CS61C MIDTERM 2

<table>
<thead>
<tr>
<th>Last Name (Please print clearly)</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name (Please print clearly)</td>
<td>Peter</td>
</tr>
<tr>
<td>Student ID Number</td>
<td>12345678</td>
</tr>
<tr>
<td>Circle the name of your Lab TA</td>
<td>Alex     Brian     Jinglin     Nate     Reese     Steven</td>
</tr>
<tr>
<td>Name of the person to your: Left</td>
<td>Samantha Student</td>
</tr>
<tr>
<td>Right</td>
<td>Larry Learner</td>
</tr>
</tbody>
</table>

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 7 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 80 minutes to complete this exam. The exam is closed book: no computers, tablets, cell phones, wearable devices, or calculators. You are allowed one page (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>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>4</td>
<td>56</td>
</tr>
</tbody>
</table>

If you have the time, feel free to doodle on this front page!
Question 1: Synchronous Digital Systems (10 pts)

Consider the circuit below for the following questions. Logical gates incur a 10 ns combinational logic delay. Registers have a CLK-to-Q delay of 5 ns and a setup time constraint of 15 ns.

![Circuit Diagram]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

a) Write the boolean expression for X in terms of A, B, and C. Only use the AND, OR and NOT logical operators, and leave the expression in its UNSIMPLIFIED form. (You are welcome to use the provided truth table, but it is NOT required and NOT graded.) [3 pts]

\[
X = AB + \overline{B}(A + C)
\]

[propagate signals along wires]

\[
X = \overline{A}\overline{B}C + AB\overline{C} + \overline{A}B\overline{C} + ABC
\]

[truth table, then Sum of Products]

\[
X = A + \overline{A}\overline{B}C
\]

[truth table with visual simplification]

b) Given the following boolean expression for a different circuit, simplify it to use the fewest possible AND, OR, and NOT logical operators. [3 pts]

\[
x = \overline{a} \cdot \overline{d} + a \cdot \overline{d} + \overline{b} \cdot \overline{c}
\]

DeMorgan’s Law

\[
x = \overline{d}(a + \overline{a}) + \overline{b} \cdot \overline{c}
\]

distribution

\[
x = \overline{d}(1) + \overline{b} \cdot \overline{c}
\]

complementarity

\[
x = \overline{d} + \overline{b} \cdot \overline{c}
\]

DeMorgan’s Law (3 NOT, 1 AND, 1 OR)

\[
x = \overline{d} + b + \overline{c}
\]

DeMorgan’s Law (2 NOT, 2 OR)

\[
x = \overline{d}(b + c)
\]

DeMorgan’s Law (1 NOT, 1 AND, 1 OR)

c) For the circuit above (part a), calculate the minimum clock period that will allow the circuit to function correctly. Remember to include units. [2 pts]

Minimum clock period (max clock frequency) is the setup time condition. The critical path is through three gates (either B through NOT, AND, OR or C through OR, AND, OR).

We need \( t_{CLK-to-Q} + 3 \times t_{logic} \leq t_{period} - t_{setup} \), so we arrive at \( 5\text{ns} + 3 \times 10\text{ns} \leq t_{period} - 15\text{ns} \).

Solving, we get \( t_{period,min} = 50\text{ns} \).

d) Assuming the hold time constraint of the registers is 20ns, calculate the minimum combinational logic delay needed per logic gate to allow the circuit (part a) to function correctly. Remember to include units. [2 pts]

Shortest path for X to change is A through top AND and then OR.

We need \( t_{hold} \leq t_{CLK-to-Q} + 2 \times t_{logic} \), so we arrive at \( 20\text{ns} \leq 5\text{ns} + 2 \times t_{logic} \).

Solving, we get \( t_{logic,min} = 7.5\text{ns} \).
Question 2: Datapath and Control (10 pts)

Consider adding the new instruction `lw eq` to our existing MIPS datapath. This instruction is intended to load into $rd a word from memory at the address specified by the contents of $rs offset by the shamt value if the values in $rt and $rs match. Below you will find the RTL-esque description as well as a set of regular MIPS instructions that are equivalent to a call to `lw eq`.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
</table>
| `lw eq rd, rs, rt, shamt` | \[
|                       | \[
|                       | \[
|                       | if (R[rs] == R[rt]) \]
|                       | RF[rd] = M[ R[rs] + (SignExt(shamt) << 2) ] \]

\[
\text{lw eq } \text{ } \text{t0 } \text{ } \text{t1 } \text{ } \text{t2 } \text{ } \text{-2} \]

\[
\text{bne } \text{ } \text{t1 } \text{ } \text{t2 } \text{ } \text{skip} \]

\[
\text{l } \text{ } \text{w } \text{ } \text{t0 } \text{ } \text{-8(} \text{t1} \text{)} \]

\[
\text{skip: } \# \text{ next instruction} \]

\[
\text{a) If we want to use the pre-existing R-type instruction format for } \text{lw eq} \text{ and decide to repurpose the shamt field to hold the memory address offset, what is the largest and smallest possible byte offset that can be applied to the value stored in } R[rs]? \quad [1 \text{ pts}] \]

5 bits signed: \(-2^{n-1} \text{ to } 2^{n-1}-1 \) for \( n = 5 \), then shift by 2 (mult by 4)

There was an error in the printed exam on the given example (\(-2(} \text{t1}\) instead of \(-8(} \text{t1}\)), so we accepted both 60/15 and -64/-16.

