

University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2015

Instructors: Vladimir Stojanovic, John Wawrzynek

2015-12-18



CS61C FINAL



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

<i>Last Name</i>	Perfect
<i>First Name</i>	Peter
<i>Student ID Number</i>	
<i>CS61C Login</i>	cs61c-
<i>The name of your SECTION TA (please circle)</i>	Alex Austin Chris David Derek Eric Fred Jason Manu Rebecca Shreyas Stephan William Xinghua
<i>Name of the person to your LEFT</i>	
<i>Name of the person to your RIGHT</i>	
<i>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)</i>	

Instructions (Read Me!)

This booklet contains XX numbered pages including the cover page. After you finish the exam, turn in only this booklet, and not the green sheet or datapath diagram.

- 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. Make sure your solution is on the answer sheet for credit.

	MT1-1	MT1-2	MT1-3	MT1-4	MT2-1	MT2-2	MT2-3	MT2-4	MT2-5
Points Possible	8	8	12	8	9	11	9	12	5

	F-1	F-2	F-3	F-4	Total
Points Possible	9	8	8	11	118

Clarifications during the exam:

MT1 - depth question:

1. **struct node * edges[]** should be
struct node ** edges
2. line in the prologue should read
sw \$ra 0(\$sp)
3. line after the epilogue should read
lw \$ra 0(\$sp)

MT1-4: The label on Line 17 should be L2

MT2-Floating Point: real: bits 8-15. imaginary: bits 0-7

MT2-Clobbering Time:

b. The cache is **direct-mapped**

c. Memory accesses = accesses in the **for loop** at part I

F-1: c. What is the maximum number of **virtual** pages a process can use?

MT1-1: Potpourri - Good for the beginning... (8 points)**a. True/False:**

- i. The compiler turns C code into instructions ready to be run by a processor **F**
- ii. The instruction `addiu $t0 $t1 0x10000` is a TAL instruction **F**
- iii. The linker computes the offset of all branch instructions **F**

b. Memory Management

```
int global = 0;

int* func() {
    int* arr = malloc(10 * sizeof(int));
    return arr;
}

int main() {
    char* str = "hello world";
    char str2[100] = "cs61c";
    int* a = func();
    return 0;
}
```

In what part of memory are each of the following values stored?

***str: static**

str2[0]: stack

a: stack

arr: stack

arr[0]: heap

MT1-2: C-ing images through a kaleidoscope (8 points)

Consider a grayscale image with a representation similar to the one you worked with in Project 4, where the image is represented by a 1-dimensional array of chars with length $n \times n$. Fill out the following function `block_tile`. It returns a new, larger image array, which is the same image tiled `rep` times in both the x and y direction. You may or may not need all of the lines.

For a better idea of what must be accomplished, consider the following example:

```
char *image = malloc(sizeof(char) * 4);
image[0] = 1;
image[1] = 2;
image[2] = 3;
image[3] = 4;
char *tiled_image = block_tile(image, 2, 2);
```

The contents of `tiled_image` would then look like:

```
tiled_image: [1, 2, 1, 2,
              3, 4, 3, 4,
              1, 2, 1, 2,
              3, 4, 3, 4];
```



```

    }
    node->label = counter++;
    return counter;
}

```

Implement `dfs_label` in TAL MIPS. Assume `node` is in `$a0` and `counter` is in `$a1`. You may not need all the lines provided.

```

dfs_label:
# prologue
    addiu $sp $sp -12
    sw $ra ($sp)
    sw $s0 4($sp)
    sw $s1 8($sp)
# base case
    addiu $t1 $0 -1
    lw $t0 4($a0)
    addu $v0 $0 $a1
    bne $t0 $t1 epilogue
# loop
    addu $s0 $a0 0
    addiu $s1 $0 0
loop:
    lw $t0 8($s0)
    beq $t0 $s1 fin
    lw $a0 12($s0)      # load edges into $a0
    sll $t0 $s1 2
    addu $a0 $a0 $t0    # load the next node
    lw $a0 0($a0)     # into $a0
    jal dfs_label
    addu $a1 $v0 $0
    addiu $s1 $s1 1
    j loop
fin:
    sw $a1 4(s0)
    addiu $a1 $a1 1
    addu $v0 $0 $a1
epilogue:
    lw $ra ($sp)
    lw $s0 4($sp)
    lw $s1 8($sp)
    addiu $sp $sp 12
    jr $ra

