University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2015 Instructors: Krste Asanović, Vladimir Stojanovic 2015-05-15

☹ CS61C FINAL ☺

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

Last Name	
First Name	
Student ID Number	
CS61C Login	`cs61c-`
*The name of your **SECTION** TA (please circle)*	David | Donggyu | Fred | Jeffrey | Martin Nolan | Sagar | Shreyas | William
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 booklet contains 14 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.**
- 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 180 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use three handwritten 8.5"x11" pages (front and back) of 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.
- **You must complete ALL THE QUESTIONS, regardless of your score on the midterm.** Clobbering only works from the Final to the Midterm, not vice versa. You have 3 hours... relax.

	M1-1	M1-2	M1-3	M2-1	M2-2	M2-3	M2-4	F1	F2	F3	Total
Points Possible	9	9	9	10	4	8	12	9	10	10	90
Points Earned											

SID: ________________

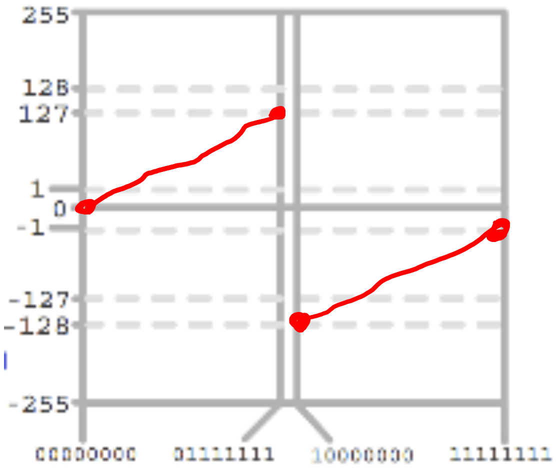
M1-1: *I smell a potpourri section covering midterm one…* (9 points)

a) Which of the following number representations give **0xFFFFFFFE** the **most positive** value when converted to decimal?

Explanation on NEXT Page

A) Bias (with standard bias) [✓ B) Unsigned] C) Two's complement D) Sign and Magnitude

b) Consider a plot that shows the mapping between 8-bit two's complement binary numbers and their decimal equivalents (i.e. binary is on the x-axis and decimal is on the y-axis). Fill in the plot to the left and answer the following questions.



i) Fill in the plot to the left.

Translate the numbers on the x-axis to decimal

ii) Describe (in binary) where discontinuities occur in the plot, if any:

The discontinuity occurs between 01111111 to 10000000, as shown by the graph.

iii) What are the most positive and most negative decimal values that this representation can store?

Most Positive: 127

Most Negative: -128

c) Consider the C code below. Indicate where the values on the right live in memory (using **(S)**tack, **(H)**eap, s**(T)**atic, or **(C)**ode). Assume no registers are used:

```
#define a 10
int b = 0;

int main(int argc, char** argv) {
   int c = a;
   char d[10];
   int* e = malloc(sizeof(int));
}
```

Explanation on next Page

a: C

b: T

***d:** S

***e:** H

e: S

d) Convert the following instructions from TAL to hex or vice versa. Use register names when possible.

i) lw $s0, 0($a0) → consider lw s0, 0(a0) I-Type

- opcode : 000 0011
- rd : s0 → 8 → 0 1000
- funct3: 010
- rs1 : a0 → 10 → 01010
- imm: 0

⇓

0000 0000 0000 0101 0010 0100 0000 0011

$\boxed{0x00052403}$

~~ii) 0x02021021~~

IGNORE, old question

1A)

In both signed and two's complement, the number 0x FFFFFFFE is negative. In unsigned format, 0xFFFFFFFFFE $=2^{32}-2$

In Biased format, the bias will be $2^{31}-1$, so 0xFFFFFFFE $=(2^{32}-2)-(2^{31}-1)$.

This is less than our unsigned representation, yielding our solution.

1C)

a → this is a macro, so the symbol "a" is replaced by 10 by the compiler. Thus, "a" is stored in the code section.

b → this is a global variable, so it is stored on static

*d → d is an array that is locally allocated (within the "main" function), so d refers to a block of memory on the stack

*e → e is a local variable, but its value is an address on the heap. Thus, when we doreference e, the block of memory e refers to is on the heap.

e → e itself is a local variable, so it is located on the stack

M1-2: *I'll believe it when I C it* (9 points)

