HOMEWORK-4

1. Cache Reads:

For program 1 with (8,8) , the hit rates were as follows:

LINES	BLOCKS	WAYS	HIT RATE	Hits	Misses	Total accesses
1	2	0	0.7424	49	17	66
2	2	0	0.7424	49	17	66
3	2	0	0.7424	49	17	66
3	3	0	0.8636	57	9	66
3	4	0	0.9242	61	5	66
3	5	0	0.9545	63	3	66
3	2	0	0.7424	49	17	66
3	2	1	0.7424	49	17	66

For program 1 with (16,16), the hit rates were as follows:

LINES	BLOCKS	WAYS	HIT RATE	Hits	Misses	Total accesses
1	2	0	0.7481	193	65	258
2	2	0	0.7481	193	65	258
3	2	0	0.7481	193	65	258
3	3	0	0.8721	225	33	258
3	4	0	0.9341	241	17	258
3	5	0	0.9651	249	9	258
3	2	0	0.7481	193	65	258
3	2	1	0.7481	193	65	258

For program 2 with (8,8), the hit rates were as follows:

LINES	BLOCKS	WAYS	HIT RATE	Hits	Miss	Total accesses
1	2	0	0.0303	2	64	66
2	2	0	0.0303	2	64	66
3	2	0	0.04545	2	64	66
3	3	0	0.8636	57	9	66
3	4	0	0.9242	61	5	66
3	5	0	0.9545	63	3	66
3	2	0	0.04545	2	64	66
3	2	1	0.7424	49	17	66

For program 2 with (16,16), the hit rates were as follows:

LINES	BLOCKS	WAYS	HIT RATE	Hits	Miss	Total Accesses
1	2	0	0.007752	2	256	258
2	2	0	0.007752	2	256	258
3	2	0	0.007752	2	256	258
3	3	0	0.007752	2	256	258
3	4	0	0.01163	3	255	258
3	5	0	0.9574	247	11	258
3	2	0	0.007752	2	256	258
3	2	1	0.007752	2	256	258

OBSERVATION:

Impact of Increasing Cache Lines:

- **Observation**: Increasing the number of lines from 1 to 3 does not improve the hit rate .
- **Why**: This occurs because the program's memory access pattern may not take full advantage of more cache lines. If the program repeatedly accesses the same small set of data, adding more lines won't help since the data being accessed already fits within the existing cache lines.

Effect of Block Size:

- **Observation**: Increasing the block size from 2 to 5 significantly improves the hit rate.
- **Why**: Larger blocks fetch more consecutive data from memory in a single cache line. If the program has spatial locality (i.e., it accesses nearby memory addresses), larger blocks will reduce the number of cache misses because more relevant data is loaded into the cache in each access.

Influence of Cache Associativity (Ways):

- **Observation**: Introducing associativity (moving from 0 ways to 2 ways) does not improve the hit rate.
- **Why**: Associativity helps when there are conflicts between data mapping to the same cache location. In this case, the program might not have enough conflicts for associativity to make a difference, meaning that most of the data does not map to the same set of cache lines, reducing the need for more ways.

OBSERVATION

Impact of Increasing Cache Lines:

- **Observation**: The hit rate remains low when increasing the number of cache lines from 1 to 2 but improves significantly with 3 lines.

- **Why:** In Program 2, memory accesses involve shifts (slli instructions), which result in non-contiguous access patterns. With fewer cache lines, not enough data is retained to reduce cache misses. When the cache has 3 or more lines, it can hold more relevant data, reducing cache misses as a result.

Effect of Block Size:

- **Observation:** Increasing the block size from 2 to 5 results in a significant improvement in the hit rate.
- **Why:** As in Program 1, larger block sizes allow more consecutive memory addresses to be fetched at once. Although Program 2's memory accesses are less contiguous (due to slli and add instructions), fetching larger blocks still reduces misses because nearby memory data may be reused soon after.

Influence of Cache Associativity:

- **Observation:** Increasing associativity to 1 or 2 ways slightly improves the hit rate
- **Why:** Associativity helps reduce conflict misses by allowing multiple cache entries to share the same cache set. However, Program 2 has fewer cache conflicts, so associativity doesn't have as large of an effect.

Program 1 accesses memory in a more **contiguous** pattern, iterating over the memory. This means that smaller block sizes and fewer cache lines can still achieve a good hit rate because the program accesses data in a predictable, sequential manner. Most data accesses hit the same cached block, leading to fewer cache misses.

Program 2 accesses memory in a more **non-contiguous** manner due to the use of the slli instruction, which shifts the memory addresses. This leads to scattered memory accesses, requiring larger cache lines and blocks to store a broader range of data. This results in more cache misses unless larger block sizes or more cache lines are used to cover more memory addresses.