```

MT1-4: Can't reveal this MIPS-tery (8 points)

```

0| Mystery:
1|         add $t0, $a0, $0
2|         add $t1, $a1, $0
3|
4|         la $s0, L1
5|         lw $s1, 12($s0)
6|         addi $s2, $0, 6
7|         addi $s3, $0, 0
8|         addi $s4, $0, 1057           #$s4 contains 0b0100 0010 0001
9|         sll $s4, $s4, 11
10|
11| L1:     beq $s3, $s2, L2
12|         addu $s1, $s1, $s4
13|         sw $s1, 12($s0)
14|         addu $t1, $a3, $t0
15|         addi $s3, $s3, 1
16|         j L1
17| Done:

```

a. When the above code executes, which line is modified? How many times?

Line 14, 6 times

b. Assume we run this block of code with $\$a0 = 1$ and $\$a1 = 1$; what is the value in $\$t2$ at the end of the code execution? How about $\$t3$?

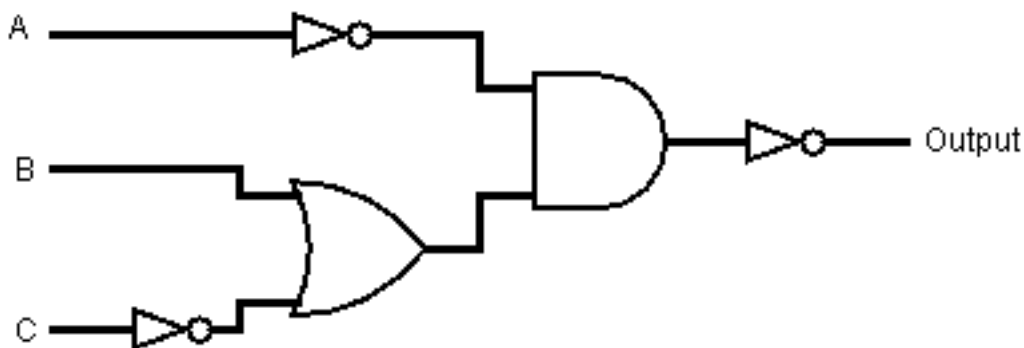
$\$t2 = 2$, $\$t3 = 3$

c. In three sentences or less, how does this code affect the temporary registers?

It takes the arguments $\$a0$ and $\$a1$ and stores them in registers $\$t0$ and $\$t1$. Then for the remaining temporary registers, it sets register $\$t_{\{n\}}$ to $\$t_{\{n-1\}} + \$t_{\{n-2\}}$.

MT2-1: Synchronous Finite State Digital Machine Systems (9 points)

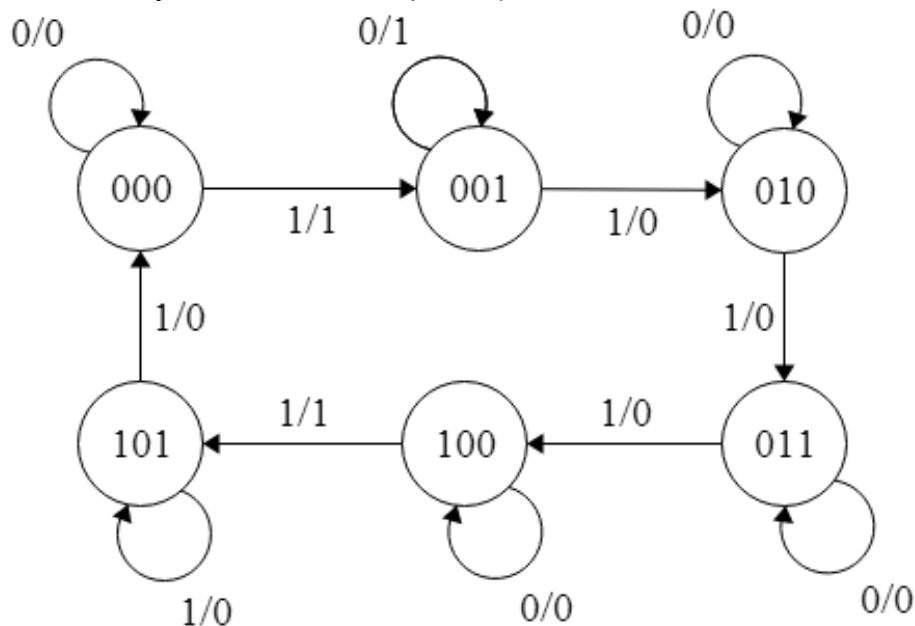
a. The circuit shown below can be simplified. Write a Boolean expression that represents the function of the simplified circuit using the minimum number of AND, OR, and NOT gates.