Your friend wants to take 61C next semester, and is learning C early to get ahead. They try to implement the ROT13 function as practice, but the code they wrote has some bugs, so you've been called in, as the C expert, to help debug their program, reproduced below:

```
 1  /* Applies the ROT13 cipher. rot13("happytimes") == "uncclgvzrf" */
 2  void rot13(char *str) {
 3      while (*str) {
 4          if (str >= 'a' && str <= 'z') {
 5              *str = (*str + 13) % 26;
 6          }
 7          str++;
 8      }
 9  }
10
11  int main(int argc, char *argv[]) {
12      char a[] = "happy";
13      char b[] = "times";
14      char *s = "XXXXXXXXXXXX"; // 12 X's
15
16      // Apply cipher to a and b.
17      rot13(a);
18      rot13(b);
19      printf("%s%s\n", a, b);
20
21      // Concatenate and place in s.
22      int i = 0;
23      for (int j = 0; a[j]; ) s[i++] = a[j++];
24      for (int j = 0; b[j]; ) s[i++] = b[j++];
25
26      printf("%s\n", s);
27  }
```

we are comparing an address to a number; these Should be *str instead

we want this to be a letter, so the ascii should be in the range ['a', 'z']. Taking the number mod 26 will not achieve this.

5 characters

5 characters

↳ s is a pointer to a static string, so it cannot be mutated

→ makes a refer to "unccl" (If rot13 works)

→ makes b refer to "gvzrf" (If rot13 works)

ERROR: mutating the String in read-only static memory

If s is longer than len(a) + len(b), then there will be residue characters here unless we manually insert a NUL Byte

a) You want to impress your friend, so you predict the result of executing the program as it is written, just by looking at it. If the program is guaranteed to execute without crashing, describe what it prints, otherwise explain the bug that may cause a crash.

The program will crash on line 23 when it tries to modify the string literal referred to by S.

b) Now, fix all the errors in the program so that it executes correctly. Fill in the corrections you made in the table below. You may not need all the rows.

Line #	Insert Before / Replace / Delete	Change (Explanation or Code)
4	Replace	*str >= 'a' && *str <= 'z'
5	Replace	*str = (*str - 'a' + 13) % 26 + 'a'
14	Replace	char s[] = "XXXXXXXXXXXX"
25	Insert	s[i] = '\0'

M1-3: *I don't want to MIPS a thing* (9 points)

The following C code recursively sums the elements in an array of length n.

```
int32_t sum_arr(int32_t *arr, size_t n) {
    if (n) {
        return sum_arr(arr + 1, n – 1) + arr[0];
    }
    return 0;
}
```

Translate sum_arr into MIPS below. Your code must follow all function calling conventions, and you may not use any pseudoinstructions. You may not need every blank.

RISC-V answers on next page

```
sum_arr:    _____________ non_zero
            addu $v0, $0, $0
            jr $ra
non_zero:   _____________________
            _____________________
            _____________________
            _____________________
            addiu $s0, $a0, 0       # Store arr into $s0
            _____________________
            _____________________
            _____________________
            _____________ sum_arr   # Make the recursive call
            _____________________   # Then add arr[0] to the result
            _____________________
            _____________________
            _____________________
            _____________________
            _____________________
            jr $ra
```

```
sum_arr:   bne   a0, x0, non_zero
           addi  a0, x0, 0          } base case
           jr    ra
non_zero:  addiu sp, sp, -8
           sw    s0, 0(sp)          } Backup saved register & return addr,
           sw    ra, 4(sp)            as we are making a function call
           addi  s0, a0, 0          # back up arr
           addi  a0, a0, 4          } Prepare parameters (arr is an int array,
           addi  a1, a1, -1                                so we do +4)
           jalr  ra, sum_arr        # recursive call
           lw    t0, 0(s0)          # get arr[0]
           add   a0, a0, t0         # add output of recursive call to arr[0],
                                      put in return value register
           lw    s0, 0(sp)
           lw    ra, 4(sp)          } Restore stack and
           addi  sp, sp, 8            get back return address
           jr    ra
```

M2-1: *I couldn't come up with a clever title for SDS.* (10 points)

a) Give the simplest Boolean expression for the following circuit in terms of A and B, using the minimum number of AND, OR, and NOT gates:

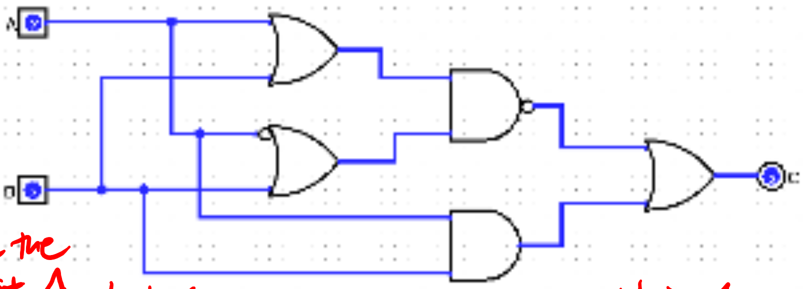
$\overline{(A+B)(\bar{A}+B)} + (AB) \leftarrow$

