 even, so P8 = 0
substitute P8 to get full ECC encoded string 101 1001 0000 1100 = 0x590C

F3) Potpourri (continued)

SID: ______________

f) In class, we discussed the different types of RAID. Fill in the blanks with **all of the following options that apply**. Options can be reused for each question:

A) RAID1 B) RAID3 C) RAID4 D) RAID5 E) None

i) A single write can require 2 reads and 2 writes b, c, d - if data is striped, read data → read Parity → write data → write parity

ii) Can execute independent writes in parallel: D - ex: two writes to 4 disks may use each disk Just once (disk1 gets data1, disk2 parity1, disk 3 data2, disk 4 parity2)

iii) Uses block striping: c, d

g) In RAID5, what logical gate is used to determine if new data does not match old data?

XOR, if XOR(data1, data2) != 0, data changed

h) Consider attaching a hard disk to a CPU. Should we use polling or interrupts to deal with the hard disk? Should we use a DMA engine? Explain.

We should use DMA + interrupts. Since we transfer data in large chunks (pages), it makes sense to give control to another program when we get a page fault and let the DMA engine do the page transfer from disk to memory. We then receive an interrupt when the transfer is completed.

i) (1pt) **Extra Credit:** What does the Cisco logo represent?

The Golden Gate Bridge