Most Positive: 60

Most Negative: -64

Based on the MIPS datapath diagram shown below, choose the modifications required in the dashed boxes to execute the new instruction correctly.
b) For box (b), circle the number below for the circuitry required: [3 pts]

1. \[\text{R[rd]}\]
2. \[\text{R[rs]}\]
3. \[\text{lweq}\]
4. \[\text{RegWr}\]

Comparing \(R[rs]\) and \(R[rt]\), so eliminate (1). Normal RegWr doesn’t require equality check (last AND gate), so eliminate (2) and (3). lweq only writes when \(R[rs]\) and \(R[rt]\) are equal and the last OR gate allows normal use of RegWr.

c) For box (c), circle the number below for the circuitry required: [3 pts]

1. \[\text{R[rt]}\]
2. \[\text{R[rt]}\]
3. \[\text{imm}\]
4. \[\text{lweq}\]

Need to sign-extend, not zero, shamt field, so eliminate (2) and (4). ALUSrc is supposed to select between \(R[rt]\) and \(\text{imm}\) for other instructions, so it’s connected to the wrong MUX in (1). (3) has the desired behavior.

d) Based on the MIPS datapath diagram shown on the previous page, fill in the control signal values necessary to execute the \text{lweq} instruction correctly. [3 pts]

<table>
<thead>
<tr>
<th>lweq</th>
<th>RegDst</th>
<th>ExtOp</th>
<th>MEMWr</th>
<th>MemToReg</th>
<th>Jump</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Question 3: Pipelining (11 pts)

Answer the following questions based on the following iterative version of `toLowerCase(char *p)` that we saw on Midterm 1. We are running the code on a 5-stage pipelined MIPS CPU.

```
1 toLower:  addu $v0, $0, $0  # count = 0
2 loop:    lbu   $t0, 0($a0)  # read char using $a0
3       bne   $t0, $zero, label  # check for null terminator
4       jr    $ra    # return
5 label:  ori   $t0, $t0, 0x20  # to lowercase
6       sb    $t0, 0($a0)  # store char
7       addiu $a0, $a0, 1  # move to next char
8       addiu $v0, $v0, 1  # count++
9       j loop           # iterate
10
```

a) If `a0` points to the string “C”, how many instructions are executed just within `toLowerCase` before it returns (but including the `jr`)? Don’t include hazards and nops/stalls/bubbles. [1 pt]

Instructions: 1,2,3,5,6,7,8,9,2,3,4

11 instructions

b) If there were no stalls, how many clock cycles would it take to go through the first iteration of the loop (lines 2-8) for the function call described in part (a)? [1.5 pt]

4 cycles to fill pipeline, then 1 each to complete instr 2,3,5,6,7,8

10 cycles

c) Below are the maximum delays through each of the datapath stages. Fill in the table with the latency and throughput for single-cycle and pipelined CPUs. Don’t forget units. You may use fractions. [2 pts]

<table>
<thead>
<tr>
<th>Stages</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-cycle</td>
<td>1000ps</td>
<td>1 / 1000ps</td>
</tr>
<tr>
<td>Pipelined</td>
<td>1750ps</td>
<td>1 / 350ps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stages</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF:</td>
<td>200 ps</td>
<td>100 ps</td>
</tr>
<tr>
<td>ID:</td>
<td>100 ps</td>
<td>200 ps</td>
</tr>
<tr>
<td>EX:</td>
<td>200 ps</td>
<td>350 ps</td>
</tr>
<tr>
<td>MEM:</td>
<td>350 ps</td>
<td>150 ps</td>
</tr>
<tr>
<td>WB:</td>
<td>150 ps</td>
<td></td>
</tr>
</tbody>
</table>

d) Next to each line below, write “Y” if it causes a data hazard (i.e. requests the data) even with forwarding, “M” if it causes a data hazard only without forwarding, or “N” otherwise: [2.5 pts]

Line 2 __N__  Line 3 __Y__  Line 4 __N__  Line 6 __M__  Line 9 __N__

e) If our CPU implements the jump delay slot, circle ONE line below that you could move into the delay slot below Line 9: [1 pt]

Line 2  Line 6  Line 7  None

f) We decide to use branch prediction of always not taken, but find that `toLowerCase` executes a lot of `nop` instructions for long input strings. Describe changes to our code to improve the performance using the lines below. [3 pts]