$= \overline{A+B} + \overline{\bar{A}+B} + AB$ (De Morgan's)

$= \bar{A}\bar{B} + A\bar{B} + AB$ (De Morgan's)

$= \bar{B}(\bar{A}+A) + AB$

$= \bar{B} + AB = A + \bar{B}$ This is because If B is 1, the the expression is just A, but if B is 0, then the statement is 1



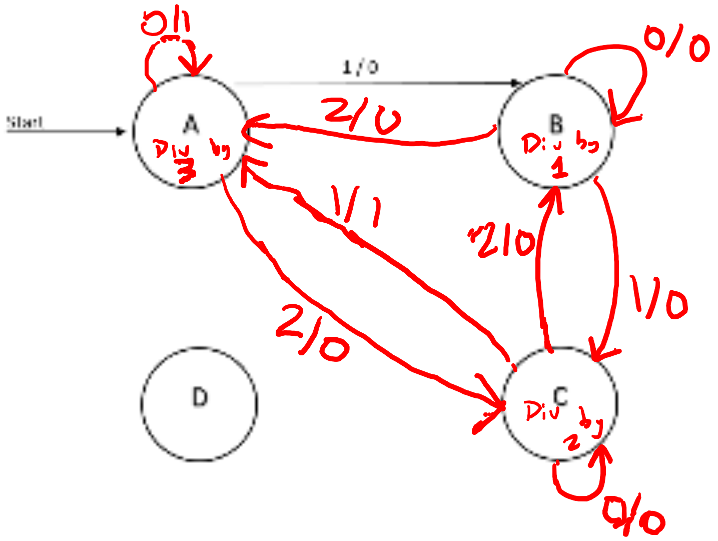
C = $A+\bar{B}$ ____________

(You must show your work above to earn points.)

b) Using as few states as possible, complete the transition table for an FSM that takes an input with 3 values: 0, 1, or 2. The machine will output a 1 when the sum of the inputs seen so far is divisible by 3. Otherwise it should output a 0.

Assume you have seen no digits at the start state. You might not need all of the states, and you should not draw additional states. You must represent your FSM using the table to the left, **the table is the only part that will be graded.** The first transition has been filled in for you.

Current State	Input/Output	Next State
A	1/0	B
A	0/1	A
A	2/0	C
B	0/0	B
B	1/0	C
B	2/1	A
C	0/0	C
C	1/1	A
C	2/0	B



A means: sum is divisible by 3

B means: Sum is divisible by 1

C means: sum is divisible by 2

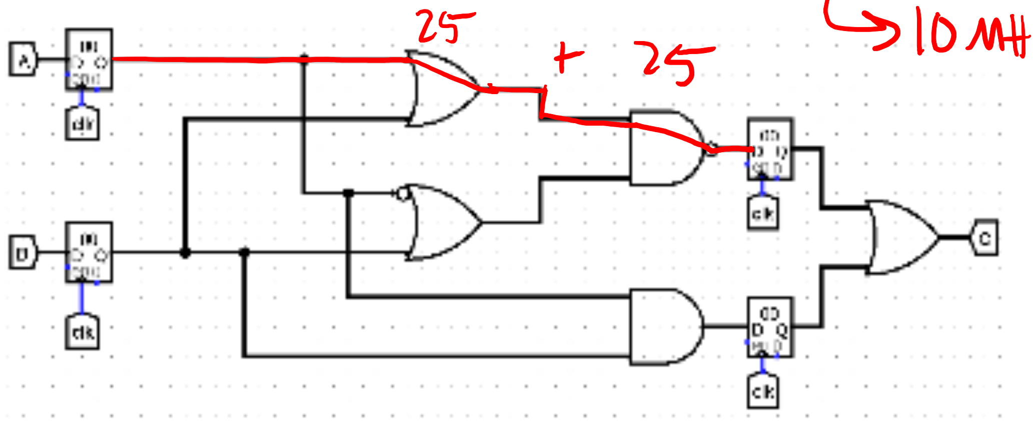
D means: unused

SID: ________________

M2-1: (continued)

c) Suppose we add registers to the unoptimized circuit in part A to increase the clock rate (this modification is shown below). What is the longest clock-to-Q that the registers on inputs A and B can have that will result in correct behavior when the circuit is clocked at 10 MHz?

