PRACTICE PROBLEMS RELATED TO HOMEWORK 5 (SOLUTIONS SHOULD NOT BE SUBMITTED AND THEY WILL NOT BE GRADED)

Problem 1

Part A
A processor has 32 bit addresses, byte addressing, and no virtual memory. Each row of the table describes a possible cache design. Fill in missing entries.

<table>
<thead>
<tr>
<th>Block Size (bytes)</th>
<th>Associativity</th>
<th>Index Bits</th>
<th>Total Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1 (Direct Mapped)</td>
<td>4-12</td>
<td>8 KB</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5-10</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>4-10</td>
<td>16 KB</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td></td>
<td>16 KB</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>4-12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Size (bytes)</th>
<th>Associativity</th>
<th>Index Bits</th>
<th>Total Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1 (Direct Mapped)</td>
<td>4-12</td>
<td>8 KB</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>5-10</td>
<td>4 KB</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>4-10</td>
<td>16 KB</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>5-12</td>
<td>16 KB</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>4-12</td>
<td>32 KB</td>
</tr>
</tbody>
</table>

Part B
For each of the following types of cache miss, explain what causes the miss (i.e., the definition of the miss). Also, briefly explain how adjusting cache size, and adjusting block size and associativity without changing the cache size will affect the miss rate of each type (discuss the effect of all three parameters).

A. Compulsory
   Miss on the first access. Increased block size reduces this sort of miss. Changing size and associativity do not affect this sort of miss.

B. Capacity
   Miss when the cache did not have enough total space to hold the required data. Increasing cache size will reduce this sort of miss. Associativity does not affect this sort of miss. Increasing block size may increase this sort of miss if spatial locality is poor.

C. Conflict
   Miss when too many other lines were loaded into the same set. Increasing associativity and cache size will reduce conflict misses. Increasing the block size will reduce the number of lines in the cache and increase the conflict miss rate.
Problem 2
Consider a machine $M$ with a simple, 5-stage, single-issue, in-order pipeline that blocks on loads until the requested data is received and blocks on stores until the data is stored in the cache. Consider a single level of separate data and instruction caches and the following characteristics:

- The base CPI with a perfect memory system (i.e., where every memory access takes 1 cycle) is 1.5.
- The hit latency for both I and D caches is 1 cycle.
- Main memory access latency is 40 cycles. This is measured as the time from when the cache issues a main memory request until the time the main memory is ready to deliver the first piece of data. After this latency, both caches receive data/instructions from main memory at the rate of 4 bytes per clock cycle.
- Assume each cache waits for the entire requested block before servicing the processor’s request to any word in the block.
- The block size is 32 bytes for both caches.
- Memory load and store instructions account for 32% of the dynamic instruction mix. 75% of these instructions are loads and 25% are stores.
- Each load and store instruction accesses 64 bits of data. All the instructions are 32-bits long.
- Assume that there is no overlap among stalls resulting from computation, data cache misses, or instruction cache misses.
- Both the data and instruction caches are 8 KB and direct mapped.
- The data cache is write-back, write-allocate. Assume the cache stalls for the write-back before it can issue the request that caused the write-back. Assume the time to write-back a block from the data cache to memory is the same as the time for the cache to read a block from memory.
- The data cache and instruction cache have miss rates of 5% and 3%, respectively.
- Assume that for 40% of the data cache misses, you need to replace a block that is dirty.

Part A
What is the CPI for the machine $M$? Be sure to show your work.

We have the following formula for the overall CPI:

$$CPI = \text{base CPI} + \text{instruction access stalls} + \text{data read miss stalls} + \text{data write miss stalls}$$

We can calculate the individual values as follows. First, cache miss penalty for x bytes = memory access latency + x bytes / data receive rate. For 32 bytes, we have cache miss penalty = 40 cycles + 32 bytes / 4 bytes/cycle = 48 cycles.

