ECE 411 Exam 1

- This exam has 6 problems. Make sure you have a complete exam before you begin.
- Write your name on every page in case pages become separated during grading.
- You will have three hours to complete this exam.
- Write all of your answers on the exam itself. If you need more space to answer a given problem, continue on the back of the page, but clearly indicate that you have done so.
- This exam is closed-book. You may use one sheet of notes.
- You may use a calculator.
- Do not do anything that might be perceived as cheating. The minimum penalty for cheating will be a grade of zero.
- Show all of your work on all problems. Correct answers that do not include work demonstrating how they were generated may not receive full credit, and answers that show no work cannot receive partial credit.
- The exam is meant to test your understanding. Ample time has been provided. So be patient and read the questions/problems carefully before you answer.
- Good luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP0/Debugging</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>MP1/Testing</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Cache w/ Virtual Memory</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>88</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. MPO/Debugging (11 points)
Here is a small piece of test code that Joe Student is using to debug his implementation of an LC3b processor:

```
LDR R1, R0, SEVEN
LDR R2, R0, MINUS
AND R3, R1, R2
ADD R4, R1, R3
HALT:
    BRnzp HALT
;Data
SEVEN:    DATA2 4x0007
MINUSTEN: DATA2 4xFFF6
```

Joe loads the code into memory and tests his design in ModelSim. The first 400 ns of the generated waveform is shown below.

(a) (4 points) Joe thinks that there is a bug in his implementation. Circle all the values in the waveform that are wrong.
(b) (2 points) For each circled value, write the correct value next to it.
(c) (5 points) Provide a brief description (1-2 sentences) of a possible implementation bug in Joe’s SystemVerilog code. Justify your answer.

In `s_and state, aluopcode = aluop-add`.

- OR -

For `aluopcode = aluop-and`, the decoder sends it to state `s-add`.

ECE 411 Exam 1
Page 2
2. MP1/Testing (11 points)
Joe Student has finished MP0, and has written test code to test the LDI instruction.

```
ORIGIN 4x0000

SEGMENT CodeSegment:
    LEA R0,DataSegment
    LDI R1,R0,PTR2
HALT:
    BRnzp HALT

SEGMENT DataSegment:
    PTR1: DATA2 VAL2
    PTR2: DATA2 VAL3
    PTR3: DATA2 VAL1
    VAL1: DATA2 4x0003
    VAL2: DATA2 4x05ff
    VAL3: DATA2 4x0404
```

(a) (6 points) The following waveform shows the LDI instruction executing for the above code. Fill in the values that should be in the waveform at the places labeled A, B, C, and D.

(b) (5 points) What aspect of the LDI is not tested in the given code? Assume that all the registers values shown in the trace are set correctly (including the ones you filled in).

It does not test whether LDI sets the condition codes.

OR

It does not test whether LDI sign extends the offset.
3. ISA (21 points)

(a) (4 points) Name one ISA with fixed-length instructions and one ISA with variable-length instructions (do not use LC3 related ones). List one advantage and one disadvantage for both fixed-length instruction ISAs and variable-length instruction ISAs.

<table>
<thead>
<tr>
<th>ISA</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed: ARM</td>
<td>+ easy fetch/decode</td>
<td>- large binary size</td>
</tr>
<tr>
<td>variable: X86</td>
<td>+ small binary size, more flexible</td>
<td>- complex fetch/decode</td>
</tr>
</tbody>
</table>

(b) (6 points) Write the program that does \( C = A - B \) using the following 4 kinds of machines. You can assume \( SUB \) instruction exists. (\( A, B, C \) are memory locations)

i. Stack Machine

\[
\text{PUSH } A, \ \text{PUSH } B, \ \text{SUB}, \ \text{PUSH } C
\]

ii. Accumulator

\[
\text{LD } A, \ \text{SUB } B, \ \text{ST } C
\]

iii. Register Memory

\[
\text{SUB } C, A, B
\]

iv. Load Store

\[
\text{LD } R1, A; \ \text{LD } R2, B; \ \text{SUB } R3, R1, R2; \ \text{ST } R3, C
\]

(c) (3 points) Discuss the tradeoffs of supporting 9 registers instead of 8 in LC3b ISA? (instruction length is still 16 bits)

+ more register to use, can hold more data in registers at a time

- need more bit to index into register file, bad for the ISA

(d) (4 points) Given your current incomplete MP2 (read is done through cache and write goes directly to memory), what could go wrong if you run a test code which contains both read and write?

Main issue: if a line is in cache, and it is being written to in memory, the line in cache would not be correctly updated.

