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 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.

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 (8 pts): __________
Problem 2 (18 pts): __________
Problem 3 (12 pts): __________
Problem 4 (10 pts): __________
Problem 5 (12 pts): __________
Problem 6 (13 pts): __________
Total (73 pts): __________
1. Machine Problem (8 pts)

a) What changes to the MP1 datapath are needed to support the following instruction? Be specific. (4 points)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Instruction Format</th>
<th>Instruction Operation</th>
</tr>
</thead>
</table>
| LDX    | [1000][000][BaseR][offset6] | R0 <= MEM[BaseR + SEXT(offset6 << 1)]  
R1 <= MEM[BaseR + SEXT(offset6 << 1) + 2]  
R2 <= MEM[BaseR + SEXT(offset6 << 1) + 4]  
R3 <= MEM[BaseR + SEXT(offset6 << 1) + 6] |

Solution:
b) Draw the state machine for LDX, starting at the DECODE state. For each state, show all necessary control signals (state actions) and transition conditions. (4 points)

Solution:
2. Instruction Set Architecture (18 pts)

Two students, Alice and Bob, become frustrated at the lack of simple arithmetic instructions in the LC-3b ISA, and set out to improve the situation. Alice takes the approach of modifying the ISA and hardware, while Bob decides to modify the assembler so that it produces sequences of instructions to implement new mnemonics, just like the MIPS assembler.

a) Identify a positive and negative characteristic of each approach. (2 points)

Solution:
ISA/HW (Alice)
- Positive – potentially a higher performance solution
- Negative – need to deal with bit encoding challenges of new instructions, need to modify HW data path and control unit design, not applicable to existing hardware

Assembler (Bob)
- Positive – no change to HW, can be applied to existing hardware
- Negative – potentially a lower performance solution, may need to reserve registers to hold intermediate results between instructions in a sequence that implements a new mnemonic.

The new instructions they want to implement are SUB, OR, XOR, NAND, NOR, XNOR, MULT, and DIV. 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. Answer the questions b), c), d), and e).

b) How many new instructions can this method support? How about the total number of instructions overall with this method? (2 points)

Solution:
The two bits can distinguish four instructions. With the two original opcodes, there are eight combinations. However, one needs to reserve two of the patterns for the existing ADD and AND instructions. Alice can add 6 new instructions with this encoding.

c) Which instructions (if any) do you think Alice cannot add this way? Which (if any) would you advise her to leave out (choose wisely), and why? Define a mapping between the function selector bits and the remaining instructions. (hint: make SUB very similar to ADD) (3 points)

Solution:
Alice should consider leaving out MULT and DIV. These instructions require either iterations and/or significantly more hardware to implement.

Mapping: Make SUB, XOR, XNOR similar to ADD
         Make OR, NAND, NOR similar to AND
One can also argue that MULT and DIV belongs nicely in the arithmetic category as ADD so they should be added along with SUB. Two of the XNOR, NAND, NOR could be left out since they can be easily synthesized with XNOR, AND, and OR followed by a NOT instruction.

d) On the LC-3b datapath below identify and logic blocks and/or signals Alice will need to add or change in order to implement the only remaining new instructions. Identify what each block does (or does differently now), and which instructions use it. (4 points)

Solution:
The non-immediate field IR[4:3] needs to go the control unit.
The ALUop now needs to specify SUB, OR, XOR, NAND, NOR, XNOR
e) 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)

Solution:
A straightforward approach is to change the ADD state to a more generic OP state and use the IR[4:3] bits to generate the appropriate ADD, SUB, XOR, XNOR operations. Similar modification can also be made to the AND state to accommodate NAND, OR, NOR.

Another way to do this is to extend the Decode state to consider IR[4:3] and add new states for SUB, XOR, XNOR, NAND, OR, NOR similar to those for ADD and AND.
Bob quickly discovers a problem with his approach: For some instruction sequences, he needs to store temporary values in registers, and if those registers are used by the programmer they will be overwritten. He decides to further modify the assembler to use R0 as a temporary register that cannot be used by the programmer. Answer the questions f) and g).

f) Which new instruction(s) do not need any temporary registers? (2 points)

Solution:

In general, the source registers should not be overwritten. Also, if the destination register is also a source register, it cannot be overwritten unless the original content has already been consumed by an earlier instruction in the sequence.