We now have the following formulas:

$$\text{Instruction access stalls} = \text{instruction cache miss rate} \times \text{instruction cache miss penalty}$$

$$\text{Data read miss stalls} = \text{data cache miss rate} \times \text{percent load instructions} \times \text{total load/store percent} \times (2 \times \text{cache miss penalty}) \times \text{percent data cache misses with dirty block} + \text{data cache miss penalty} \times (1 - \text{percent data cache misses with dirty block})$$

$$\text{Data write miss stalls} = \text{data cache miss rate} \times \text{percent store instructions} \times \text{total load/store percent} \times (2 \times \text{cache miss penalty}) \times \text{percent data cache misses with dirty block} + \text{data cache miss penalty} \times (1 - \text{percent data cache misses with dirty block})$$
Applying these formulas, we find the following values:

**Instruction access stalls** = 0.03 * 48 cycles = 1.44 cycles

**Data read miss stalls** = 0.05 * 0.75 * 0.32 * ((48 cycles * 2) * 0.4 + 48 cycles * 0.6) = .8064 cycles

**Data write miss stalls** = 0.05 * 0.25 * 0.32 * ((48 cycles * 2) * 0.4 + 48 cycles * 0.6) = .2688 cycles

Thus, we have the following overall CPI using the CPI formula given above:

\[
CPI = 1.5 + 1.44 + .8064 + .2688 = 4.0152
\]

**Part B**

Now consider the machine M, but with the separate I and D caches replaced with a unified 4-way set associative 16 KB cache using a write-through, no write-allocate policy that has a miss rate of 6%. Assume that for writes, writing to the cache and writing to the memory proceed in parallel. What additional assumptions, if any, do you need to make to determine the CPI of this modified machine? Make reasonable assumptions (if you need to) and calculate the CPI.

Several assumptions may need to be made to answer this problem. First, we must assume whether or not there is a write-through buffer. If there is no write-through buffer, we must assume how long it will take to complete a write-through before the processor moves on with execution. However, if a write-through buffer is assumed, we must also assume how often the buffer is full and for how long.

**If no write-through buffer is assumed:**

In this case, we assume it takes \(x\) cycles to execute the write-through.

We have the following formula for the overall CPI:

\[
CPI = base\ CPI + instruction\ access\ stalls + data\ read\ miss\ stalls + data\ write\ miss\ stalls
\]

We can calculate the individual values as follows. First, cache miss penalty for \(x\) bytes = memory access latency + \(x\) bytes / data receive rate. For 32 bytes, we have cache miss penalty = 40 cycles + 32 bytes / 4 bytes/cycle = 48 cycles.

We now have the following formulas:

**Instruction access stalls** = instruction cache miss rate * instruction cache miss penalty

**Data read miss stalls** = data cache miss rate * percent load instructions * total load/store percent * cache miss penalty

**Data write miss stalls** = percent store instructions * total load/store percent * write-through time
Applying these formulas, we find the following values:

Instruction access stalls = 0.06 * 48 cycles = 2.88 cycles
Data read miss stalls = 0.06 * 0.75 * 0.32 * 48 cycles = 0.6912 cycles
Data write miss stalls = 0.25 * 0.32 * x cycles = 0.08x cycles

Thus, we have the following overall CPI using the CPI formula given above:

CPI = 1.5 + 2.88 + 0.6912 + 0.08x

If a write-through buffer is assumed:

We assume the buffer is full y percent of the time and this requires an average stall time of z cycles.

We have the following formula for the overall CPI:

\[ CPI = base\ CPI + instruction\ access\ stalls + data\ read\ miss\ stalls + data\ write\ miss\ stalls \]

We can calculate the individual values as follows. First, cache miss penalty for x bytes = memory access latency + x bytes / data receive rate. For 32 bytes, we have cache miss penalty = 40 cycles + 32 bytes / 4 bytes/cycle = 48 cycles.

We now have the following formulas:

\[ Instruction\ access\ stalls = instruction\ cache\ miss\ rate * instruction\ cache\ miss\ penalty \]
\[ Data\ read\ miss\ stalls = data\ cache\ miss\ rate * percent\ load\ instructions * total\ load/store\ percent * cache\ miss\ penalty \]
\[ Data\ write\ miss\ stalls = percent\ store\ instructions * total\ load/store\ percent * percent\ buffer\ full * buffer\ stall\ time \]

Applying these formulas, we find the following values:

Instruction access stalls = 0.06 * 48 cycles = 2.88 cycles
Data read miss stalls = 0.06 * 0.75 * 0.32 * 48 cycles = 0.6912 cycles
Data write miss stalls = 0.25 * 0.32 * y * z cycles = 0.08yz cycles

Thus, we have the following overall CPI using the CPI formula given above:

CPI = 1.5 + 2.88 + 0.6912 + 0.08yz = 5.0712 + 0.08yz
Problem 3

Consider the following code, which sums the elements of a product of two matrices:

register int i, j, k; /* i, j, k are in the processor registers */
register float sum;

float a[8][8], b[8][8];

for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
        for (k = 0; k < 8; k++) {
            sum += a[i][k] * b[k][j];
        }
    }
}

