CO224 – Computer Architecture **Group 49: Nawanjana A.L.D. (E/19/256)**

Lab 6 – Building a Memory Hierarchy **Bandara M.G.R.S. (E/19/038)**

Performance Comparison of Cached and Cache-less Systems Report

In the 2nd part of Lab 06, the CPU's data memory hierarchy was enhanced by incorporating a data cache module between the data memory and the CPU to reduce access delays. To evaluate performance differences, sample programs were run on both the previous cache-less setup and the new configuration, and the delay variations were observed.

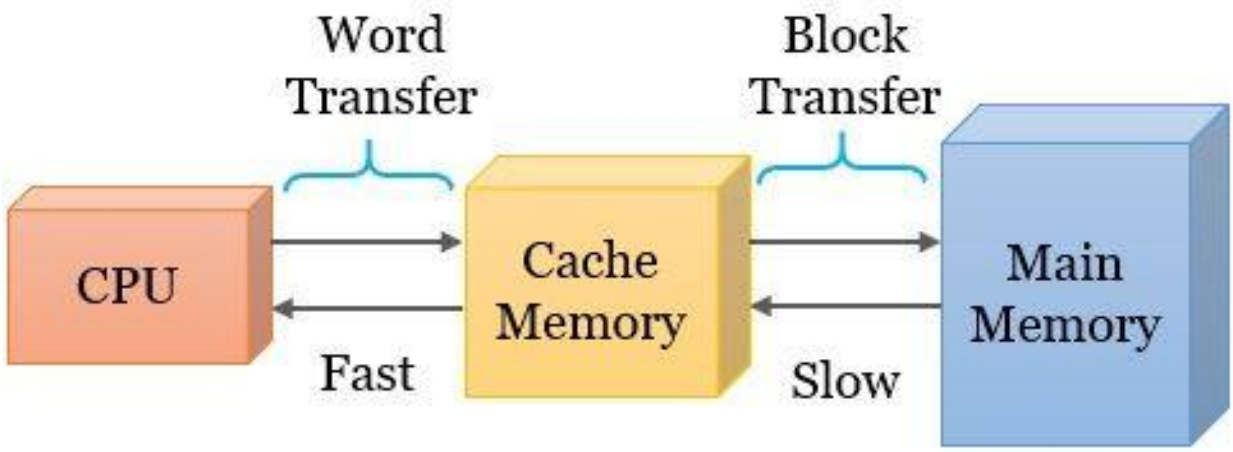

Figure 01: Cache Memory and Main Memory

In the cache-less configuration, each data access resulted in a delay of 40 units and stalled the CPU for 5 clock cycles.

In contrast, the data cache's performance depended on the cache state. On a cache hit, data was served within the same clock cycle without stalling the CPU, significantly improving performance compared to the cache-less setup.

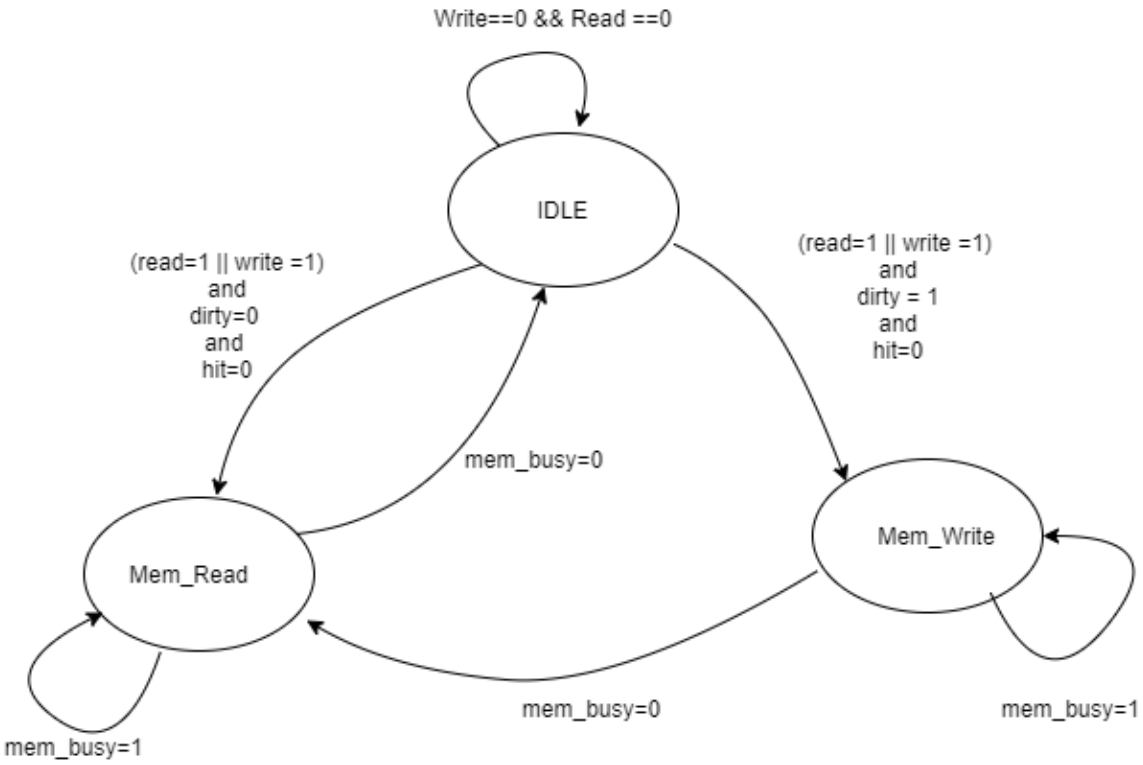

Figure 02: Cache Controller FSM

However, a cache miss led to considerable delays: 21 clock cycles if the cached data wasn't dirty, and 42 clock cycles if it was. Thus, performance in case of misses was worse than in the cache-less system.