NAND does not need a temporary. It can be implemented with an AND instruction that writes into the destination register followed by a NOT instruction on the destination register.

SUB will need one temporary since we need to do a two’s complement on the second source operand. This is NOT followed by plus 1. However, the destination may be the same as the first source so it will need to be preserved in general.

OR will need one temporary. The temporary and the destination registers can hold the NOT versions of the two sources. An AND to the destination followed by a NOT on the destination should generate the right OR result. (De Morgan).

XOR and XNOR need to generate both min terms in the truth table so will need to have at least one temporary.

One could also argue SUB could avoid using a temporary register, but will not receive full credit if the explanation is not correct. (one typical wrong explanation is put Rb’s 2’s complement into itself, which will corrupt Rb, another typical wrong explanation is to put Rb’s 2’s complement into the destination register, but fails to explain the corner case when Rb is also the destination register)

Because we did not specify if we allow behavioral design, one may argue that all of these one cycle operations do not need temporary storage, as they could use a powerful ALU to do it all. I also give credits to this case.

g) Assuming programs can easily be rewritten to not use R0, what kind of effect will this have on performance? What instructions will become more frequently used by programs written for this new assembler? (2 points)

Solution:

The reduced number of registers will likely cause more register spilling and cause load/store instructions to be more frequently used.
3. Performance (12 pts)

Use the state diagram for MP2.1 including all instructions listed on the back of the exam. Consider the following code:

1. LDR R1, R0, N
2. LEA R2, R0, A ; load address of A
3. LEA R3, R0, B ; load address of B
4. LEA R4, R0, C ; load address of C

TOP: ; This is the LC3 equivalent C code:
    ; *c++ += *a++ + *b++

5. LDR R5, R2, 0 ; load value of A[i]
6. LDR R6, R3, 0 ; load value of B[i]
7. ADD R5, R5, R6 ; A[i] + B[i]
8. LDR R6, R4, 0 ; load value of C[i]
9. ADD R6, R5, R6 ; C[i] += A[i] + B[i]
10. STR R6, R4, 0
11. ADD R2, R2, 2 ; Increase A ptr
12. ADD R3, R3, 2 ; Increase B ptr
13. ADD R4, R4, 2 ; Increase C ptr
14. ADD R1, R1, -1
15. BRp TOP
16. DONE: BRnzp DONE

a) How many cycles (in terms of N) does the loop take to execute on an MP2.1 CPU? Assume memory responds instantly. Do not include the setup code (line 1-4) and final branch instruction (line 16). Count up to, but not including, the first IF1 state of ‘BRnzp DONE’. (2 points)

Solution: LDR/STR – 7 cycles
    ADD – 5 cycles
    BR – 6 cycles (taken), 5 cycles (not taken)

Loop = 7 * 4 + 5 * 6 + 6 = 64 cycles
Last loop = 7 * 4 + 5 * 6 + 6 = 63 cycles
Total = 64(N-1) + 63 = 64N – 1 cycles
b) You wish to increase the speed at which this loop can run on your CPU. Design a new instruction for the LC3-b to further meet this goal. *You may not modify the memory interface, add additional registers, or add additional ALU operations.* (7 points)

i. Give the assembly syntax of the instruction. (1 point)

ii. Show what each of the 16 bits of the instructions signify. (1 point)

iii. Give the RTL for the instruction. (1 point)

iv. Describe the functionality of the instruction. (2 points)
v. Draw the additional states in the state diagram from Decode stage with details on what happens in each state. (2 points)

c) Rewrite the program with new instruction and find the speedup of the new version vs. the original one. Assume N=30. (3 points)
4. Cache Architecture (10 pts)

Consider a direct-mapped 2KB byte data cache with 8 byte line and 32-bit address space, byte-addressable. Suppose A is a 1024 element integer (4 bytes) array and is aligned on a cache block boundary (i.e., A[0] is at the start of a cache block). Consider the following code that increments the first half of the array A and decrements the second half of A (again, be sure to explain all your answers for the parts below):

```plaintext
For (i = 0; i < 512; i += 1) {
}
```

a) What is the data cache miss rate for the above code with the given cache? (2 points)

Solution:
The miss rate is 50%. In each iteration of the loop, the access to A[i] will bring a block that is useful in the next iteration due to spatial locality. Unfortunately, A[i+512] maps to the same cache frame and so the subsequent access to A[i+512] in the same iteration will result in evicting A[i]. So each read the array A[i] will miss, and each write to A[i] will hit, giving a read miss rate of 50%.