Line __3__  Change to beq $t0,$zero,label
Line __4__  Move label above / below Line __9__
Line __10__
Question 4: Caches (11 pts)

We have a 64 KiB address space and two possible data caches. Both are 1 KiB, direct-mapped caches with random replacement and write-back policies. Cache X uses 64 B blocks and Cache Y uses 256 B blocks.

a) Calculate the TIO address breakdown for Cache X: [1.5 pts]

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

b) During some part of a running program, Cache Y’s management bits are as shown below. Four options for the next two memory accesses are given (R = read, W = write). Circle the option that results in data from the cache being written to memory. [2 pts]

<table>
<thead>
<tr>
<th>Slot</th>
<th>Valid</th>
<th>Dirty</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1000 01</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>0101 01</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1110 00</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0000 11</td>
</tr>
</tbody>
</table>

(1) R 0x4C00, W 0x5C00 (R then W into slot 00)
(2) W 0x5500, W 0x7A00 (W into dirty slot 01 – tag matches, W into slot 10)
(3) W 0x2300, R 0x0F00 (W into slot 11, then kick dirty block out)
(4) R 0x3000, R 0x3000 (2 reads into non-dirty slot 00)

c) The code snippet below loops through a character array. Give the value of LEAP that results in a Hit Rate of 15/16 for Cache Y. [4 pts]

```c
#define ARRAY_SIZE 8192
char string[ARRAY_SIZE]; // &string = 0x8000
for(i = 0; i < ARRAY_SIZE; i += LEAP)
    string[i] |= 0x20; // to lower
```

Access pattern is R then W for each address. To get a hit rate of 15/16, need to access exactly 8 addresses per block (compulsory miss on first R, then followed by all hits). Since block size for Cache Y is 256 B and char size is 1 B (256 array elements per block), we need our LEAP to be 256/8 = 32.

d) For the loop shown in part (c), let LEAP = 64. Circle ONE of the following changes that increases the hit rate of Cache X: [2 pts]

- Increase Block Size (hit rate ↑)
- Increase Cache Size (no change to hit rate)
- Add a L2$ (miss penalty ↓)
- Increase LEAP (hit rate ↓)

e) For the following cache access parameters, calculate the AMAT. All miss and hit rates are local to that cache level. Please simplify and include units. [1.5 pts]

<table>
<thead>
<tr>
<th></th>
<th>L1$ Hit Time</th>
<th>L1$ Miss Rate</th>
<th>L2$ Hit Time</th>
<th>L2$ Hit Rate</th>
<th>MEM Hit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 ns</td>
<td>40%</td>
<td>20 ns</td>
<td>95%</td>
<td>400 ns</td>
</tr>
</tbody>
</table>

AMAT = 2 + 0.4 * (20 + 0.05*400)

18 ns
### Question 5: Floating Point (10 pts)

Assume integers and IEEE 754 single precision floating point are 32 bits wide.

**a)** Convert from IEEE 754 to decimal: \(0xC0900000\) [3 pts]

\[
S = 1, \ E = 0b1000\ 0001, \ M = 0010...0; \ -1.001_2 \times 2^2 = -100.1_2 \]

\(-4.5\)

**b)** What is the smallest positive integer that is a power of 2 that can be represented in IEEE 754 but not as a signed int? You may leave your answer as a power of 2. [2 pts]

Largest 32-bit signed int is \(2^{31} - 1\).

\(2^{31}\)

**c)** What is the smallest positive integer \(x\) such that \(x + 0.25\) can't be represented? You may leave your answer as a power of 2. [3 pts]

Need \(2^{-2}\) digit to run off end of mantissa, so 

\[
10000000000000000000000.012 = 1.000000000000000000000001 \times 2^{22}\]

\(2^{22}\)

**d)** We have the following word of data: \(0xFFC00000\). Circle the number representation below that results in the most negative number. [1 pt]

<table>
<thead>
<tr>
<th>Unsigned Integer (positive number)</th>
<th>Two’s Complement (negative number)</th>
<th>Floating Point (NaN)</th>
</tr>
</thead>
</table>

**e)** If we decide to stray away from IEEE 754 format by making our Exponent field 10 bits wide and our Mantissa field 21 bits wide. This gives us (circle one): [1 pt]

MORE PRECISION // LESS PRECISION

Fewer mantissa bits means less precision.

### Question 6: Performance (4 pts)

We are using a processor with clock period of 1 ns.

**a)** Program A contains 1000 instructions with a CPI of 1.2. What is the CPU time spent executing program A? [2 pts]

CPU Time = 1000 * 1.2 * 1 ns

\(1200\ \text{ns} = 1.2\ \mu s\)

**b)** Program B contains 500 instructions but accesses memory more frequently, what is the maximum CPI that program B can have without executing slower than program A? [2 pts]

Half as many instructions, so can have twice as big CPI.

\(2.4\)