→ 10 MHz = 100 ns



Assumptions:

- Assume that clock-to-Q > hold time
- All registers have a setup time of 2 ns
- All logic gates have a delay of 25 ns
- Bubbles on gates do not introduce additional delay

$t_{CLK} = 100_{ns}$

$t_{LOGIC} = 50\text{ns}$ b/c critical path has 2 logic gates (as shown on path above)

$t_{CLK} >= t_{CLK\text{-}to\text{-}Q} + t_{LOGIC} + t_{setup}$

$100 >= t_{CLK\text{-}to\text{-}Q} + 50 + 2$

$t_{CLK\text{-}to\text{-}Q} <= 48\text{ns}$ → max is 48ns Answer: 48ns

M2-2: Float like a butterfly and sting like an IEEE (4 points)

Let's take another look at the IEEE754 standard for single-precision floating-point numbers. [x, y) represents a range where x is included and y is not.

a) How many floats are representable in the interval [0.5, 1)? Answer: 1 · 2^23

All numbers like $0.1\underbrace{XXXX\cdots X}_{23}$ → each 'X' can be 0 or 1, so we have 2^{23} in total

significand field bits

b) How many floats are representable in the interval [0, 0.5)? Answer: 126 · 2^23

Denorm numbers → 2^{23} of these for each bit of significand field being 0 or 1

Any normalized float of form → $1.XXXX\cdots X \cdot 2^n$ for $n \geq 2$

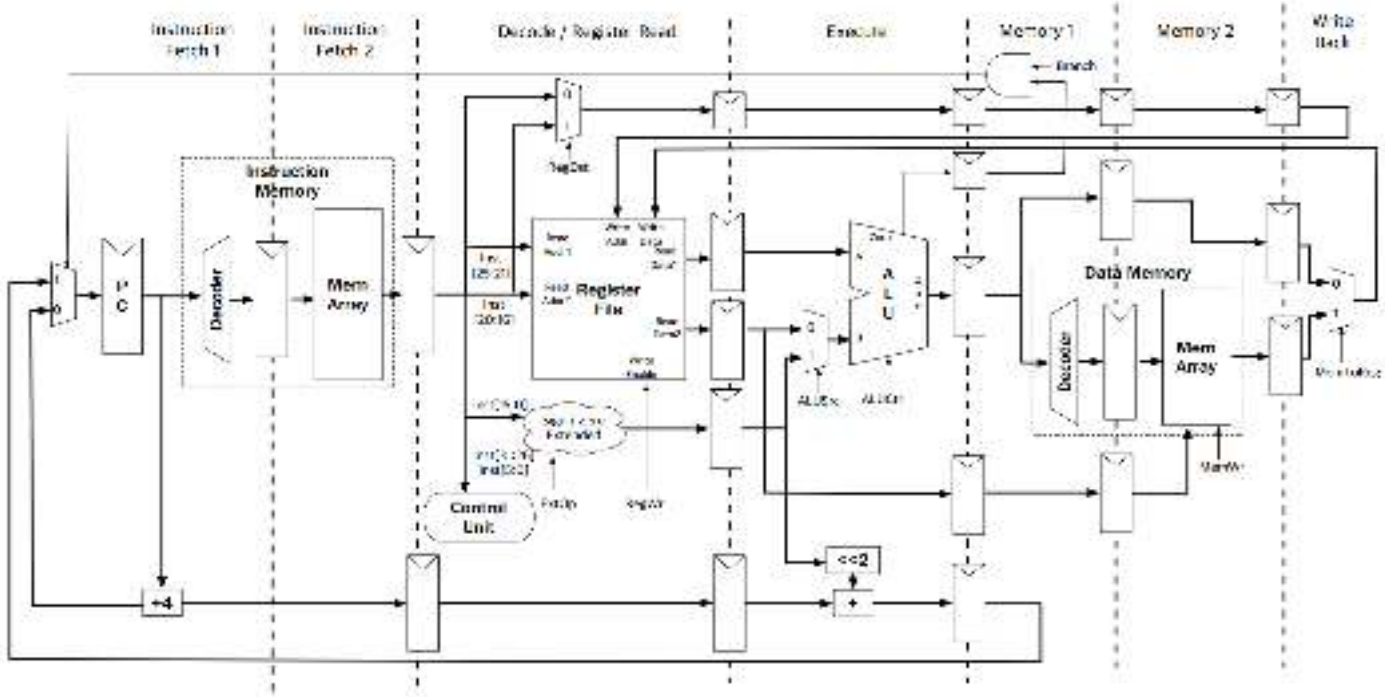
As maximum exponent is 127, we have 125 such values of n, and each has 2^{23} value (each significand bit can be 0 or 1)

