ECE 411, Exam 1

- This exam has 5 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 also use a calculator.
- DO NOT do anything that might be perceived as cheating. The minimum penalty 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. If a dinosaur is drawn as an answer, it will not receive credit. But we will think kindly of those that make us smile.

- The exam is meant to test your understanding. Ample time has been provided. So, be patient. Read the questions/problems carefully before you answer.

- **Good luck!**
Problem 1 (15 pts): __________
Problem 2 (6 pts): __________
Problem 3 (20 pts): __________
Problem 4 (10 pts): __________
Problem 5 (16 pts): __________
Total: __________
1. ISA

Alice Assembler, an ECE 411 student, has become frustrated at the lack of simple arithmetic instructions in the LC-3b ISA, and would like to improve the situation by modifying the ISA and hardware.

a) The instructions that Alice would like to implement are the standard "SUB", "OR", "XOR", "NAND", "NOR", "XNOR", "MULT", and "DIV". In addition they would like to add "LDBSE" or "load byte sign extend", which should work the same way as LDB except that the data is sign extended rather than zero extended.

Alice decides that the easiest way to add most of the new instructions is to utilize bits [4:3] in the non-immediate ADD and AND instructions as a function selector.

How many total instructions can this choice support? How many new instructions can this choice support? (3 point)

2 x 2^2 = 8 total instructions
We still need to support the non-immediate ADD and AND instructions, so that gives 6 new instructions.

b) Define a mapping between the function selector bits and the new instructions that can be implemented with the ALU (hint: make SUB very similar to ADD). (5 points)

There are a number of correct answers for this problem. Any reasonably logical mapping was marked correct. Here is an example:

ADD: AND:
SUB NAND
OR XOR
NOR XNOR

You should have recognized that LDBSE could not have been mapped.

c) Which instructions (if any) do you think Alice cannot add this way? On the LC-3b data path below identify any logic blocks and/or signals Alice will need to add or change in order to implement
only the REMAINING new instructions. Identify what each block does (or does differently now), and which instructions use it. (4 points)

LDBSE and whichever 2 instructions didn’t fit in the last part (in this example MULT/DIV).

There was some confusion as to what “REMAINING new instructions” meant. Full credit was given for correctly implementing either the instructions listed in part b) or for the instructions above that could not be added with ADD/AND.

d) Below is a Control State Machine that includes the implementation of the normal register to register ADD instruction. Describe how to change it to implement the SUB instruction as well. How many of the other new instructions can also follow this pattern (even if they do not use the old ADD opcode number)? (3 points)

The main point of this answer was that you need an extra state to check bits [4:3] and then transition to ADD or SUB. Every instruction listed in part b) can also follow this pattern. LDBSE cannot because it does not map this way. MULT/DIV cannot because they require too many cycles to complete.

2. **Performance**

Ben Bitdiddle is comparing two single-cycle-architecture computers. He notices that computer FOO has a clock frequency of 1.8GHz and computer BAR is clocked at 3.6GHz.

a) When he runs a similar program on the two different machines, he observes that computer FOO runs the program much faster. Why is that? (3 points)

CPI for computer BLAH is much lower (more than twice as small) then CPI for computer BOOO
b) Now he runs a different program on the two machines. Program X has 5 times as many instructions as program Y. Ben runs program X on computer FOO and program Y on computer BAR. It turns out that both computers finish running the program at the same time. Calculate the ratio of CPIs of computer FOO over computer BAR. (3 points)

\[
\text{time} = \frac{\text{instr/\text{prog}} \times \text{cyc/\text{instr}} \times \text{time/cyc}}{5 \times \frac{\text{CPI}_\text{BLAH}}{\text{CPI}_\text{BOOO}} = 1 \times \frac{\text{CPI}_\text{BOOO}}{3.6}} \\
\text{CPI}_\text{BLAH} / \text{CPI}_\text{BOOO} = 1/10
\]
3. Cache
Consider a 2,048B (2KB) cache designed for LC-3b which is organized as 4-way set associative with 64B blocks. A logic diagram of the cache is shown below along with its initial state: empty cells represent empty blocks and cells labeled “Reserved” cannot be used to store any valid data, and cannot be evicted by any replacement policy. The diagram shows only the data banks of the cache; you may assume that whenever the data is in the cache, the tag comparison results in a “hit”.