$$\sim(\sim(A) (B+\sim(C))) = A + \sim(B+\sim(C)) = A + (\sim B)C$$

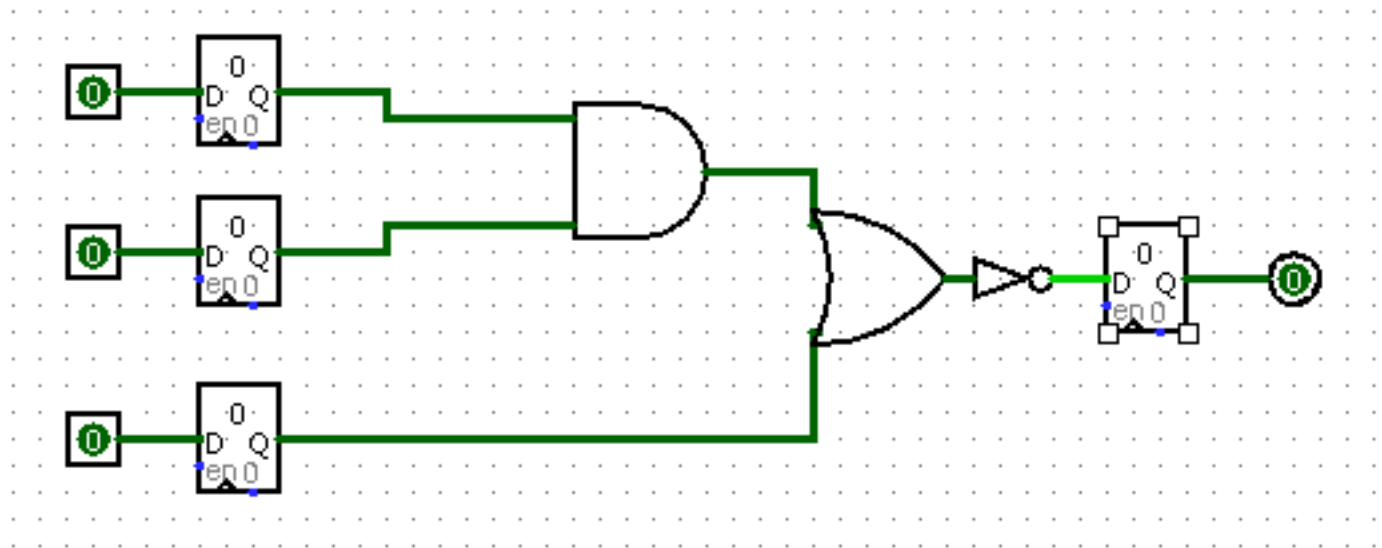
- b. Consider the finite state machine below which has 6 states, and a single input that can take on the value of 0 or 1. The finite state machine should output 1 *if and only if* 6 + the sum of all the input values is not divisible by 2 or 3. One transition has been provided; complete the remainder of the diagram.

(Hint: If the sum of the inputs is a multiple of 6, then we have $6 + \text{the sum of the inputs} = 6n$ for some n . As $6n$ is divisible by 2, $6n$ cannot be prime.)



All transitions going to 001 and 101 should output 1, as we get something in the form of $6n+1$ and $6n+5$ respectively. If you look at the 4 other cases, $6n$, $6n+2$, $6n+3$, $6n+4$, all numbers in these forms are divisible by either 2 and/or 3, and therefore can never be prime for $n \geq 1$ (Which is what we have as we add 6 to the sum of all our inputs). Therefore for all transitions going to 000, 010, 011, and 101, we should output 0.

- c. Consider the following circuit. Assume registers have a CLK to Q time of 60ps, a setup time of 40ps, and a hold time of 30ps. Assuming that all gates have the same propagation delay, what is the maximum propagation delay each individual gate could have to achieve a clock rate of 1GHz.



300 ps

MT2-2: Do stray off of the well-worn *datapath* (11 points)

Suppose we have a new instruction, bmeq. We branch if the value in memory at the address in \$rs equals the value in \$rt. The instruction format is as follows:

bmeq (\$rs) \$rt offset

Use the datapath diagram provided as a reference for input and output names. Assume we are working with a non-pipelined single cycle datapath.

a. Write the register transfer language that represents the logic of this command.