Thus, we have $(1 \cdot 2^{23}) + (125 \cdot 2^{23}) = 126 \times 2^{23}$

MIPS Question, IGNORE

M2-3: If this exam were a CPU, you'd be halfway through the pipeline (8 points)

We found that the instruction fetch and memory stages are the critical path of our 5-stage pipelined MIPS CPU. Therefore, we changed the IF and MEM stages to take **two** cycles while increasing the clock rate. You can assume that the register file is written at the falling edge of the clock.



Assume that no pipelining optimizations have been made, and that branch comparisons are made by the ALU. Here's how our pipeline looks when executing two add instructions:

Clock Cycle #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
add $t0, $t1, $t2	IF1	IF2	ID	EX	MEM1	MEM2	WB	
add $t3, $t4, $t5		IF1	IF2	ID	EX	MEM1	MEM2	WB

Make sure you take a careful look at the above diagram before answering the following questions:

a. How many stalls would a data hazard between back-to-back instructions require?

b. How many stalls would be needed after a branch instruction?

c. Suppose the old clock period was 150 ns and the new clock period is now 100ns. Would our processor have a significant speedup executing a large chunk of code…

   i. Without any pipelining hazards? Explain your answer in 1-2 sentences.

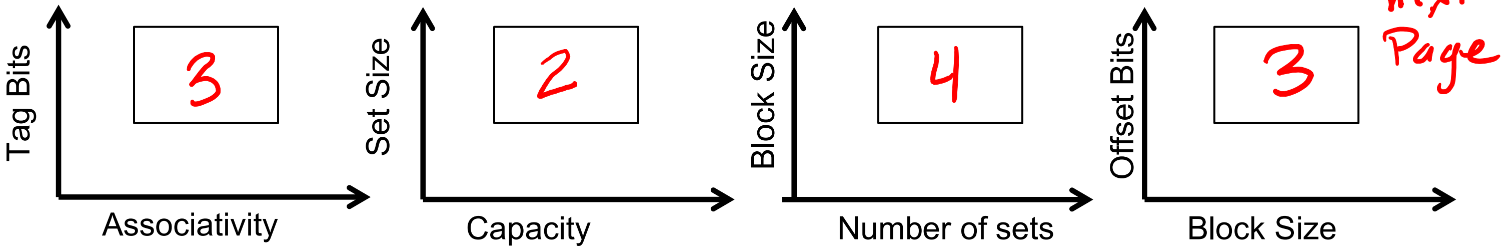
   ii. With 50% of the code containing back-to-back data hazards? Explain your answer in 1-2 sentences.

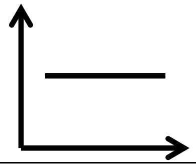
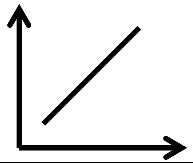
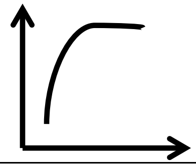
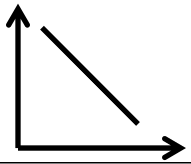
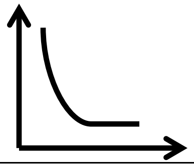
SID: ________________

M2-4: Some say there’s nothing better than cold, hard cache (12 points)

a) What shape do the following trade-off curves have? Select a shape and enter its number into the box for each of the graphs. Unless they are the parameters being varied, assume that associativity, capacity and block size are constant. You should assume that the axes are linear.

Explanations on next Page



				
1: Constant	2: Linear	3: Logarithmic	4: Reverse linear	5: Constant – Logarithm

b) Consider a system with inclusive L1 and L2 caches with 4B cache block size. Assume we have 1 MiB of on-chip memory available and want to determine how much of this memory we should give to the L1 cache and how much to the L2 cache. We will try to minimize the AMAT to do so.