b) Would adding a victim cache improve the above miss rate? Why? (2 points)

Solution:
Yes. The block that would have previously been evicted now would be moved to the victim cache. So on the subsequent iteration, each access would find the data it needs in the victim cache, except for the very first time that block is loaded into the cache. Thus, by adding a victim cache, we will only observe compulsory misses.

c) Then your boss gave you a 2KB memory and asked you to implement the cache. After a brief thinking, you found out that your boss has forgotten the overhead for a cache. Considering all the overhead (dirty bit, valid bit, and tag), and using only 2KB of memory, what’s the maximum useful data size of a cache? (3 points)

Solution:
Original number of lines = $2^{11}/2^3 = 2^8 = 256$
Tag bits = 32 – 8 (index) – 3 (offset) = 21 bits
Overhead for each line = 21 + 1 + 1 = 23 bits
New cache line size = 8 bytes + 23 bits = 87 bits
New number of lines = 2KB / 87 bits ≈ 188 lines
Data size = 188*8 bytes = 1504 bytes
d) Once you get the result, you are very excited and told your boss the maximum capacity you can have. However, your boss is not satisfied with this answer. He told you that maybe if you design a 4 way set associative cache, you can have a larger capacity. Do the same calculation for the 4 way set associative cache, and tell your boss if his assumption is true. (3 points)

Solution:
Original number of lines = \( \frac{2^{11}}{(2^3 \times 2^2)} = 2^6 = 64 \)
Tag bits = 32 – 6 (index) – 3 (offset) = 23 bits
Overhead for each line = 23 + 1 + 1 = 25 bits
New cache line size = 8 bytes + 25 bits = 89 bits
New number of lines = 2KB / 89 bits ≈ 184 lines
Data size = 184*8 bytes = 1472 bytes
5. Cache/VM Interaction (12 pts)

a) Consider a virtual memory system with the following parameters:

- 32 KB direct mapped cache with 32 byte lines
- 48 bit virtual addresses
- 32 bit physical addresses
- 4 KB page size
- A fully associative TLB with 256 entries
- Memory is byte addressable

For the system described, what is the disadvantage of using a physically-indexed cache? Give two different changes to the cache design to enable physical indexing without this disadvantage and without changing the cache size. Please explain your answers and be specific to get partial credit. (4 points)

Solution:
A 32KB direct mapped cache requires 15 bits for the index(10) and block offset(5). Since 3 of these bits overlap with the page number bits, using a physically-indexed cache needs to translate virtual addresses before accessing the cache. This can delay every memory access. Two changes to the cache design to solve the above problem are:

1) We can increase the cache associativity to 8-way. Now we need three fewer bits to index the cache, thereby removing the overlap between the index and page number bits.

2) We could search 8 cache sets in parallel, corresponding to the 3 overlapping bits. This search can happen in parallel with the address translation. Once the translated address is available, we can use a mux to determine which cache set(from the 8 cache sets) is the one we need.
b) Draw a flow diagram, showing all cases of TLB miss, TLB hit, page hit, page fault, and where the TLB and page table are updated, all cases of cache hit/miss, read/write, both states of the dirty bit, and when the dirty bit is set/cleared for a virtually-indexed, physically tagged write-back cache with write-allocation. (8 points)

Solution:
6. Potpurri (13 pts)

a) You want to analyze the performance of a Multi-Cycle processor with 5 stages (Fetch, Decode, Execute, Memory, Writeback). Every instruction must pass through each stage. Decode, Execute, Writeback stages take 1 cycle each. For Fetch and Memory stages, the access time varies on memory architecture. Assume 40% of instructions are loads and stores. (3 points)

i. Memory access takes 50 cycles without any cache implemented. What is the CPI of this machine?

Solution: \(0.6 * (50 + 1 + 1 + 1 + 1) + 0.4 * (50 + 1 + 1 + 50 + 1) = 32.4 + 41.2 = 73.6\) CPI

ii. Now, you want to implement split L1 cache. Both L1 instruction and data cache take 1 cycle cache hit and 50 cycles cache miss. L1 instruction cache has hit rate of 95%, when L1 data cache has 80% hit rate. What is the new CPI of this machine? What is the speedup over part A?

