Leondi Soetojo

Project 2 Report

First of all, we need to determine the size of index and the number of bits in physical memory. We are given that 2048 bytes of MM and each cache has 1 byte block with 16 sets as the total. Therefore, we initially determine the number of bits in physical memory, then we can move to determine the offset and the number of bits for index for Direct-Mapped.

- Bits in Physical Memory: 32.
- Bits in Offset: $\text{Log}_2(1) = 0$.
- Bits in Index: $\text{Log}_2(16) = 4$.
- Bits in Tag: 32 – 4 – 0 = 28.

However, it will be a bit different with another type, such as Fully Associative and 4-way set-associative cache. In Fully Associative, we will not have index bits, so we can get,

- Bits in Physical Memory: 32.
- Bits in Offset: $\text{Log}_2(1) = 0$.
- Bits in Index: 0.
- Bits in Tag: 32 – 0 – 0 = 32.

On the other hand, in 4-way set-associative cache case, we need to divide the total sets by 4, which is 4 sets. Therefore, the number will be,

- Bits in Physical Memory: 32.
- Bits in Offset: $\text{Log}_2(1) = 0$.
- Bits in Index: $\text{Log}_2(4) = 2$.
- Bits in Tag: 32 – 2 – 0 = 30.

Result

File: mini_debug.txt

mini_debug <0>:

- Load Access: 20
- PropLoads: 20/104 = 0.192
- Store Access: 20
- PropStores: 20/104 = 0.192
- Number of Instruction: 40
- Number of Miss: 16
- Number of Hits: 4
- Number of Cycle: 104

- Miss Rate: 0.8
- (104, 0.8)
- IPC = 0.6 + 0.4 (0.192 + 0.192 [(5 * 0.8) + (1 – 0.8)]) = 0.9994

mini_debug <1>:

- Load Access: 20
- PropLoads: 20/96 = 0.208
- Store Access: 20
- PropStores: 20/96 = 0.208
- Number of Instruction: 40
- Number of Miss: 14
- Number of Hits: 6
- Number of Cycle: 96
- Miss Rate: 0.7
- (96, 0.7)
- IPC = 0.6 + 0.4 (0.208 + 0.208 [(5 * 0.7) + (1 – 0.7)]) = 0.9994

mini_debug <2>:

- Load Access: 20
- PropLoads: 20/96 = 0.208
- Store Access: 20
- PropStores: 20/96 = 0.208
- Number of Instruction: 40
- Number of Miss: 14
- Number of Hits: 6
- Number of Cycle: 96
- Miss Rate: 0.7
- (96, 0.7)
- IPC = 0.6 + 0.4 (0.208 + 0.208 [(5 * 0.7) + (1 – 0.7)]) = 0.9994

File: debug.txt

debug <0>:

- Load Access: 100
- PropLoads: 100/386 = 0.259
- Store Access: 110
- PropStores: 110/386 = 0.285
- Number of Instruction: 210
- Number of Miss: 44

- Number of Hits: 56
- Number of Cycle: 386
- Miss Rate: 0.44
- (386, 0.44)
- IPC = 0.6 + 0.4 (0.285 + 0.259 [(5 * 0.44) + (1 – 0.44)]) = 0.9999

debug <1>:

- Load Access: 100
- PropLoads: 100/358 = 0.279
- Store Access: 110
- PropStores: 110/358 = 0.307
- Number of Instruction: 210
- Number of Miss: 37
- Number of Hits: 63
- Number of Cycle: 358
- Miss Rate: 0.37
- (358, 0.37)
- IPC = 0.6 + 0.4 (0.307 + 0.279 [(5 * 0.37) + (1 – 0.37)]) = 0.9995

debug <2>:

- Load Access: 100
- PropLoads: 100/362 = 0.276
- Store Access: 110
- PropStores: 110/362 = 0.304
- Number of Instruction: 210
- Number of Miss: 38
- Number of Hits: 62
- Number of Cycle: 362
- Miss Rate: 0.38
- (362, 0.38)
- IPC = 0.6 + 0.4 (0.304 + 0.276 [(5 * 0.38) + (1 – 0.38)]) = 0.9998

Question

1. Based on my result, the cache design that has the highest miss rate is Direct-Mapped.
2. We have a formula of:

$$IPC = 0.6 + 0.4(PropStores + PropLoads[(5 * MissRate) + (1 - MissRate)])$$