Assume both caches are fully associative with LRU replacement. Their combined capacity is 1MiB (excluding tags and meta-data). **You can consider all miss rates approximate.**

Say you are running the following program starting from cold L1 and L2 caches:

```
#define ARRAY_SIZE 256*1024
int a[ARRAY_SIZE];
int sum = 0; // assume sum, i, and j are stored in registers

for (int i = 0; i < 100000; i++) {
  for (int j = 0; j < ARRAY_SIZE; j++) sum += a[j];
  for (int j = ARRAY_SIZE-1; j >= 0; j--) sum += a[j];
}
```

1) How would we compute AMAT if we had the local L1 miss rate (“L1Miss”), the local L2 miss rate (“L2Miss”) and the memory access time (“Memory”)? Use “H1” and “H2” to represent the L1 and L2 hit times respectively. (We will compute these quantities later in the question)

AMAT = H1 + L1Miss(H2 + L2Miss(Memory))

Just use the original AMAT formula with the variables here a)

Tag Bits – Associativity: The number of tag bits increases by 1 every time the associativity doubles, as doubling the associativity halves the # of sets, thus decreasing the # of index bits by 1. This decrease leads to 1 more tag bit. This relationship is logarithmic.

Capacity – Set Size: Assuming the # of sets (rows) remains constant, doubling the cache capacity will make each set twice as large because we cannot add sets to our cache. This is a linear relationship.

Number of Sets – Block Size: If we double the number of sets, but keep the overall capacity and assoc. the same, we must have half as much data per set. This equates to halving the block size, creating a neg. linear relationship.

Offset Bits – Block Size: We have $2^{\text{offset bits}} = \text{Block Size}$ by definition, so offset bits = $\log_2(\text{Block Size})$

M2-4: (continued)

2) For the program above, express the local miss rate for the L1 cache in general terms as a function of the L1 cache size (write L1 for the size of L1 in bytes). Hint: The miss rate is 0 for a 1 MiB cache, 0.5 for a 0.5 MiB cache and 1 for a 0 MiB (i.e., no) L1 cache.

$$L1Miss = 1 - \frac{L1}{1MiB}$$

If we have 1MiB L1 cache, the all memory is L1, so we always hit. As we decrease this size, our miss rate increases proportional to the %-age of the 1MiB memory we have in L1.

3) What is the global miss rate for the L2 cache as a function of the L1 cache size? Hint: Start by expressing the global miss rate as a function of the L2 cache size.

$$GlobalMiss2 = 1 - \frac{L2}{1MiB} = \frac{1MiB - L2}{1MiB} = \frac{L1}{1MiB}$$

Global miss rate is %-age of ALL accesses that miss at L2. Same as frac of mem NOT in L2.

1MiB − L2 = L1 b/c L1+L2=1MiB

4) What is the local miss rate for the L2 cache as function of the L1 and L2 sizes? Hint: Use your results from questions 2 and 3.

all accesses that miss at L2

$$Local\ Miss2 = \frac{GlobalMiss2}{L1Miss} = \frac{L1/1MiB}{1 - L1/1MiB} = \frac{L1}{1MiB - L1} = \frac{L1}{L2}$$

all accesses that get to L2

L1+L2=1MiB

5) Assume the hit time of the L1 cache is 10 cycles, the hit time of the L2 cache is 20 cycles and the memory access time is 100 cycles. Using the formula from question 1, what is the AMAT for this system as a function of only the L1 size?

$$1 - \frac{L1}{1MiB} = \frac{1MiB - L1}{1MiB} = \frac{L2}{1MiB}$$

$$AMAT = 10 + \left(\frac{L2}{1MiB}\right)\left(20 + \frac{L1}{L2} \cdot 100\right)$$ (from parts 2 & 4)

$$= 10 + \frac{20 \cdot L2}{1MiB} + \frac{L1}{1MiB} \cdot 100 = 10 + \frac{20MiB - 20L1}{1MiB} + \frac{100L1}{1MiB}$$

$$= 10 + 20 + \frac{100L1 - 20L1}{1MiB} = \boxed{30 + \frac{80L1}{1MiB}}$$

6) What sizes of L1 and L2 caches should we pick to minimize the AMAT? (assume the caches have non-zero size, i.e., both of them exist)

We want $30 + \frac{80L1}{1MiB}$ to be small, so L1 must be as small as possible. Thus, the answer is L1: 4B, L2: 1MiB−4B, as neither is zero and both have blocks of 4B (min size is 4B).

F1: *Paging all CS61C students* (9 points)

Consider a byte-addressed machine with a 13-bit physical address space that can hold two pages in memory. Every process is given 16MiB of virtual memory and pages are evicted with an LRU replacement scheme.

Solve in order of blue circles

Virtual Addresses are $\log_2(16MiB) = \log_2(2^{24}) = 24$ bits

a) What are the sizes of the following fields in bits?

