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 3 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 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>Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache + VM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. **ISA**

In this problem, you will explore the effect of different instruction lengths on program performance.

**Make the following assumptions**

- You have a non-pipelined processor with separate instruction and data caches.
- Fetching an instruction takes $M$ cycles:
  - $M = 1$ on *I-Cache Hit*
  - $M = 10$ on *I-Cache Miss*
- After fetch, each instruction takes $N$ cycles to complete:
  - $N = 1$ cycle for *fld, add, sub, cmp, bz*
  - $N = 2$ cycle *fadd*
  - $N = 3$ cycle *fmul*
  - $N = 5$ cycle *fsqrt*
- Execution of an instruction takes $M+N$ cycles.
- The floating point instructions used in the program are 3-byte: *fld, fadd, fmul, fsqrt.*
- The integer instructions used in the program are 1-byte: *add, sub, bz.*
- 32 byte *I-Cache Line.*
- The I-Cache starts empty, and the D-Cache never misses.
- The program always starts at PC = 0.

You will now consider ISA tradeoffs in the following program:

```assembly
LOOP:
    fld f2, (r2)
    fld f3, (r3)
    fmul f1, f2, f3
    fadd f0, f0, f1
    add r2, r2, #4
    add r3, r3, #4
    sub r0, r0, #1
    bp LOOP
    fsqrt f0, f0
```

**Initial values**

- $f0 = 0.0$
- $r0 = 2$
(a) How many cycles does the program take to complete? (3 points)

(b) Now, all instructions are encoded in 4 bytes. How many cycles does the program take? (3 points)

(c) Now assume a new instruction “fmacc” is used in the program to replace

```
fmul f1, f2, f3
fadd f0, f0, f1
```

with

```
fmacc f0, f2, f3
```

All instructions are still encoded in 4 bytes. What is the maximum number of cycles the new instruction may use (find N) in order for the program to complete in the same number of cycles as part (a)? (5 points)

(d) Now, assume the processor in part (a) operates at 2.7 GHz. Also assume that fmacc takes 8 cycles (N = 8) in part (c). At what frequency must the processor in part (c) operate to complete the program in the same time as part (a)? Assume that caches can run at the same frequency as the processor. (5 points)
2. MP

(a) The following is the mp0 design with a multiply (mul) instruction implemented. Add a new instruction called fused multiply and add (fma) to this datapath. You may add signals and any modules used in mp0 and mp1 for this question. You may also use adders. See the instruction specs and state machine for more details on mul and fma. (5 points)

**Multiply (MUL)**

| 1111 | DR | SR1 | 0  | 0  | 0  | SR2 |

DEST <- SR1 * SR2
setcc();

Example:

mul R0, R1, R2
R0 <- R1 * R2

**Fused Multiply and Add (FMA)**

| 1000 | SR2 | SR1 | IMM6 |

R0 <- SR1 * SR2 + SEXT(IMM6)
setcc();

Example:

fma R1, R2, 5
R0 <- R1 * R2 + SEXT(5)

*NOTE: This is the same opcode as TRAP, but this is not implemented in MP0

**NOTE: This is the same opcode as RTI, but this is not implemented in MP0
(b) Write the signals values for every signal in the processor during the state fma. Default any signal values to 0. Add any signals you may have added to the design. (3 points)

load_ir =
storemux_sel =
load_regfile =
marmux_sel =
mdrmux_sel =
load_mar =
load_mdr =
aluop =
alumux_sel =
load_cc =
multout_sel =
regfilemux_sel =
pcmux_sel =
load_pc =
(c) Example code with multiplies and adds

**Initial register values (decimal):**
- R0 = 0
- R1 = 2
- R2 = 5

**LOOP:**
- `mul R0, R1, R0`
- `add R0, R0, 2`
- `add R2, R2, -1`
- `brp LOOP`

(i) Rewrite the code above using the new *fma* instruction. (2 points)

(ii) How many cycles does it take to execute the original code on the unmodified design? Assume the state *fetch2* takes one cycle. (2 points)
(iii) How many cycles does it take to execute the new code on the new design? Assume the state
 FETCH2 takes one cycle. (2 points)

(iv) Suppose the processor without the FMA instruction has a 10% faster clock than the processor
 with FMA. Does the code with FMA still run faster on the modified processor? (2 points)

(v) Why may adding a FMA instruction incur a 10% frequency loss? (2 points)
3. Cache

Your company is building a processor to run a particular program as fast as possible. You have been tasked with optimizing the cache. The processor does not support virtual memory. Currently your team is using a 2-way set associative cache of size 512 bytes (not counting tag storage) with block size of 16 bytes, using a least recently used replacement policy. Memory addresses are 12 bits long and main memory is byte addressable.

