PROBLEM 1. [15 points] The instruction-type break down of three programs \( P_1 \), \( P_2 \), and \( P_3 \) are given below:

<table>
<thead>
<tr>
<th>Number of instructions</th>
<th>ALU</th>
<th>Load</th>
<th>Store</th>
<th>Branch</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>700</td>
<td>500</td>
<td>200</td>
<td>50</td>
<td>1450</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>700</td>
<td>1330</td>
<td>200</td>
<td>50</td>
<td>2280</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>700</td>
<td>500</td>
<td>2275</td>
<td>50</td>
<td>3525</td>
</tr>
</tbody>
</table>

1.1 [5 points] We are running \( P_1 \), \( P_2 \), and \( P_3 \) on a processor for which any ALU instruction uses 1 clock-cycle and any Branch instruction uses 3 clock-cycles. Suppose that the numbers of clock-cycles for Load and Store instructions are unknown and, that we want to determine them experimentally. Assume that \( P_2 \) and \( P_3 \) run respectively 2 and 3 times slower than \( P_1 \). Deduce the numbers of clock-cycles for a Load instructions and a Store instruction (Ignoring memory-stall cycles).

\[
2 * \#P_1 = \#P_2 \\
3 * \#P_1 = \#P_3 \\
\#cycles_{load} = 5 \\
\#cycles_{store} = 4
\]

Verification in MAPLE:

\[
\begin{align*}
P_1 & := 700 \cdot 1 + 500 \cdot 5 + 200 \cdot 4 + 50 \cdot 3; \\
P_1 & := 4150 \\
P_2 & := 700 \cdot 1 + 1330 \cdot 5 + 200 \cdot 4 + 50 \cdot 3; \\
P_2 & := 8300 \\
P_3 & := 700 \cdot 1 + 500 \cdot 5 + 2275 \cdot 4 + 50 \cdot 3; \\
P_3 & := 12450 \\
P_2 & / P_1 \\
& = 2
\end{align*}
\]
> P3_cycles / P1_cycles ;

1.2 [5 points] Compute the execution time of Program \( P_1 \) for a clock rate of 3 GHz. (Still ignoring memory-stall cycles).

\[
\begin{align*}
(700 \times 1 + 500 \times 5 + 200 \times 4 + 50 \times 3) \times \frac{1}{3 \times 10^9}
&= 1383.333 \times 10^{-9} s = 1383.333 \text{ns}
\end{align*}
\]

1.3 [5 points] Compute the execution time of Program \( P_1 \) on the same processor, now taking into accounts memory-stall cycles. To do so, we assume that

- the processor has a 100 cycle miss penalty,
- Program \( P_1 \) has 2% instruction-cache miss rate, and 3 %data-cache miss rate,

\[
\begin{align*}
&= (2100 + 5000 + 1800 + 250) \times \frac{1}{3 \times 10^9}
&= 3049.695 \text{ns}
\end{align*}
\]

**PROBLEM 2.** [15 points] Let us consider a computer with a L1, L2 and L3 cache memory hierarchy and with the following characteristics:

- Processor Clock Frequency = 1 GHz,
- Hit Time L1 = 1 clock cycle,
- Hit Time L2 = 4 clock cycles,
- Hit Time L3 = 8 clock cycles,
- Miss Penalty L3 = 15 clock cycles.

Assume that, on this computer, we are running a program \( P \) for which Hit Rate L1 = 95%, Hit Rate L2 = 92% and Hit Rate L3 = 90%.

2.1 [5 points] How much is the Global Miss Rate for the three levels of cache (that is the product of the miss rates at levels L1, L2 and L3)?

\[
\text{Miss RateL1} \times \text{Miss RateL2} \times \text{Miss RateL3} = 0.05 \times 0.08 \times 0.1 = 0.04\%
\]

2.2 [5 points] How much is the AMAT?

\[
\text{Miss Penalty L2} = \text{Hit Time L3} + \text{Miss RateL3} \times \text{Miss PenaltyL3}
= (8 + 0.1 \times 15) \text{ clock cycles}
\]
= 9.5 clock cycles

We have:

Miss PenaltyL1 = Hit TimeL2 + Miss RateL2 x Miss PenaltyL2
  = (4 + 0.08 x 9.5) clock cycles
  = 4.76 clock cycles

Thus we have:

AMAT = Hit TimeL1 + Miss RateL1 x Miss PenaltyL1
  = 1 clock cycle + 0.05 * 4.76 clock cycles
  = 1.238 clock cycles

Since a cycle lasts 1.0*10^-10 s, we also have

AMAT = 1.238 clock cycles * 1.0*10^-10 s
  = 0.1232 ns

2.3 [5 points] Given a MAPI (Memory Access Per Instruction, that is, the percentage of Load/Store instructions) of 78% and a CPI_{ideal} of 3, how much is the CPI_{stall} when the hit rate in L3 becomes 100%?