Solution: \(0.6 * ((0.95 * 1 + 0.05 * 50) + 1 + 1 + 1 + 1) + 0.4 * ((0.95 * 1 + 0.05 * 50) + 1 + 1 + (0.8 * 1 + 0.2 * 50) + 1) = 4.47 + 6.9 = 11.37\) CPI

\[
\text{Speedup} = \frac{73.6}{11.37} = 6.47x
\]

iii. You are becoming ambitious and planning to add additional unified L2 cache. This L2 cache has 5 cycles cache hit and 50 cycles cache miss with 98% hit rate. What is the speedup over part i? What is the speedup over part ii?

Solution: \(0.6 * ((0.95 * 1 + 0.05 * (0.98 * 5 + 0.02 * 50)) + 1 + 1 + 1 + 1) + 0.4 * ((0.95 * 1 + 0.05 * (0.98 * 5 + 0.02 * 50)) + 1 + 1 + (0.8 * 1 + 0.2 * (0.98 * 5 + 0.02 * 50)) + 1) = 3.147 + 2.49 = 5.637\) CPI

\[
\text{Speedup over part i} = \frac{73.6}{5.637} = 13.057x
\]

\[
\text{Speedup over part ii} = \frac{11.37}{5.637} = 2.017x
\]

b) A program composed of 80% floating point instructions is run on a processor which takes 5ns on average for a floating point instruction. The processor is improved to complete floating point operations in 4ns. The program is optimized by removing unnecessary floating point instructions so that its new composition is 70% floating point instructions. No new instructions are added. All other instructions (e.g. integer operations) takes 2ns on average and remain unchanged between the optimization. What is the fractional speedup? (3 points)

Solution:

Original: \(0.8 * 5ns + 0.2 * 2ns = 4.4ns\)

Improved: \(0.7 * 4ns + 0.3 * 2ns = 3.4ns\)

\[
\text{Speedup} = \frac{4.4}{3.4} = 1.294x
\]
c) Consider two different machines. The first has a single cycle datapath (i.e., a single stage, non-pipelined machine) with a cycle time of 4ns. The second is a pipelined machine with four pipeline stages and a cycle time of 1ns. (3 points)

i. What is the speedup of the pipelined machine versus the single cycle machine assuming there are no stalls?

Solution: Assuming it’s fully pipelined, instructions will be completed every cycle.
Speedup: 4ns/1ns = 4x

ii. What is the speedup of the pipelined machine versus the single cycle machine if the pipeline stalls 1 cycle for 30% of the instructions?

Solution:
Pipelined processor: 1ns + 0.3*1ns (=1 cycle) = 1.3ns
Speedup: 4ns/1.3ns = 3.077x

iii. Now consider a 3 stage pipeline machine with a cycle time of 1.1ns. Again assuming no stalls, is this implementation faster or slower than the original 4 stage pipeline? Explain your answer.

Solution: Also, assuming it’s fully-pipelined, instructions will be completed every clock cycle.
3-stage pipeline: 1.1ns/instruction
4-stage pipeline: 1ns/instruction
3-stage pipeline is slower than the 4-stage pipeline if fully-pipelined.

d) Why does Java Virtual Machine (JVM) have a stack-based architecture? (2 points)
e) A `subleq` instruction has the following semantics:

\[
\text{subleq } a, b, c \quad ; \text{Mem}[b] = \text{Mem}[b] - \text{Mem}[a]
\]
\[
; \text{if (} \text{Mem}[b] \leq 0 \text{) goto } c
\]

A variant is also possible with two operands and an internal accumulator.

\[
\text{subleq2 } a, b \quad ; \text{Mem}[a] = \text{Mem}[a] - \text{ACCUM}
\]
\[
; \text{ACCUM = Mem}[a]
\]
\[
; \text{if (} \text{Mem}[a] \leq 0 \text{) goto } b
\]

A copy instruction, `Mov a, b`, replaces content at location `b` with the content at location `a`. Implement `Mov a, b` using `subleq` instructions. *Hint:* You can introduce as many operands as you want other than `a` and `b`. (2 points)

**Solution:**

\[
\text{MOV a, b == subleq b, b}
\]
\[
\text{subleq a, Z}
\]
\[
\text{subleq Z, b}
\]
\[
\text{subleq Z, Z}
\]