LAB EXERCISE 5

a.) How many data cache misses will this code generate? Breakdown your answer into the three types of misses. What is the data cache miss rate?

Problem Setup
- Cache Characteristics:
  - Size: 16 KB (16,384 bytes)
  - Associativity: Direct-mapped
  - Line size: 64 bytes
  - Addressing: Physical
- Code:

```
c
for (i = 0; i < 4096; i++) {
    X[i] = X[i] + Y[i] + C;
}
```

  - Arrays `X` and `Y`: 4,096 elements each, 4 bytes per element (single-precision float).
  - Total size per array: 4,096 × 4 = 16,384 bytes.
  - Arrays are consecutive in physical memory.
- Assembly:
  - Loads `X[i]` (4 bytes), loads `Y[i]` (4 bytes), performs multiplication and addition, stores result in `X[i]`.
  - Each iteration: 2 loads (`X[i]`, `Y[i]`), 1 store (`X[i]`).
- Cache Analysis:
  - Cache lines: 16,384 ÷ 64 = 256 lines.
  - Each line holds 64 ÷ 4 = 16 elements of an array.
  - Array size matches cache size (16 KB), but `X` and `Y` together are 32 KB.
  - Direct-mapped: Each memory address maps to one cache line (index = (address ÷ 64) mod 256).

Part a: Data Cache Misses and Miss Rate (Original Code)

Step 1: Memory Accesses
- Each iteration (i from 0 to 4,095):

  - Load `X[i]`: 1 read (4 bytes).
  - Load `Y[i]`: 1 read (4 bytes).
  - Store `X[i]`: 1 write (4 bytes).
- Total iterations: 4,096.
- Total accesses:
  - Reads: 4,096 × 2 = 8,192.
  - Writes: 4,096 × 1 = 4,096.
  - Total: 8,192 + 4,096 = 12,288 data accesses.

Step 2: Cache Miss Analysis
- Memory Layout:
  - `X` starts at address 0 (aligned to a 64-byte boundary).
  - `X` occupies 0 to 16,383 bytes (16 KB).
  - `Y` starts at 16,384, ends at 32,767 bytes.
  - Cache line mapping: Address `A` maps to line `(A ÷ 64) mod 256`.
  - For `X[i]` at address `4i`, line = `(4i ÷ 64) mod 256` = `(i ÷ 16) mod 256`.
  - For `Y[i]` at address `16,384 + 4i`, line = `((16,384 + 4i) ÷ 64) mod 256` = `(256 + (i ÷ 16)) mod 256` = `(i ÷ 16) mod 256`.
  - **Critical Issue**: `X[i]` and `Y[i]` map to the **same cache line** because `(i ÷ 16) mod 256` is identical for both.

- Access Pattern:
  - For each `i`, access `X[i]` (read), `Y[i]` (read), `X[i]` (write).
  - Since `X[i]` and `Y[i]` are in different 64-byte cache lines but map to the same cache line index, they conflict in the direct-mapped cache.

- Miss Breakdown:
  - Compulsory Misses:
    - First access to each cache line is a miss.
    - `X` has 4,096 elements ÷ 16 elements/line = 256 lines.
    - `Y` has 256 lines, but maps to the same cache lines as `X`.
    - Total unique lines accessed: 256 (since `X` and `Y` overlap in cache).
    - Compulsory misses: 256 (first load of each line).
  - Conflict Misses:
    - For each `i`:

- Load `X[i]`: If `i` is the first in a cache line (i.e., `i mod 16 == 0`), it’s a compulsory miss (counted above). Otherwise, it’s a hit if `X[i]`’s line is still in cache.
- Load `Y[i]`: `Y[i]` maps to the same cache line as `X[i]`. If `X[i]`’s line is in cache, loading `Y[i]` evicts `X[i]`’s line (conflict miss).
- Store `X[i]`: Now `Y[i]`’s line is in cache, so storing `X[i]` evicts `Y[i]`’s line (conflict miss).
- For each group of 16 elements (1 cache line):
  - First `X[i]` load: Compulsory miss (1 per line).
  - Subsequent 15 `X[i]` loads: Hits (line already loaded).
  - `Y[i]` loads: Conflict miss (evicts `X`’s line).
  - `X[i]` stores: Conflict miss (evicts `Y`’s line).
- Per 16 elements (256 groups):
  - 16 `X[i]` loads: 1 compulsory miss, 15 hits.
  - 16 `Y[i]` loads: 16 conflict misses.
  - 16 `X[i]` stores: 16 conflict misses.
- Total for 4,096 elements:
  - `X[i]` loads: 256 compulsory misses, 4,096 − 256 = 3,840 hits.
  - `Y[i]` loads: 4,096 conflict misses.
  - `X[i]` stores: 4,096 conflict misses.
- Capacity Misses:
  - The working set (32 KB) exceeds cache size (16 KB), but since `X` and `Y` map to the same lines, the misses are due to conflicts, not capacity exhaustion.
  - Capacity misses: 0.
- Total Misses:
  - Compulsory: 256.
  - Conflict: 4,096 (`Y[i]` loads) + 4,096 (`X[i]` stores) = 8,192.
  - Total: 256 + 8,192 = 8,448.

Step 3: Miss Rate
- Total accesses: 12,288.
- Misses: 8,448.
- Miss rate: 8,448 ÷ 12,288 ≈ 0.6875 (68.75%).