<table>
<thead>
<tr>
<th>Set #</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>A,B,C</td>
<td></td>
<td>E,F</td>
</tr>
<tr>
<td>6</td>
<td>G</td>
<td>I</td>
<td>H</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td>K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a.) Which bits of the physical address represent offset, index, and tag? (2 points)
Bits 15-9: Tag

Bits 8-6: Index

Bits 5-0: Offset

b.) The execution of a small program on this version of the LC-3b processor generated the following memory trace, where all addresses are taken to be physical addresses:


where each symbol corresponds to the following physical addresses:

\[
\begin{align*}
A &= 000000101000111 \\
B &= 000000101001001 \\
C &= 000000101010000 \\
D &= 000000110101100 \\
E &= 000010101100000 \\
F &= 000010101000011 \\
G &= 001100011001010 \\
H &= 010000011000000 \\
I &= 000100111000110 \\
J &= 0011101110100010 \\
K &= 000010111011100
\end{align*}
\]

Specify the contents of the cache after execution of this trace. For this part, assume perfect LRU replacement. Show the content of the cache on the diagram provided on the previous page. (8 points)
(c) How many misses did you record during execution of the trace?

Number of misses with LRU replacement: __5________________________(2 points)

(d) Suppose now that we want to use a pseudo-LRU replacement policy. The pseudo-LRU state is given as a bit vector [Bit 2, Bit 1, Bit 0], where each bit signifies the following:

Bit 2: Which half was more recently accessed.

Bit 1: Which half of the first half was more recently accessed.

Bit 0: Which half of the second half was more recently accessed.

Specify the pseudo-LRU state after each memory access, assuming that all sets share the same bit vector.

A -> [0,1,-]
B -> [0,1,-]
C -> [0,1,-]
F -> [1,1,1]
E -> [1,1,1]
J -> [0,0,1]
K -> [0,1,1]
D -> [0,0,1]
I -> [0,1,1]
B -> [0,1,1]
H -> [1,1,0]
A -> [0,1,0]
B -> [0,1,0]
G -> [1,1,0]
D -> [0,0,0]

Which block would be replaced if J were read again? _Way 3, Line 6__________(8 points)
4. Cache/VM Interaction

A machine has the following configuration:
- 32 bit virtual address space, byte addressable
- 4kB page size
- 32 bit physical address space, byte addressable
- Cache size is 128kB

a) To implement a standard (the one you learned in class) cache-VM interaction with VIPT, what is the minimum associativity for the cache so that virtual address translation and cache look-up can be done in parallel? (3 points)

   4kB page requires 12 bits for page offset. To implement a perfect VIPT, the cache index and offset bit in total has to be within these 12-bit field so that we can have address translation and cache look-up in parallel. (1pt)

   worst case associativity: 1 way = 2^(index bits) * 2^(offset bits) * 1B = 2^12 byte per way. cache size = 128kB = 2^17 bytes. So worst case, we need 2^5 = 32 way associativity. (2pt)

b) Suppose the cache in the machine is an 8-way associative cache. Do you think there will be a problem if we use a simple VIPT approach? If you think there is a problem, give one memory access pattern that will result in erroneous behavior. If you think it’s fine, explain why. (3 points)

   big problem!!
   8-way = 2^3. That means offset+index bits are 17-3 = 14 bits. This is bigger than 12 bits (page size). Sooo, not all index+offset bits fall inside the “non-translated” virtual address. That means, there are gonna be some bits (most likely from the index bits) that need to be translated. Simple VIPT approach can’t be done. (1.5pt)

   2-bit overlap. Accept all reasonable solution to this. But here is one example (bear in mind that some index bits need to be translated). VPN and PPN number will have 20 bits. Suppose the high 18 bits of 2 different VPN translate to a same high 18 bits of the PPNs. And suppose the remaining 2 bits (this is shared by the index bits) of the VPN are the same, but are translated to different remaining 2 bits of the PPN. This will result in 2 different data being stored in a single location in the cache (with the same tag and index) where what we want is to store those 2 data in different index with the same tag. (1.5pt)

c) Now consider if the cache block size is 16 bytes and there are 11 index bits. There is a problem to this! Without changing the cache size, how would you solve the problem so that the ideal hit time is close to the ideal VIPT hit time? Close means, up to 2-level logic gates
6. Miscellaneous

a) Ben Bitdiddle wants to deploy his MP2.1 pipeline in a real world setting. To do this, Ben decides to add a cache to his MP2.1 pipeline. Ben decides to use the LC3b TRAP instruction to implement syscalls and the LC3b instruction JSR to implement subroutines. User level code is found in addresses x2000 - xFFFF, for ease of TRAP syscall implementation.