Both show little improvement from increased associativity because their access patterns do not frequently cause conflicts where multiple memory addresses map to the same cache line.

2. Write – policies:

   After considering 32-entry 4-word direct mapped cache which means 2^n, n is given as lines – 5, ways-0, blocks – 2 and changing "ld x20, 0(x12)" to "sd x20, 0(x12)" for program 1 and program 2.

   For program 1 with .dword (8,8) the hit rates are,

Wr. hit	Wr. Miss	Hit rate	Hits	Misses	Total accesses
Write-through	Write allocate	0.7424	49	17	66
Write-through	No Write allocate	0.04545	3	63	66
Write-back	Write allocate	0.7424	49	17	66
Write-back	No Write allocate	0.04545	3	63	66

For program 1 with .dword (16,16) the hit rates are,

Wr. hit	Wr. Miss	Hit rate	Hits	Misses	Total accesses
Write-through	Write allocate	0.7481	193	65	258
Write-through	No Write allocate	0.01163	3	255	258
Write-back	Write allocate	0.7481	193	65	258
Write-back	No Write allocate	0.01163	3	255	258

For program 2 with .dword (8,8) the hit rates are,

Wr. hit	Wr. Miss	Hit rate	Hits	Misses	Total accesses
Write-through	Write allocate	0.7424	49	17	66
Write-through	No Write allocate	0.04545	3	63	66
Write-back	Write allocate	0.7424	49	17	66
Write-back	No Write allocate	0.04545	3	63	66

For program 2 with .dword (16,16) the hit rates are,

Wr. hit	Wr. Miss	Hit rate	Hits	Misses	Total accesses
Write-through	Write allocate	0.01163	3	255	258
Write-through	No Write allocate	0.01163	3	255	258
Write-back	Write allocate	0.01163	3	255	258
Write-back	No Write allocate	0.01163	3	255	258

Write Allocate:

- Loading the block into the cache after a write miss allows future accesses (whether reads or writes) to hit in the cache, which reduces memory access time and increases the hit rate.
- Programs with good locality (i.e., they access the same data or nearby data multiple times) benefit significantly from this policy. That's why the hit rate is high (0.7424) when Write Allocate is used.

No Write Allocate:

- Since the cache isn't involved after a write miss, subsequent accesses to the same address continue to miss the cache. This leads to a lower hit rate (0.04545).
- This policy is more efficient in some scenarios where the program doesn't frequently access the same data, but for programs that do (like Program 1), it significantly worsens cache performance.

The hit rate increases from 8,8 to 16,16 in Write Allocate for Program 1 because the larger word size (16,16) allows better cache utilization by reducing the frequency of misses, leading to more efficient use of cache blocks.

The hit rate decreases for Program 2 from 8,8 to 16,16 because the larger word size (16,16) likely exceeds the cache's capacity or worsens the memory access pattern, causing more cache misses due to less effective cache utilization.

All hit rates same (0.01163):

- The hit rates for Program 2 with .dword sizes (16,16) remain 0.01163 across all policies (Write-through/Write-back with and without allocate). This suggests that the cache behaviour for Program 2 is dominated by cache misses due to the access pattern or memory footprint of the program.
- This means Program 2 might be accessing a much larger data set, or its memory access patterns aren't favourable to the cache configuration.

3. Associativity

   For program 3 with .dword (8,8) , the hit rates are given as:

Configurations	Hit rate	Hits	Miss	Total Accesses
32-entry 4-word direct mapped	0.01538	2	128	130
32-entry 4-word 2-way set associative	0.7385	96	34	130
32-entry 4-word fully associative	0.7385	96	34	130

   For program 3 with .dword (16,16) , the hit rates are given as:

Configurations	Hit rate	Hits	Miss	Total accesses
32-entry 4-word direct mapped	0.06809	35	479	514
32-entry 4-word 2-way set associative	0.7471	384	130	514
32-entry 4-word fully associative	0.7471	384	130	514

Direct Mapped Cache :

- Very low hit rate for both configurations of .dword values.
- Direct-mapped caches have a simple structure but suffer from conflict misses. In the given program, consecutive memory accesses result in conflicts, as multiple addresses map to the same cache line. Hence, the cache misses frequently and the hit rate remains low.

2-way Set Associative Cache :

- Significant improvement in hit rate compared to direct-mapped.
- The 2-way set associative cache allows more flexibility by giving two possible locations for each block. This reduces conflict misses and improves the hit rate, especially with more evenly distributed accesses in memory.

Fully Associative Cache :

- Same hit rate as the 2-way set associative cache.
- In a fully associative cache, any block can be placed in any location, which theoretically should reduce misses further. However, for this program, the 2-way set associative structure seems to offer enough flexibility, leading to a similar hit rate as fully associative