Answer for Part a:
- Misses: 8,448.
  - Compulsory: 256.
  - Capacity: 0.
  - Conflict: 8,192.
- Miss rate: 68.75%.

b.) Provide a software solution that significantly reduces the number of data cache misses. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?
Proposed Solution: Array Merging
- Idea: Instead of separate arrays `X` and `Y`, use a single array of structures where each element contains both `X[i]` and `Y[i]`.
- New Data Structure:
```
c
struct Pair {
    float x;
    float y;
};
struct Pair Z[4096];
```

- **Modified Code**:
```
c
for (i = 0; i < 4096; i++) {
    Z[i].x = Z[i].x + Z[i].y + C;
}
```

- Effect:
  - Each `Z[i]` is 8 bytes (`x` and `y`).
  - Total size: 4,096 × 8 = 32,768 bytes.
  - `Z[i].x` and `Z[i].y` are adjacent in memory, likely in the same cache line.
  - Cache line (64 bytes) holds 64 ÷ 8 = 8 `Pair` elements.
  - Total lines needed: 4,096 ÷ 8 = 512 lines, but cache has only 256 lines.

Cache Miss Analysis
- Accesses:
  - Each iteration:
    - Load `Z[i].x`: 1 read.
    - Load `Z[i].y`: 1 read.
    - Store `Z[i].x`: 1 write.
  - Total:
    - Reads: 4,096 × 2 = 8,192.
    - Writes: 4,096.
    - Total: 12,288 (same as original).
- Miss Breakdown:
  - Compulsory Misses:
    - 512 unique lines accessed (32 KB ÷ 64 bytes).
    - First access to each: 512 compulsory misses.
  - Capacity Misses:
    - Working set (32 KB) exceeds cache size (16 KB).
    - After filling 256 lines, additional lines cause capacity misses.
    - Assume sequential access:
      - Lines 0–255 fit in cache.
      - Lines 256–511 evict lines 0–255 (direct-mapped).
      - Second half of accesses (i ≥ 2,048, lines 256–511) cause misses when re-accessing earlier lines.
    - For each group of 8 elements (1 cache line):
      - 8 `Z[i].x` loads, 8 `Z[i].y` loads, 8 `Z[i].x` stores.
      - First access to line: Compulsory miss.
      - Subsequent accesses in same line: Hits.
    - Total lines: 512.
    - Compulsory: 512 misses.
    - Capacity: When accessing lines 256–511, they evict lines 0–255. If we assume a simple model, each line access after the first 256 is a miss due to capacity.
    - Total accesses per line: 8 reads + 8 writes = 16 accesses.
    - For 512 lines: 512 × 16 = 8,192 accesses.
    - First access per line: 512 misses (compulsory).
    - Remaining accesses: 8,192 − 512 = 7,680.

  - Assume half the lines (256–511) miss due to capacity: 256 lines × 16 accesses = 4,096 capacity misses.
- Conflict Misses:
  - Since `Z[i].x` and `Z[i].y` are in the same line, no conflicts between them.
  - Conflict misses: 0.
- Total Misses:
  - Compulsory: 512.
  - Capacity: 4,096.
  - Conflict: 0.
  - Total: 512 + 4,096 = 4,608.

Miss Rate
- Accesses: 12,288.
- Misses: 4,608.
- Miss rate: 4,608 ÷ 12,288 ≈ 0.375 (37.5%).

Answer for Part b:
- Software solution: Merge `X` and `Y` into an array of structures.
- Misses: 4,608.
  - Compulsory: 512.
  - Capacity: 4,096.
  - Conflict: 0.
- Miss rate: 37.5%.

c.) Provide a hardware solution that significantly reduces the number of data cache misses. You are free to alter the cache organization and/or the processor. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?

Proposed Solution: Increase Cache Associativity
- Idea: Change the cache to 4-way set-associative (same size: 16 KB, line size: 64 bytes).
- Effect:
  - Sets: 16,384 ÷ (64 × 4) = 64 sets.

  - Each set holds 4 lines.
  - `X` and `Y` lines that previously conflicted can now coexist in different ways of the same set.
  - `X` and `Y` each need 256 lines (16 KB ÷ 64).
  - Total lines: 512, but only 256 lines in cache.
  - With 4-way associativity, conflicts are reduced unless more than 4 lines map to the same set.

Cache Miss Analysis
- Accesses: Same as original (12,288).
- Miss Breakdown:
  - Compulsory Misses:
    - `X`: 256 lines.
    - `Y`: 256 lines.
    - Total unique lines: 512.
    - Compulsory misses: 512.
  - Conflict Misses:
    - `X` lines map to sets: `(i ÷ 16) mod 64`.
    - `Y` lines map to same sets (offset by 16,384 bytes, but modulo 256 lines).
    - Each set can hold 4 lines, so `X` and `Y` lines (2 per set) fit without conflict.
    - No conflict misses unless capacity is exceeded.
  - Capacity Misses:
    - Cache holds 256 lines.
    - Working set: 512 lines.
    - After 256 lines, additional lines evict earlier ones.
    - Assume LRU replacement:
      - First 256 lines (`X[0–2047]`, `Y[0–2047]`): Compulsory misses.
      - Second 256 lines (`X[2048–4095]`, `Y[2048–4095]`): Evict earlier lines.
      - Total misses: 512 compulsory (first access to each line).
      - Capacity misses: When accessing later lines, some evictions occur, but 4-way associativity minimizes conflicts.
      - Conservative estimate: 256 additional capacity misses for second half.
  - Total Misses:

  - Compulsory: 512.
  - Capacity: 256.
  - Conflict: 0.
  - Total: 512 + 256 = 768.

Miss Rate
- Accesses: 12,288.
- Misses: 768.
- Miss rate: 768 ÷ 12,288 ≈ 0.0625 (6.25%).

Answer for Part c:
- Hardware solution: 4-way set-associative cache.
- Misses: 768.
  - Compulsory: 512.
  - Capacity: 256.
  - Conflict: 0.
- Miss rate: 6.25%.