If MEM[\$rs] == \$rt PC = PC + offset << 2; else PC = PC + 4

b. You are given a new control signal, BMEQ, which is 1 when it is a BMEQ instruction and 0 when it is not. In the following table, please fill in the inputs, control signal, and output destination for any additional MUXes you would need in order for this instruction to work correctly. You might not need all the lines.

Inputs: ReadData1, ReadData2, ReadData, AluOut, MemoryData, PC

Output destinations: Addr, ReadReg1, ReadReg2, WriteAddr, InputA, InputB, ReadAddr, WriteData

Control Signal	Input0	Input1	Output Destination
BMEQ	ReadData1	ALUOut	ReadAddr
BMEQ	ReadData1	MemoryData	InputA

d. Fill in the values for the control signals for this new instruction. Use X if the signal does not matter. For ExtOp, write SIGN for sign-extension and ZERO for zero-extension.

Reg Dst	ExtOp	RegWr	ALU Src	ALU Ctr	Mem Wr	Memto Reg	Branch	Jump	BMEQ
X	SIGN	0	0	X	0	X	1	0	1

e. In ≤ 1 sentence, why can't this instruction work with a normal pipelined 5-stage MIPS datapath?
The execute stage now requires a data to be loaded in from memory.

MT2-3: Enough stalling, that will only slow you down (9 points)

Consider the standard 5-stage pipelined MIPS CPU with instruction fetch, register read, ALU, memory, and register write stages. Register writes happen before register reads in the same clock cycle, branch comparison is done during the register read stage, there is a branch delay slot, and forwarding is implemented.

For the following stream of instructions, assume that $\$t0$ is not equal to 0, so the branch is not taken.

```

0| start:   lw $t0 0($a0)
1|         beq $t0, 0, end
2|         addiu $t0, $t0, 10
3|         sw $t0 0($a0)
4| end:
    
```

a. For each pair of instructions, circle whether the CPU needs to be stalled for the execution of the second instruction, and if so, for how many cycles.

```

i. 0| start: lw $t0 0($a0)
    1|      beq $t0, 0, end
stall for 2 cycles
    
```

```

ii. 1|      beq $t0, 0, end
     2|      addiu $t0, $t0, 10
no stall
    
```

```

iii. 2|      addiu $t0, $t0, 10
      3|      sw $t0 0($a0)
no stall
    
```

Logic in each stage of the pipeline has the following timing:

Instruction Fetch	Register Read	ALU	Memory	Register Write
150 ps	100 ps	100 ps	200 ps	100 ps

The pipelining registers in between stages have the following timing:

Clock-to-Q	Hold time	Setup
------------	-----------	-------

30 ps	20 ps	30 ps
-------	-------	-------

b. What is the minimum clock period, in picoseconds, for which the processor can run?

260

c. What is the time required, in picoseconds, that it takes for the CPU, starting from the first stage of the lw instruction, to finish the execution of the final sw instruction? You may use the variable `stall_cycles` in place of the sum of your answers for question a, and `clock_period` in place of your answer for question b.

$(8 + \text{stall_cycles}) * \text{clock_period}$

d. Which timing values, if lowered independently (all other timing remain the same), will allow us to increase the frequency of the CPU? Circle all that apply.

Pipelining Register Clock-to-Q, Pipelining Register Hold time, Pipelining Register Setup time, Instruction Fetch, Register Read, ALU, Memory, Register Write

pipelining register clock-to-q, pipelining register setup time, memory (can be either circled or not circled)

MT2-4: If you do well, it's *clobbering* time! (12 points)

The information for one student in regards to clobbering a single midterm is captured in the data of the following *tightly-packed* struct:

```
typedef struct student {
    int studentID;
    float oldZScore;
    float newZScore;
    int clobber;           /* a value equal to 1 if a student clobbers,
                           0 if otherwise */
} student;
```

We run the following code on a 32-bit machine with a 4 KiB write-back cache. `importStudent()` returns a struct `student` that is in the course roster and that has not been returned by `importStudent()` previously. For simplicity, assume `importStudent()` does not affect the cache.

```
int ARR_SIZE = 512; //Class size rounded down for simplicity
student *61CStudents = (student *) malloc (sizeof(student) * ARR_SIZE);

/* Assume malloc returns a cache block aligned address */
for (int i = 0; i < ARR_SIZE; i++) {                               <=== part I
    61CStudents[i] = importStudent() <- what does import student do?
}

for (int i = 0; i < ARR_SIZE; i++) {                               <=== part II
    if (61CStudents[i].oldZscore > 61CStudents[i].newZscore){
        61CStudents[i].clobber = 0;
    } else {
```

```

        61CStudents[i].clobber = 1;
    }
}

```

a. How many bytes is needed to store the information for a single student?