VPN = Addr Size − offset = 24 − 12

4 Virtual Page Number: 12   3 Virtual Address Offset: 12

pages are same size

PPN bits = $\log_2(2) = 1$, as we have 2 pages

1 Physical Page Number: 1   Physical Address Offset: 12

13 bit addr − 1 bit PPN

b) Consider the following code snippet: 2

```
// a and b are both valid pointers to
// different arrays of length ARRAY_SIZE
void enumerate(int* a, int* b) {
     for (int i = 0; i < ARRAY_SIZE; i++) {
          a[i] = i;
          b[i] = ARRAY_SIZE - i;
     }
}
```

The compiled binary for the program containing this code snippet weighs in at 4096B. If this code was executed on the machine, what is the maximum value of `ARRAY_SIZE` that would allow this code to execute with 0 page faults in the best-case scenario? (Answer in IEC prefix: 8Gi, 32Ti, etc)

Each page is $2^{12} = 4096B$. Thus, the code fits in one page, so to have no faults, the data must be in the other. Each array is the same size, so $2 * ARRAY\_SIZE \le PAGE\_SIZE \Rightarrow$ max

`ARRAY_SIZE` = 512   arr size is $\frac{4096B}{2} = 2048B \Rightarrow 512$ ints

c) How could we modify the above code snippet to allow a larger `ARRAY_SIZE` and execute with the fewest page faults in the best-case scenario? Write the new code below:

```
for (int i = 0; i < ARRAY_SIZE; i++) {
   a[i] = i;
}
for (int i = 0; i < ARRAY_SIZE; i++) {
   b[i] = ARRAY_SIZE - i;
}
```

We can make ARRAY_SIZE = page size because we no longer worry about a & b knocking each other, or the code, out of memory

F2: *Why can't you use parallelism at a gas station? It might cause a spark.* (10 points)

1. Optimize `factorial()` using SIMD intrinsic(AVX).

```
double factorial(int k) {
  int i;
  double f = 1.0;
  for (i = 1 ; i <= k ; i++) {
    f *= (double) i;
  }
  return f;
}
```

You might find the following intrinsics useful:

`__m256d _mm256_loadu_pd(double *s)`	returns vector(s[0], s[1], s[2], s[3])
`void _mm256_store_pd(double *s, __m256d v)`	stores p[i] = v_i where i = 0, 1, 2, 3
`__m256d _mm256_mul_pd(__m256d a, __m256d b)`	returns vector(a_0b_0, a_1b_1, a_2b_2, a_3b_3)

```
double factorial(int k) {
  int i, j;
  double f_init[] = {1.0, 1.0, 1.0, 1.0};
  double f_res[4];                          initialize to all 1's (multiplicative
  double f = 1.0;                                                  identity)
  // initialize f_vec
  __m256d f_vec = _mm256_loadu_pd(f)

  // vectorize factorial                    ← largest multiple of 4 ≤ k
  for (i = 1 ; i <= 4*(k/4) ; i+=4 ) {
    double l[] = {
      (double) i , (double) i+1 ,

      (double) i+2 , (double) i+3 };

    __m256d data = _mm256_loadu_pd(l);
    f_vec = _mm256_mul_pd(f_vec, data);    }
  // reduce vector
  _mm256_store_pd(f_res, f_vec); → put result into f_res array
  for (j = 0 ; j < 4 ; j++) {
    f = f * f_res[j] ;
  }                 ↖ reduce into f
  // handle tails
  for ( ; i <= k ; i++) {
    f *= (double) i
  }           ↖ manually handle tail case
  return f;
}
```

SID: ________________

F2: (continued)

2. Cache Coherence:
We are given the task of counting the number of even and odd numbers in an array, A, which only holds integers greater than 0. Using a single thread is too slow, so we have decided to parallelize it with the following code:

```
#include <stdio.h>
#include “omp.h”
void count_eo (int *A, int size, int threads) {
    int result[2] = {0, 0};
    int i,j;

    omp_set_num_threads(threads);

    #pragma omp parallel for
    for (j=0; j<size; j++)
        result[(A[j] % 2 == 0) ? 0 : 1] += 1;

    printf("Even: %d\n", result[0]);
    printf("Odd: %d\n", result[1]);

}
```

We can pick ANY ordering

These are the assembly ops nec. for the increment

Thread 0	Thread 1
lw ①	lw ③
add ②	add ④
sw ⑤	sw ⑥

If we execute in the order shown above, then both threads load the old value of res[0] before incrementing instead of going sequentially.

As we increase the number of threads running this code:

a) Will it print the correct values for Even and Odd? If not, explain the error.