CPI_{stall} = CPI_{ideal} + Avg. memory stall cycles

Avg. memory stall cycles = 0.78 * Miss RateL1 * Miss PenaltyL1

Miss Penalty L2 = Hit Time L3 = 8 (since 100%)
Miss Penalty L1 = Hit TimeL2 + Miss RateL2 * Miss PenaltyL2
  = 4 + 0.08 * 8
  = 4.64 cycles
so, avg. mem. stall cycles = 0.78 * 0.05 * 4.64
  = 0.18 cycles

CPI_{stall} = 3 + 0.18 = 3.18 cycles

PROBLEM 3. [30 points] The following four questions are using this simple cache memory illustrated below.
We specify its key features, which all similar to cache features seen in class:

- byte addressable memory,
- direct-mapped cache,
- the cache has a capacity of 32Kbyte and each block holds 64 bytes, thus the cache can store up to 512 blocks,
- assuming that each variable of type int in a C program uses 4 bytes, this cache can store up to $2^9 \times 2^4 = 2^{13}$ int values,
- byte addresses map into cache as follows:
  - bottom 6 bits are used as offset in a cache block,
  - next 9 bits determine the cache block,
  - top 17 bits give the tag.

For each of the following four questions, we estimate the cache miss rate of a fragment of a program written in the programming language C. Each program fragment reads and/or writes arrays. We assume that:

- the only data being stored in cache is coefficients from those arrays,
- the cache is initially empty when the execution of the program fragment starts,
- the arrays are laid out sequentially in memory, that is, two consecutive array elements are stored in consecutive memory words,
- the arrays are aligned in memory, that is, the first element of each array is at the beginning of a block.

### 3.1 [7 points]

```c
#define S (1<<20)
int A[S];
// Thus size of A is $2^{20}$
for (i = 0; i < S; i++) {
}
```

What are the total numbers of cache misses and cache hits? What kind of locality does it have, if any? What kind of cache misses?

- S reads and S writes to A; each write is a hit.
- 16 elements of A per cache line
- 1/16 miss rate for reads and 0% miss rate for writes [4 points]
- Spatial locality. [2 points]
- Cold misses. [2 points]

### 3.2 [7 points]

```c
#define S (1<<20)
#define T (1<<12)
#define R (1<<8)
int A[S];
// Thus size of A is $2^{20}$
for (j = 0; j < R; j++) {
    for (i = 0; i < T; i++) {
    }
}
```
What are the total numbers of cache misses and cache hits? What kind of locality does it have, if any? What kind of cache misses?

- S reads and S writes to A; each write is a hit.
- 16 elements of A per cache line
- 15 of every 16 hit in cache during the first iteration of the outer loop
- every access hits in the subsequent iteration of the outer loop
- So the total number of read misses is \( T/16 = 256 \). [2 points]
- The total number of read hits is \( T \times 15/16 + (R - 1)T = 1048320 \). [2 points]
- Spatial then temporal locality. [2 points]
- Cold misses. [2 points]

3.3 [8 points]

```c
#define S (1<<20)
#define T (1<<14)
#define R (1<<6)
int A[S];
// Thus size of A is 2^(20)
for (j = 0; j < R; j++) {
  for (i = 0; i < T; i++) {
  }
}
```

What are the total numbers of cache misses and cache hits? What kind of locality does it have, if any? What kind of cache misses?