(e) (4 points) How would you fix the issue in part (d)? (with minimal effort) Your answer should not be more than one sentence.

Invalidate the line in cache if we are writing to memory on that line.
4. Cache (18 points)
Ben Bitdiddle is a student from one of our peer institutions. Although he has taken a computer architecture course, he did not learn very well and is making terrible design constantly. Now you, as an ECE 411 student, are going to help him clean up the mess he made.

(a) (2 points) Ben designed a cache that has block size equal to the word size. Why this is a bad cache design? The answer should be no more than one sentence.

Not able to exploit DRAM burst and spatial locality.
(either of these is ok)

(b) (4 points) Ben is examining a 384-byte set associative cache that has following characteristics.
- Byte addressable.
- 16-bit memory address.
- 8 cache lines per way. 16 bytes of data per cache line.
- Read/write allocate. Write back policy. True LRU replacement policy.

In the spaces below, indicate the specifications of the cache

4 offset bits
3 index bits
9 tag bits
3 set associativity
(c) (6 points) *(Use the cache configuration from part b)* Below is a sequence of hex memory address in the order they are used to reference memory. They are generated by an iteration of a loop. Assume the cache is initially empty. For each reference, write down tag and index bits in binary and identify if the reference is a hit or miss.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Memory Address</th>
<th>Tag</th>
<th>Index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>0x2DB2</td>
<td>001011011</td>
<td>011</td>
<td>miss</td>
</tr>
<tr>
<td>Read</td>
<td>0x06FC</td>
<td>000011011</td>
<td>111</td>
<td>miss</td>
</tr>
<tr>
<td>Write</td>
<td>0x5AB8</td>
<td>010110101</td>
<td>011</td>
<td>miss</td>
</tr>
<tr>
<td>Write</td>
<td>0x773E</td>
<td>011101110</td>
<td>011</td>
<td>miss</td>
</tr>
<tr>
<td>Write</td>
<td>0x4BB6</td>
<td>010010111</td>
<td>011</td>
<td>miss</td>
</tr>
<tr>
<td>Read</td>
<td>0x5ABC</td>
<td>010110101</td>
<td>011</td>
<td>hit</td>
</tr>
<tr>
<td>Read</td>
<td>0x2DB6</td>
<td>001011011</td>
<td>011</td>
<td>miss</td>
</tr>
<tr>
<td>Read</td>
<td>0x4BB2</td>
<td>010010111</td>
<td>011</td>
<td>hit</td>
</tr>
</tbody>
</table>

(d) (6 points) *(Use the cache configuration from part b)* Suppose access to cache takes 10 cycles, and when a miss happens, access to DRAM takes an additional 150 cycles. Calculate the cycles taken to run the loop two rounds. Here is a spare table to work on the question.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Memory Address</th>
<th>Tag</th>
<th>Index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>0x2DB2</td>
<td></td>
<td></td>
<td>hit +</td>
</tr>
<tr>
<td>Read</td>
<td>0x06FC</td>
<td></td>
<td></td>
<td>hit +</td>
</tr>
<tr>
<td>Write</td>
<td>0x5AB8</td>
<td></td>
<td></td>
<td>hit +</td>
</tr>
<tr>
<td>Write</td>
<td>0x773E</td>
<td></td>
<td></td>
<td>miss</td>
</tr>
<tr>
<td>Write</td>
<td>0x4BB6</td>
<td></td>
<td></td>
<td>miss</td>
</tr>
<tr>
<td>Read</td>
<td>0x5ABC</td>
<td></td>
<td></td>
<td>hit +</td>
</tr>
<tr>
<td>Read</td>
<td>0x2DB6</td>
<td></td>
<td></td>
<td>miss</td>
</tr>
<tr>
<td>Read</td>
<td>0x4BB2</td>
<td></td>
<td></td>
<td>hit +</td>
</tr>
</tbody>
</table>

First Round: 2 hits, 6 misses, 2 writebacks

Miss w/ Writeback = access cache + WB DRAM + Read DRAM + Read Cache = 320 cycles

Hit + = 10 cycles

T_1 = 2 \times 320 + 4 \times 150 + 2 \times 10 = 1340

Second Round: 5 hits, 3 misses, 2 writebacks

T_2 = 2 \times 320 + 1 \times 150 + 5 \times 10 = 860

T_{total} = T_1 + T_2 = 2200
5. Cache w/ Virtual Memory (16 points)

(a) (2 points) Ben knows that virtual memory is a crucial part in modern computer systems so he decides to add virtual memory support into his design. Ben remembers there is a cache called TLB but unfortunately he forgot what it is. Do you know what the full name of TLB is and what content is stored in TLB?