Assume the following:

- There is a perfect instruction cache; i.e., do not worry about the time for any instruction accesses.
- Both int and float are of size 4 bytes.
- Assume that only the accesses to the arrays a and b generate accesses to the data cache. The rest of the variables are all allocated in registers.
- Assume a fully associative, LRU data cache with 8 lines, where each line is 32 bytes.
- Initially, the data cache is empty.
- The arrays a and b are stored in row major form.
- To keep things simple, we will assume that statements in the above code are executed sequentially. Lines (1), (2), and (3) take 10 cycles for each invocation. Line (4) takes 10 cycles plus an additional 20 cycles per data cache miss to wait for the data. That is, if both array accesses in line (4) miss, it takes a total of 50 cycles.
- Assume that the arrays a and b both start at cache line boundaries.

Part A

How many accesses to arrays a and b will result in cache misses? Explain your answer.

A full row of each array fits in one cache line. There are 8 total lines in the cache. Each access to a within the outermost loop is to the same cache line, so it is never evicted from the LRU cache until the next iteration of the outermost loop. For b, however, the accesses iterate through the 8 rows, and each line is evicted from the cache before it is accessed again. Thus there are:

a: 8 misses
b: 8 * 8 * 8 = 512 misses
520 total misses
Part B

Now assume there is a data prefetch instruction with the format prefetch(array[index1][index2]). This prefetches the entire block containing the word array[index1][index2] into the data cache. It takes 1 cycle for the processor to execute this instruction and send it to the data cache. The processor can then go ahead and execute subsequent instructions. If the prefetched data is not in the cache, it takes 20 cycles for the data to get loaded into the cache.

Add prefetch instructions to the code so as to minimize the execution time. Do not transform the code in any other way. How many cache misses for each of arrays a and b will occur at line (4) in your modified code?

```c
for (i = 0; i < 8; i++) {
    prefetch(a[i][0]);      /* 1 */
    prefetch(b[0][0]);      /* p1 */
    for (j = 0; j < 8; j++) {
        for (k = 0; k < 8; k++) {
            prefetch(b[k+1][0]);  /* p2 */
                sum += a[i][k] * b[k][j];  /* 2 */
            }                        /* p3 */
        }                          /* 3 */
    }
    prefetch(b[0][0]);        /* p4 */
}                               /* 4 */
```

For each iteration of the outermost loop, we can insert one prefetch to avoid the compulsory miss for array a (line p1). Every access to b would be a miss, so we need to insert a prefetch for each of these accesses. In this solution, we prefetch the first row of b before each iteration of the innermost loop (lines p2 and p4), and then a prefetch in the innermost loop for the next iteration (line p3, execution of line 4 and 3 provide the 20 necessary cycles to transfer the data to the cache).

With this modification, there are no cache misses at line 4.
Part C
Now, assume that the prefetch instruction is no longer available. Instead, you have access to a third matrix $c[8][8]$, which is not used anywhere else in the code. Assume that cache misses on a read to $c$ incur the same 20 cycle penalty as $a$ and $b$, but misses on a write are without penalty. Using matrix $c$, transform the code above so that it reduces the number of cache misses during data reads in part A by at least 75%. You may modify and add instructions, assume each one takes 10 cycles to execute plus appropriate data cache miss penalties. How many read misses (to arrays $a$, $b$, and $c$) does your new code have?

Several correct solutions are possible, we list two possibilities:

1) By setting $c$ to the transpose of $b$, we can reduce the number of cache misses:

```c
for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
        c[j][i] = b[i][j];
    }
}
for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
        for (k = 0; k < 8; k++) {
            sum += a[i][k] * c[j][k];
        }
    }
}
```

Each time a new row is accessed for any of $a$, $b$, or $c$, it will miss in the cache. This happens 8 times for array $b$ in line (0.3), 8 times for array $a$ in line (4), and $8 \times 8$ times for array $c$ in line (4).

Total misses: $8 \times 0.3 \times b + 8 \times 4 \times a + 8 \times 8 \times 4 \times c = 80$

2) Alternatively, simply interchanging the order of the $j$ and $k$ for loops meets the goal, without using array $c$. For this approach, $a$ still misses 8 times, while $b$ now only misses 64 times, for a total of 72 misses.