Problem 1. Integrating a processor with a data cache

In this problem, we will consider a program executing on a processor connected to a data cache. The processor is a 5-stage pipelined TinyRV1 processor with full bypassing to resolve data hazards. The data cache is a two-line direct-mapped cache with 8B cache lines.

The program is calculating accumulated sum of an array.

```c
int accu_sum( int src[], int size ) {
    int sum = 0;
    for ( int i = 0; i < size; i++ ) {
        sum += src[i];
        src[i] = sum;
    }
    return sum;
}
```

The simplified assembly instructions which correspond to the loop above are as shown below. Study the assembly instructions and make sure that you understand how it corresponds to the C++ program above. For simplicity, assume the following:

- \( x1 \) : holds the accumulated sum of the \( src[] \) array
- \( x2 \) : holds the base address of the \( src[] \) array
- \( x3 \) : holds the size or loop count

```
loop:
1 lw x4, 0(x2)
2 add x1, x1, x4
3 sw x1, 0(x2)
4 addi x2, x2, 4
5 addi x3, x3, -1
6 bne x3, x0, loop
7 opA
8 opB
9 ...
```

Part 1A Cache Behavior

Assume the data cache is empty initially. We also assume the base address of the \( src[] \) array stored in \( x2 \) is 0x2000. The table below shows the first few dynamic memory transactions. Fill in the table. What's the cache miss rate of this program? Can you think of any way to reduce cache miss rate? Will larger cache help?

<table>
<thead>
<tr>
<th>15</th>
<th>15</th>
<th>0x2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>Index Offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>idx</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>1</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>1</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>1</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>1</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>rd</td>
<td>0x200</td>
<td>0</td>
</tr>
<tr>
<td>wr</td>
<td>0x200</td>
<td>0</td>
</tr>
</tbody>
</table>

Miss rate is 25%.

Larger cache cannot help reduce cache miss rate of this program.

Solutions: 1. Large cache lines.
           2. Software/hardware prefetching.

Part 1.B Pipeline Diagram Illustrating Cache Behavior

Now let’s try to draw the pipeline diagram of the first two iterations of the loop. We also include the first instruction of the third iteration as shown in the instruction sequence.

Assume the miss penalty is two cycles. So on a cache miss, the processor stalls for two extra cycles in the M stage. Note that the cache is not pipelined, so please be aware of potential structural hazards in the M stage.

Draw arrows in your pipeline diagram to indicate any data or control hazards. Ignoring the startup overhead, what’s the CPI of the first iteration? What’s the CPI of the second iteration? What’s the overall CPI if loop size is 64? If cache lines are 16B instead of 8B, what’s the overall CPI?
lw x4, 0(x2)  FDXMMW
addi x1, x1, x4  FDMDXMW
sw x1, 0(x2)  FDDDXMW
addi x2, x2, 4  FDFDXMW
addi x3, x3, -1  FDXXMW
bne x3, x0, loop  FDXMW

opA
opB
lw x5, 0(x2)
addi x3, x1, x4
sw x1, 0(x2)
addi x2, x2, 4
addi x3, x3, -1
bne x3, x0, loop
opA
opB
lw x4, 0(x2)

11 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Start
up

First Iteration

Overhead

Second Iteration

CPI of the first iteration: \( \frac{11}{6} \)

CPI of the second iteration: \( \frac{9}{6} \)

Overall CPI: \( \frac{11 \times 52 + 9 \times 32}{6 \times 64} = \frac{10}{6} = \frac{5}{3} \)

16B cache lines

CPI of the first iteration: \( \frac{11}{6} \)

CPI of the second, third, fourth iteration: \( \frac{9}{6} \)

Overall CPI: \( \frac{11 \times 16 + 9 \times 48}{6 \times 64} = \frac{19}{12} \left( < \frac{5}{3} \right) \)