This highlighted a compromise with the inclusion of cache memory:

high hit rates improved performance over the cache-less setup, while high miss rates degraded it. Therefore, optimizing block size to suit the CPU's needs and writing cache-friendly code can help manage miss rates, ensuring that the data memory hierarchy outperforms the cache-less configuration.

Instruction Execution Flow

Instruction: loadi 0 0x09

- Load the immediate value 0x09 into register 0. This instruction does not access memory, so no cache hit or miss.

Instruction: loadi 1 0x01

- Load the immediate value 0x01 into register 1. This instruction does not access memory, so no cache hit or miss.

Instruction: swd 0 1

- Store the word from register 0 into the memory address specified by register 1 (0x01).
- Memory address: 0x01
- Index: Block (0x01 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00
- Offset: 0x01 % 4 = 1
- Action: Cache miss (block is empty), store 0x09 to Block 0 at offset 1, set valid bit and dirty bit.

Instruction: swi 1 0x00

- Store the immediate value 0x01 to memory address 0x00.
- Memory address: 0x00
- Index: Block (0x00 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00
- Offset: 0x00 % 4 = 0
- Action: Cache hit (Block 0 is valid and tag matches), store 0x01 to Block 0 at offset 0, set dirty bit.

Instruction: lwd 2 1

- Load the word from memory address specified by register 1 (0x01) into register 2.
- Memory address: 0x01
- Index: Block (0x01 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00 Offset: 0x01 % 4 = 1
- Action: Cache hit (Block 0 is valid and tag matches), load 0x09 from Block 0 at offset 1 into register 2.

Instruction: lwd 3 1

- Load the word from memory address specified by register 1 (0x01) into register 3.
- Memory address: 0x01
- Index: Block (0x01 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00
- Offset: 0x01 % 4 = 1
- Action: Cache hit (Block 0 is valid and tag matches), load 0x09 from Block 0 at offset 1 into register 3.

Instruction: sub 4 0 1

- Subtract the value in register 1 from register 0 and store the result in register 4. This instruction does not access memory, so no cache hit or miss.

Instruction: swi 4 0x02

- Store the immediate value (result of sub) into memory address 0x02. Memory address: 0x02
- Index: Block (0x02 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00
- Offset: 0x02 % 4 = 2
- Action: Cache hit (Block 0 is valid and tag matches), store the result into Block 0 at offset 2, set dirty bit.

Instruction: lwi 5 0x02

- Load the word from memory address 0x02 into register 5.
- Memory address: 0x02
- Index: Block (0x02 / 4) % 8 = 0 % 8 = Block 0
- Tag: 0x00 Offset: 0x02 % 4 = 2
- Action: Cache hit (Block 0 is valid and tag matches), load the value from Block 0 at offset 2 into register 5.

Instruction: swi 4 0x20

- Store the immediate value from register 4 into memory address 0x20.
- Memory address: 0x20
- Index: Block (0x20 / 4) % 8 = 8 % 8 = Block 0
- Tag: 0x02 Offset: 0x20 % 4 = 0
- Action: Cache miss (tag mismatch), evict Block 0 if dirty, store the value into new Block 0 with tag 0x02, set valid bit and dirty bit.

Instruction: lwi 6 0x20

- Load the word from memory address 0x20 into register 6.
- Memory address: 0x20
- Index: Block (0x20 / 4) % 8 = 8 % 8 = Block 0
- Tag: 0x02
- Offset: 0x20 % 4 = 0
- Action: Cache hit (Block 0 is valid and tag matches), load the value from Block 0 at offset 0 into register 6

Conclusion with Examples from the Test Program

The conclusion highlights how the performance of a cached system significantly outperforms a cache-less system, based on the examples provided in the test program. The examples illustrate how cache hits and misses affect memory access latency, showing the benefits of using a cache memory.

Cached System:

- For instructions with cache hits, the data is quickly accessed from the cache, providing faster execution. For example, the `*lwd 2 1*` instruction benefits from a cache hit, where the data 0x09 is loaded into register 2 from the cache.
- Cache misses incur some latency, but subsequent accesses to the same data benefit from faster cache hits. For instance, the `*swd 0 1*` instruction initially causes a cache miss, but once the data is in the cache, subsequent accesses are faster.

Cache-less System:

- Every memory access requires fetching data directly from the main memory, which is significantly slower. This would be akin to every operation being a cache miss.
- For instance, the `*lwd 2 1*` instruction would always involve fetching data from the main memory, resulting in higher latency compared to the cached system where a cache hit speeds up the process.

To further enhance performance, it is crucial to optimize the cache configuration, such as adjusting the block size and associativity, and to write cache-friendly code. These strategies help manage miss rates effectively, ensuring that the data memory hierarchy consistently outperforms a cache-less configuration.