(1) Translation Look-aside Buffer
(1) Stores physical page number (physical address)

(b) (2 points) When should we flush the TLB and Why?

(4) Context Switch. Each process has its own page directory. Virtual addresses are mapped to different physical addresses for different processes. The PDBR of different processes are different. Translations cached by the TLBs pertain to the address space of respective process.

(c) (4 points) The virtual memory system specifies the page size as 4KB. Ben would like to provide a 128KB data cache. He would also like to designed it as a Physical-Indexed-Physical-Tagged cache, what is the minimum associativity so that virtual address translation and cache look-up can be done in parallel? (Hint: line size should not matter.)

(1) 4 KB page requires 12 bits for page offset. The cache index and offset 6+ in total has to be within this 12-bit field so that we can have address translation and cache look-up in parallel.

(3) Worst case associativity: 1 way \( = 2^{\text{index}} \times 2^{\text{offset}} \times 1 \times 1 = 2^3 \times 2 \text{ way} \).

Cache Size = 128KB = \( 2^{18} \) B. So, in this case, we need \( 2^5 = 32 \) way set associativity.
(d) (5 points) Given a 2KB direct-mapped cache, with 8 bytes per line. The computer system also supports VM with page size of 2KB and 1 entry TLB. Ben is running two dimensional histogramming of a matrix of characters (1 Byte per character) on a computer with the above configuration. Ben finds the code runs unacceptably slow.

```cpp
unordered_map<char, int> histogram411(vector<vector<char>> myString)
{
    unordered_map<char, int> myHistogram;
    for(int i = 0; i < 1024; i++) {
        for(int j = 0; j < 1024; j++) {
            uint_32 val = myNums[j][i];
            // mark 1 if it is the first encounter
            // increment value if not
            myHistogram[val] =
                !myHistogram.count(val) ? 1 : myHistogram[val] + 1;
        }
    }
    return myHistogram;
}
```

Calculate the hit rate of the cache and the TLB of the code. Please ignore the hit or miss of accessing unordered_map when you do the calculation since a hash table is difficult to trace.

(2.5) Cache hit rate is 0.

(2.5) TLB hit rate is 1/2.
(e) (3 points) Could you modify the code to make it faster? Calculate the hit rate of the cache and the TLB of your modified code.

```cpp
unordered_map<char, int> histogram411(vector<vector<char>> myString)
{
    unordered_map<char, int> myHistogram
    for(int i = 0; i < 1024; i++) {
        for(int j = 0; j < 1024; j++) {
            uint32_val = myHistogram[j];
            ---------------
            myHistogram[j] = 1;
            count(val) += 1;
            myHistogram[val+1];
        }
    }
    return myHistogram;
}
```

(2) Cache hit rate = \( \frac{7}{8} \)

TLB hit rate = \( \frac{3047}{3048} \)
6. DRAM (11 points)
This part tests your understanding of the DRAM organization. Assume the following design parameter values:

- DDR (Double Data Rate) interface – the interface transfers 2 times in each clock cycle, once at the rising edge and once at the falling edge.
- The interface is clocked at 1.0 GHz \((G = 10^9)\)
- Each memory channel connects to one interface
- Each bank consists of one DRAM core, which is 4M bits
- The DRAM core is clocked at 1/8 of the interface clock speed.
- The interface width is 64 bits wide
- The core access latency (delay) is 24 interface clock cycles

(a) (2 points) What is the burst size?

\[
\frac{\text{ratio of clock over DRAM core clock}}{\text{interface width}} = 8 \times 64 = 512 \text{ bits (64B)}
\]

(b) (2 points) What is the maximal memory bandwidth that can be supported by one interface (channel)?

\[
\frac{64 \text{ bits}}{8 \text{ bytes}} \times \frac{1 \text{ GB} \times 2 \text{ transfers per clock}}{} = 16 \text{ GB/s}
\]

(c) (4 points) What is the minimal number of banks required to achieve an actual transfer rate equal to the channel (interface) bandwidth?

\[
\text{core latency} = 24, \quad \text{Each burst takes } \frac{24}{4} = 6 \text{ banks needed for } \frac{64\text{B}}{8\times2} = 4 \text{ cycles to transmit}
\]

(d) (3 points) What is the number of channels needed to equip a CPU with 48 GBytes/sec memory bandwidth?

\[
\frac{48\text{ G B}}{16\text{ GB}} = 3 \text{ channels}
\]