- S reads and S writes to A; each write is a hit.
- The first T/2 iterations of the inner loop write the entire array A to the cache
- The last T/2 iterations of the inner loop write again the entire array A to the cache
- The total number of read misses is \( R \times 2 \times 2^9 = 65536 \). [2 points]
- The total number of read hits is \( R \times 2 \times 15 \times 2^9 = 983040 \). [2 points]
- Spatial then temporal locality. [2 points]
- Cold and capacity misses. [2 points]

3.4 [8 points]

```c
#define S (1<<19)
int A[S];
int B[S];
int tmp1;
int tmp2;
// Thus, in the main memory, the cache lines of
```
// B are just after all the cache lines of A
for (i = 0; i < S; i++) {
    tmp1 = A[i];
    tmp2 = B[i];
    A[i] = tmp2;
    B[i] = tmp1;
}

What are the total numbers of cache misses and cache hits? What kind of locality
does it have, if any? What kind of cache misses?
- A and B interfere in cache: indeed two cache lines whose addresses differ by a
  multiple of $2^9$ have the same way to cache.
- $2S = 2^{20}$ read misses for A and B. [2 points]
- $2S = 2^{20}$ write misses for A and B. [2 points]
- Spatial locality but the cache cannot exploit it. [2 points]
- Cold and conflict misses. [2 points]

PROBLEM 4. [40 points] Download the archive perf_examples.tgz from the course web
site. Compile the programs mm_naive.c and mm_tiled.c using the Makefile.

4.1 [10 points] Tune the program mm_tiled.c by varying the values of the parameters BLOCK_X,
BLOCK_Y and BLOCK_Z defined in the code at the beginning of the source file. Note that
those parameters must be powers of 2, no greater than 1024. What are the best values
of BLOCK_X, BLOCK_Y and BLOCK_Z on your computer?
The expected values should be 16, 32 or 64. Most likely 16 on an average desktop or
laptop.

4.2 [10 points] What do the two programs do? Which approach does each program take?
Both programs compute the product of two (random) matrices and print the element
of the product at the bottom right corner. [4 points]
mm_naive.c uses the naive (i.e. school-book) for-loop method
[3 points]
mm_tiled.c uses a blocking strategy for increasing temporal locality. [3 points]

4.3 [5 points] Show the CPU info of the machine that your are using for your measurements.
Under Linux, you will find this information in /proc/cpuinfo. On my laptop, for each
of my 4 physical cores, I see
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 58
model name : Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz
stepping : 9
microcode : 0x12
cpu MHz : 1200.000
The important number there is the cache size.

4.4 [15 points] Choose proper performance metrics and use perf to measure them for both programs. Consider to vary the values of $x$, $y$, $z$ (keeping $x = y = z$) to successive powers of two: 512, 2014, 2048, 4096, etc. Which program is faster? Explain briefly why one is faster than the other.

The number of cache misses is substantially higher with `mm_naive.c` due to the following facts:
- C arrays are stored in row-major layout
- `mm_naive.c` makes consecutive accesses to the coefficients of $C$ where those coefficients are not on the same cache-line for $y$ and $z$ large enough
- `mm_tiled.c` uses a block-based strategy where each block fits in cache; hence accessing rows and columns of a block yields cold misses only.

PROBLEM 5. [20 points]
Below is a list of 32-bit word addresses in the main memory:

$$4, 194, 46, 6, 196, 94, 197, 22, 6, 54, 195, 265$$

As in Problem , and as in the lectures, the memory is byte addressable.

5.1 [10 points] Consider a direct-mapped cache with 2-word cache lines. Indicate whether each reference below is either a hit or a miss, assuming the cache is initially empty and is large enough to accommodate all the cache lines referred below.
5.2 [10 points] Consider now a 2-way set associative cache with 2-word cache lines. Indicate whether each reference below is either a hit or a miss, assuming the cache is initially empty and is large enough to accommodate all the cache lines referred below. Use LRU (least recently used) replacement if necessary.

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>00 000</td>
<td>010</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>194</td>
<td>01 100</td>
<td>001</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>46</td>
<td>00 010</td>
<td>111</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>6</td>
<td>00 000</td>
<td>011</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>196</td>
<td>01 100</td>
<td>010</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>94</td>
<td>00 101</td>
<td>111</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>197</td>
<td>01 100</td>
<td>010</td>
<td>100</td>
<td>Hit</td>
</tr>
<tr>
<td>22</td>
<td>00 001</td>
<td>011</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>6</td>
<td>00 000</td>
<td>011</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>54</td>
<td>00 011</td>
<td>011</td>
<td>000</td>
<td>Miss</td>
</tr>
<tr>
<td>195</td>
<td>01 100</td>
<td>001</td>
<td>100</td>
<td>Hit</td>
</tr>
<tr>
<td>265</td>
<td>10 000</td>
<td>100</td>
<td>100</td>
<td>Miss</td>
</tr>
</tbody>
</table>