16 bytes

b. Assume that the block size is 32 B. What is the tag:index:offset breakdown of the cache?

20:7:5

c. At the label **part I**, assume that 61CStudents is filled with the correct data. What type of misses will occur from memory accesses during the process? Why?

Compulsory. New Data; 512 * 16 B >= 4 KiB cache

d. Suppose we run the code again and the cache block size is now 8 B long and the cache is direct-mapped. For the for-loop in **part II**, what is the miss rate in the best case scenario (we want the highest hit rate possible)? What type of misses occur?

Capacity, 2/3

e. For the for-loop in **part II**, assume that the cache block size is now 128B.

i. If the cache is direct-mapped, what is the hit rate?

8 students per block. 3 Memory accesses per student, 1 miss & 23 hits. 23/24.

ii. If the cache is fully associative, what is the hit rate? Does associativity help? Why or why not?

23/24.

MT2-5: What is the floating point of complex numbers? (5 points)

We realize that you want to represent complex numbers, which are in the form $a + bi$, where a is the real component, b is the imaginary component, and the magnitude is $\sqrt{a^2 + b^2}$.

We create a 16-bit representation for storing both the real and imaginary components as floating point numbers with the following form: The first 8 bits will represent the real component, and the latter 8 bits will represent the complex component. Our new representation will look like:

Sign	Exponent	Significand	Sign	Exponent	Significand
15	14-12	11-8	7	6-4	3-0

Bits per field:

Sign: 1

Exponent: 3

Significand: 4

Everything else follows the IEEE standard 754 for floating point, except in 16 bits

Bias: 3

a. Convert 0xB248 into the complex number form $a + bi$.

-1.125 + 3i

b. What is the smallest positive number you can represent with a nonzero real component and zero complex component.

2^{-6}

Recall the following floating point representation from the midterm:

Sign	Exponent	Significand
15	14-9	8-0

Bits per field:

Sign: 1

Exponent: 6

Significand: 9

Everything else follows the IEEE standard 754 for floating point, except in 16 bits

c. Ignoring infinities, which of the two representations presented above can represent a number with the larger magnitude.

The midterm floating point representation can represent 2^6 , whereas complex numbers are strictly bounded by 2^5 .

F-1: You may need to *context switch* for this question (9 points)

The system in question has 1MiB of physical memory, 32-bit virtual addresses, and 256 physical pages. The memory management system uses a fully associative TLB with 128 entries and an LRU replacement scheme.

a. What is the size of the physical pages in bytes?

2^{12} bytes

b. What is the size of the virtual pages in bytes?

2^{12} bytes

c. What is the maximum number of virtual pages a process can use?

2^{20} pages

d. What is the minimum number of bits required for the page table base address register?

20 bits

Everybody Got Choices

e. Answer “Yup!” (True) or “Nope!” (False) to the following questions

i. The page table is stored in main memory

Yup!

ii. Every virtual page is mapped to a physical page

Nope!

iii. The TLB is checked before the page table

Yup!

iv. The penalty for a page fault is about the same as the penalty for a cache miss

Nope!

v. A linear page table takes up more memory as the process uses more memory

Nope!

F-2: Not all optimizations are created equal (8 points)

For this question, you will be looking at several different versions of the same code that has been, or at least has tried to be, optimized. For each of the versions, indicate the correctness and speed with the appropriate letter:

Correctness:

- A. Always Correct
- B. Sometimes Correct
- C. Always Incorrect

Speed:

- A. Faster
- B. Same
- C. Slower

For reference, here is the serial version of the code:

```
#DEFINE RESULT_ARR_SIZE 8
#DEFINE ARR_SIZE 65536

result[0] = 0;
for (int i = 1; i < RESULT_ARR_SIZE; i++) {
    int sum = 0;
    for (int j = 0; j < ARR_SIZE; j++) {
        sum += arr[j] + i;
    }
    result[i] = sum + result[i - 1];
}
```

a. Version 1:

```
result[0] = 0;
#pragma omp parallel
for (int i = 1; i < RESULT_ARR_SIZE; i++) {
    int sum = 0;
    for (int j = 0; j < ARR_SIZE; j++) {
        sum += arr[j] + i;
    }
    result[i] = sum + result[i - 1];
}
```