Not always. Consider the case where we have 2 threads, and each has an index that corresponds to an even #. See the explanation above.

b) Can there be false sharing if the cache block size is 8 bytes?

Yes, the result array can be stored in a single block, So writing to any element will remove the block from all other caches.

c) What about 4 bytes?

No, false sharing is only applicable when the data being accessed by the threads are distinct. If the block is 4 bytes, then a single element of result is in a cache block, so we would invalidate blocks in other caches ONLY if they had the SAME element of the result array. Thus, false sharing is technically not possible.

SID: ________________

F3: *This isn't a bathroom. Why is there potpourri?* (10 points)

1. **T/F Questions (Circle one. If the circling is unclear, you will receive no credit.)**

1) CPUs need separate instructions to access I/O devices. (True / [✓ False]) consider memory mapped I/O
2) Segmentation (base + bound) has fragmentation problems. ([✓ True] / False) contiguous, large blocks per process wastes memory fragmented
3) Exceptions in early pipeline stages override exceptions in later stages for a given instruction. ([✓ True] / False) Exceptions in early stages can cause problems later on
4) Exceptions are handled in the pipeline stage where they occur. (True / [✓ False])

Done by exception handler

2. **Polling, Interrupts, and DMA**

1) Choose polling or interrupt for the following devices.

Device	Data Rate	Transfer Block Size	Polling/Interrupt?
A	80B/s	4B	Polling
B	400MB/s	4B	Interrupt
C	400MB/s	2KB	Interrupt

2) To support interrupts, the CPU should be able to save and restore the current state. Which of the following should be saved before handling interrupts to ensure correct execution?

[✓ a. Program Counter] [✓ b. User Registers] c. TLB d. Caches

We need these 2 to know the state of our process. The others are just for performance optimizations

3) To which device in 1) is direct memory access (DMA) most beneficial? Explain briefly.

C. CPU can do work when transferring large blocks, and DMA enables this.

3. **Warehouse Scale Computing and Amdahl's Law.**

1) We are going to train convolutional neural networks on Amazon EC2. It turns out that 90% of training can be parallelized but the rest takes twice as long due to the communication overhead among the instances. What is the maximum speedup we can achieve?

10% is non parrallelized, so this is doubled 90% is parrallelized, sped up by s

$$\Rightarrow \frac{1}{(0.1*2) + \frac{0.9}{s}} = \frac{1}{0.2+\frac{0.9}{s}} \leq 5 \quad (\text{as } s \to \infty, \text{expr} \to \frac{1}{0.2})$$

2) Which of the following can increase the maximum speedup in 1)?

a. Use more instances
b. Deploy the application across multiple arrays.
[✓ c.] Reduce the number of reduce operations.
[✓ d.] Increase network bandwidth.
e. Increase the capacity of disks.

We need to improve the NON parrallelizable part, which is the communication done. This communication is boosted by a better network OR if we do it less frequently. These options are d & c respectively.

SID: ________________

F3: (continued)

4. **Hamming Error-Correction Code (ECC)**

1) Suppose we have a one-byte data value, 01101101_{two}. Fill in the encoded data in the following table.

Bit	1	2	3	4	5	6	7
Encoded Data							
P1	X		X		X		X
P2		X	X			X	X
P4				X	X	X	X

2) Assume that we have an encoded value, 1001110_{two} with a single-bit error. Indicate below each parity bit if it has an error:

Parity Bit	P1	P2	P4
OK/ERROR	OK	ERROR	ERROR

As both P2 & P4 have errors, there is an error in bit 2+4 = 6 (should be 0)

P1: 1^0^1^0 = 0
P2: 0^0^1^0 = 1
P4: 1^1^1^0 = 1
use table above

Incorrect bit position: ______6______

Correct data: ______0100______

Fixed: 1001100 → 0100 is data

5. **Dependability and RAID**

1) Which of the following can increase the availability?

[✓ a. Increasing MTTF]
b. Decreasing MTTF
c. Increasing MTTR
[✓ d. Decreasing MTTR]
[✓ e. Redundant data copies]

We want to reduce failures OR decrease repair time. a & e follow the first reason, while d follows the second.

2) Explain very briefly why RAID1 is the most expensive form of RAID.

It requires a full copy of disks, so the overhead is 100%.

3) How many check disks are needed for RAID3?

One, RAID3 has a dedicated parity disk and uses byte-striping

4) Explain why RAID5 has a higher write throughput than RAID4.

The check information is distributed across disks, so not a single disk is queried for ALL checks. Thus, the load is balanced and the parity disk bottleneck is mitigated.