(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 = 0
storemux_sel = 0
load_regfile = 1
marmux_sel = 0
mdrmux_sel = 0
load_mar = 0
load_mdr = 0
aluop = add
alumux_sel = 0
load_cc = 1
multout_sel = 0
regfilemux_sel = 0
pcmux_sel = 0
load_pc = 0
fma = 1
(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 = 0
storemux_sel = 0
load_regfile = 1
marmux_sel = 0
mdrmux_sel = 0
load_mar = 0
load_mdr = 0
aluop = add
alumux_sel = 0
load_cc = 1
multout_sel = 1
regfilemux_sel = 0
pcmux_sel = 0
load_pc = 0
fma = 1

ECE 411 Exam 1
6
(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)

LOOP: fma R0,R1,2
- add R2, R2, -1
- brp LOOP

(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)

3 \cdot 5 \cdot 5 + 6.4 + 5 = 104 \text{ cycles}

3 5\text{cycle instructions} 4 \text{branch (one branch taken; 6 cycle not taken)}
(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)

\[
2 \cdot 5 \cdot 5 + 6 \cdot 4 + 5 = 79 \text{ cycles}
\]

2 6 cycle instructions
in 5 loops

4 6 cycle branch
takes

1 branch
not taken.

(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)

\[
time = \frac{\text{cycles}}{\text{frequency}}
\]

old: \[ time = \frac{104}{1.1f} = \frac{94.554}{f} \text{, new still faster} \]

new: \[ time = \frac{79}{f} = \frac{79}{f} \]

(v) Why may adding a `fma` instruction incur a 10% frequency loss? (2 points)

The critical path is likely going through the multiplier and adder in one cycle requiring a slow down of the clock to allow the final value to reach the
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 $TTIIBB$ to indicate $tag[5:4], index[3:2], block\ offset[1:0]$. (2 points)

\[
TTTIIIIIBBBB
\]

\[
16 = 2^4 \Rightarrow 4 \text{ block offset bits}
\]

\[
\frac{512 \text{ bytes}}{2 \text{ ways \times 16 \text{ bytes}}} = 16 \text{ lines} \Rightarrow 4 \text{ index bits}
\]

(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
```

All indices are the same, but block offset is ignored.

How many misses would your current cache have when given this memory trace assuming a cold (initially empty) cache? (3 points)

12
(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}T_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)

\[
\begin{align*}
\text{T T I I I I I I B B B B} \\
7A &= 01111010 \rightarrow 0110 \quad 1110 \\
5A &= 01011010 \rightarrow 0110 \quad 0110 \\
C A &= 11001010 \rightarrow 1110 \quad 0010 \\
2A &= 00101010 \rightarrow 0010 \quad 1010
\end{align*}
\]

The 12 addresses exist in only 4 blocks, which each map to a different index using this design.

⇒ no evictions
⇒ obligatory misses only

(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: \(8\) (1 point)

(b) Number of physical page number bits: \(6\) (1 point)

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

\[
\text{bit } \begin{array}{c} 6 \\ 1 \end{array}
\]

(d) What is the overall size of tag store overhead (including tag and other bookkeeping bits)? (3 points)

\[
\begin{align*}
tag & : \text{2 bit} \\
\text{valid} & : \text{1 bit} \\
\text{write through} & : \text{no dirty bit} \\
3 \times 64 & = 192 \text{ bits}
\end{align*}
\]
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>Virtual page</th>
<th>Physical page</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>Virtual page</th>
<th>Physical page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>2</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)

\[ \text{example PA: 00001100} \]

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

\[ \text{Virtual page 15: 111000} \]
\[ \text{7: 011000} \]

(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)

\[ 16 \text{ ways} \]
ISA Tradeoffs

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 take 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

So, execution of an instruction takes M+N cycles.

32 byte I-Cache Line
The floating point instructions used in the program are 3 byte: fld, fadd, fmul, sqrt
The integer instructions used in the program are 1 byte: add, sub, bz

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:
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?
5 float instructions, 4 int1 = 15 bytes + 4 bytes = 19 bytes 1 cache line
1 lmiss + 2 * (2+2+4+3+2+2+2+2) + 6 = 9 + 2 * 19 + 6 = 53 cycles
b. Now, all instructions are encoded in 4 bytes. How many cycles does the program take?

\[ 9 \times 4 = 36 \text{ bytes} \quad 2 \text{ cache lines} \]

Cycles = 62

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

\[
\begin{align*}
&f \text{mul } f_1, f_2, f_3 \\
&f \text{add } f_0, f_0, f_1
\end{align*}
\]

with

\[
\begin{align*}
f \text{mac } c f_0, f_2, f_3
\end{align*}
\]

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

\[
\begin{align*}
32 \text{ bytes} & \quad 1 \text{ cache line} \\
9 + 2 \times (2+2+1+x +2+2+2+2) + 6 = 9+2^*(13+x) = 35 + 2x + 6 \\
41 + 2x & \leq 53 \\
x & \leq 6
\end{align*}
\]

d. Now, assume the cpu in part (a) operates at 2.7 GHZ. Also assume that fmacc takes 8 cycles (N) 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 be run at the same frequency as the processor.

\[
\begin{align*}
\frac{C}{F} = \frac{C'}{F'} \\
53 / 2.7 = (41 + 2^8) / f' \\
f' = (57/53)*2.7
\end{align*}
\]

\[ f' = 2.904 \text{ GHZ} \]