Correctness: **B**

Speed: **C**

b. Version 2:

```
result[0] = 0;
#pragma omp parallel for
for (int i = 1; i < RESULT_ARR_SIZE; i++) {
    int sum = 0;
    for (int j = 0; j < ARR_SIZE; j++) {
        sum += arr[j] + i;
    }
}
```

```

    }
    #pragma omp critical
    result[i] = sum + result[i - 1];
}

```

Correctness: **B**Speed: **A**

c. Version 3:

```

result[0] = 0;
for (int i = 1; i < RESULT_ARR_SIZE; i++) {
    int sum = 0;
    #pragma omp parallel for reduction(+: sum)
    for (int j = 0; j < ARR_SIZE; j++) {
        sum += arr[j] + i;
    }
    result[i] = sum + result[i - 1];
}

```

Correctness: **A**Speed: **A**

d. Consider the correctly parallelized version of the serial code above.

- i. Could it ever achieve perfect speedup? **No**
- ii. What law provides the answer to this question? **Amdahl's Law**

F-3: Map and Reduce are 2nd degree friends - when you also Combine (8 points)

Imagine we're looking at Facebook's friendship graph, which we model as having a vertex for each user, and an undirected edge between friends. Facebook stores this graph as an adjacency list, with each vertex associated with the list of its neighbors, who are its friends. This representation can be viewed as a list of degree 1 friendships, since each user is associated with their direct friends. We're interested in finding the list of degree 2 friendships, that is, an association between each user and the friends of their direct friends.

You are given a list of associations of the form `(user_id, list(friend_id))`, where the `user_id` is 1st degree friends with all the users in the list.

Your output should be another list of associations of the same form, where the first item of the pair is a `user_id`, and the second item is a list of that user's 2nd degree friends. **Note:** a user is not their own 2nd degree friend, so the list of second degree friends must not include the user themselves.

Write pseudocode for the mapper and reducer to get the desired output from the input. Assume you have a set data structure, with `add(value)` and `remove(value)` methods, where `value` can be an item or a list of items. You can iterate through a list with the `for item in items` construct. You may not need all the lines provided.

```
map(user_id, friend_ids):  
    for friend in friend_ids:  
        emit(friend, friend_ids)  
  
reduce(key, values):  
    second_degree_friends = set()  
    for value in values:  
        second_degree_friends.add(value)  
    second_degree_friends.remove(key)  
    emit(key, second_degree_friends)
```

F-4: Potpourri – ... and good for the end! (11 points)

- a. We have a hard drive with a controller overhead of 5 ms. The disk has 12000 cylinders, and it takes 2 ms to cross 1000 cylinders. The drive rotates at 2400 RPM, and we want to copy half a MB of data. Our hard drive has a transfer rate of 500 MB/s. What is the access time of a read from disk?

$$5 \text{ ms} + 12000/3 * (2/1000) + 1000 * 1/(24000/60/2) + 1000*((1/2)/500) = 15.25 \text{ ms}$$

- b. I launched a new online app at the start of this year (2015), and I want to have at least three nines of availability per year. Up until today, my app has been available at all times this year. However, some malicious hackers crashed my app for today; it took me 4 hours to get it back up again. For the rest of this year, what is the most downtime I can have on my app to meet my availability goals, rounded to the closest hour? (There are 8760 hours this year)

5 hours

- c. If a receiver checks the header and the checksum is correct, what does it do? (In ≤ 1 sentence)

Ack

- d. For the standard single-error correcting Hamming code presented in class, is the 12-bit code word $0x61C$ corrupted? What is the correct data value in decimal format?

$$0x61C = 0b0110\ 0001\ 1100$$

$$P1: 0 \wedge 1 \wedge 0 \wedge 0 \wedge 1 \wedge 0 = 0$$

$$P2: 1 \wedge 1 \wedge 0 \wedge 0 \wedge 1 \wedge 0 = 1$$

$$P4: 0 \wedge 0 \wedge 0 \wedge 0 \wedge 0 = 0$$

$$P8: 1 \wedge 1 \wedge 1 \wedge 0 \wedge 0 \wedge 0 = 1$$

$$0x61C = 0b0110\ 0001\ 1100 \Rightarrow 0b0110\ 0001\ 1000 = 0b1000\ 1000 = 128 + 8 = 136$$

e. True/False

- i. Raid 4 allows for concurrent independent writes to disk. **F**
- ii. Raid 5 allows for concurrent independent writes to disk. **T**
- iii. Raid 5 allows for concurrent independent reads to disk. **T**
- iv. IP guarantees delivery **F**