(a) Which bits of the address are used for the tag, index, and block offset in a standard implementation of the aforementioned cache? Use $T$ to indicate tag bits, $I$ for index bits, and $B$ for block offset bits. For example, use $TTIBB$ to indicate $tag[5:4], index[3:2], block\ offset[1:0]$. (2 points)

(b) You have profiled the program and found a loop that accounts for a large proportion of total run time. You gathered the following memory trace that represents a typical iteration of the loop:

0x7A4, 0x5A6, 0xCA0, 0x7AC, 0x2A2, 0xCA4, 0x5AE, 0x7A6, 0x2A5, 0x5A8, 0x7A1, 0xCAA

How many misses would your current cache have when given this memory trace assuming a cold (initially empty) cache? (3 points)
(c) Although you were taught the standard cache address bit mapping in class, there’s no rule saying that the tag and index bits have to be mapped in this way. You can try swapping these bits around to see if your cache can perform better.

Let’s say that you swapped the two least significant tag bits with the two most significant index bits (e.g., ..T_2T_1T_0I_{n-1}I_{n-2}.. → ..T_2I_{n-1}I_0T_1T_0I_{n-2}..). How many misses would your cache have for the same memory trace as above (again assuming a cold cache)? (4 points)

(d) Draw the logic that supports both of these mappings, using a single select signal $s$ to choose between them. Use only basic logic gates and single bit width muxes. (4 points)
4. Cache + Virtual Memory

Consider a byte-addressable computer that has a physical address space of 512 bytes. The computer uses a one-level virtual memory system. The page size (both virtual page and physical page) is 8 bytes and virtual address space is 2KB. There is a 128-byte write through, direct-mapped cache, with cache line size of 2 bytes. The cache is virtually indexed and physically tagged (VIPT). Answer the following questions.

(a) Number of virtual page number bits: ____________ (1 point)

(b) Number of physical page number bits: ____________ (1 point)

(c) What bits of a virtual address are used to index into the cache? (2 points)

bit [_________ : _________]

(d) What is the overall size of tag store overhead (including tag and other bookkeeping bits)? (3 points)
Suppose we have two processes running. These processes share some portion of the physical memory. Virtual-physical page mappings are given below:

<table>
<thead>
<tr>
<th>Process 0</th>
<th>Process 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual page</td>
<td>Physical page</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>

(e) Give a complete physical address (in binary) whose data can exist in two different locations in the cache. (2 points)

(f) Give the indexes (in binary) of those two different locations in the cache. (3 points)

(g) We do not want the same physical address stored in two different locations in the 128-byte cache. We can prevent this by increasing the associativity of our VIPT cache. What is the minimum associativity required? (4 points)
5. Pipelining

Consider two processors A and B implementing the LC3b instruction set. A is pipelined and B is not pipelined. B runs at 500 MHz. Assume A is implemented as a standard 5-stage pipeline with a stage latch at the end of each stage. Stage latches add 0.1 ns delay, each.

(a) What is the maximum frequency at which A could run? (2 points)

(b) For processor B, all instructions take 1 cycle. Now assume that for processor A, 30% of instructions cause a 1 cycle stall and 10% of instructions cause a 2 cycle stall. What is the speedup of A over B for a large program (disregard filling the pipeline)? (3 points)
(c) Consider adding a fused multiply add \((fma, \text{same as problem 1 and 2})\) instruction to the LC3b instruction set. FMA R1, R2, 0x7 would compute \(R0 \leftarrow R1 \times R2 + 0x7\). The total combinational delay of the \(fma\) execution stage is 0.8 ns, split 0.6 ns for multiply and 0.2 ns for add. Name **two** ways that you may add this instruction to pipeline A. Calculate the maximum frequency of a pipeline implementing each idea, and briefly discuss advantages and disadvantages. (4 points)

(d) After implementing your new \(fma\) instruction, you decide to test out your pipeline performance with a standard benchmark program. To your surprise, you see that your IPC has decreased! Should you revert the change? Briefly explain why or why not. (3 points)
6. **APPENDIX**

![Image of LC-3b Instruction Formats]

Figure 1.2: LC-3b Instruction Formats. NOTE: + indicates instructions that modify condition codes.