Ben initially decides to use a single cache to cache memory accesses. He quickly finds performance issues and decides to split his cache into an Instruction and Data cache and use Writeback as the cache’s write policy.

i) Why would separate Instruction and Data caches be a sensible design decision? (2 points)

Instructions and Data tend to occur in different portions of memory, so a single specialized instruction cache can be used for instruction lookup and another for data lookup.

ii) Under what situation would separate Instruction and Data caches with a Writeback policy produce a drop in performance? (2 points)

Self-modifying code would have instructions be loaded into the data cache, and have reads from subsequent instructions wait for the writeback cache to flush its contents back into memory. Both caches would be thrashed and memory traffic would be wasted flushing and fetching modified memory.
iii) Ben’s friend has written an operating system for Ben’s processor after taking ECE 391. Ben decides to deploy his MP2.1 processor with his friend’s operating system. After many calls to printf(), Ben finds that his processor suffers from cache-related slowdown, despite having a dedicated instruction cache. Name a possible source of this slowdown. Outline how Ben could fix this slowdown. (Note: Ben’s friend has written a perfect operating system and the operating system cannot be altered.) (2 points)

Trap vectors dispatch into the area of system level code before x2000. Constant syscalls fill the instruction cache with instructions from the x0000 - x2000 range and then user level code again fills the cache with instructions from the x2000 - onward range. Constant syscalls continue to cause cache thrashing and greatly slow down performance.

A solution would be to implement a vector cache, specifically for caching system calls. Because trap vectors are contained in their memory usage, the vector cache could easily cache TRAP operations.

b) Ben Bitdiddle is taking a break at Murphy’s when he hears a disgruntled graduate student talk about “Loop Unrolling”. “Loop Unrolling” occurs when a compiler takes a loop with fixed bound and unrolls it into a static set of instructions. For example:

```c
int my_vector[5];
for(int i = 0; i < 5; i++) {
    my_vector[i] = my_vector[i] * my_vector[i];
}
```

ECE 411, Exam 1
Ben has access to a version of the MP2.1 pipeline that has a single unified cache for Instruction and Data. Ben decides to use the graduate student’s work to produce unrolled LC3b code for the above vector scale operation. Ben notices that the unrolled loop performs worse than the original code for very large vectors. Why could this be happening? (Hint: Rewrite the above LC3b code for an unrolled version of the loop.) (3 points)

Very large vector scale operations would unroll into a very large set of instructions. Cache misses on these instructions would constantly cause the cache to evict cached contents and cache another set of instructions. With very large datasets, eventually the instructions and data will both contend for cache occupancy, thrashing the cache. Keeping the set of instructions small keeps their presence in the cache limited, so the cache will only fill up due to data lines.

c) A 16-bit instruction takes the following format <OPCODE><DR><SR><IMM> where DR specifies the destination register, SR specifies the source register, and IMM is a 2’s complement immediate value. If there are 27 opcodes and 16 registers, the number of bits left to specify the immediate value is _______ 3 bits ________ (3 points)

CEILING( log2(27) ) = 5
CEILING ( log2(16)) = 4
16 – 5 – 4 – 4 = 3 bits

d) A system with a 16 bit address space and byte-addressable memory has a direct-mapped cache that consists of 16 lines of 16 bytes each. The cache has a write-back, write-allocate policy.
Compute the total number of bits of storage (including valid, dirty, and tag bits) it takes to implement the cache. (3 points)

\[16 \times 16 \times 8 \text{ (DATA)} + 16 \times 1 \text{ (VALID)} + 16 \times 1 \text{ (DIRTY)} + 16 \times 8 \text{ (TAG)} = 2208 \text{ bits}\]

e) A Television Studio is thinking about making a reality TV show about ECE 411. They’re looking for help with ideas. Please come up with an idea for a title and/or catch phrase for this new show. (Example: Welcome to ECE 411, where everything’s made up and the points don’t matter!) (1 point)
Name: ___________________________