Gaurav Ghop gg2961

NYU Tandon School of Engineering
Fall 2024, ECE 6913
Homework Assignment 8

1St Question

(a) In this configuration, each block holds a single word and is directly assigned to a specific set within the cache. Misses occur when multiple addresses are mapped to the same set. For a direct-mapped cache, where b=1 (each block contains one word), only one block is present in each set (N=1). Assuming the cache contains SS sets, the block offset becomes log2(b)= 0 bits. Formula to find set index is given below:

SetIndex = (Address / b) mod S

Each word belongs to a particular set; however, conflict miss occurs when the multiple words will be mapping to the same set due to which one replaces other. To review the record of cache hits and misses, analyze the sequence of memory accesses.

(b) In a fully associative cache, there is only one set in which all blocks reside and this actually eliminates conflict misses. Only compulsory misses due to limited cache capacity, are obtained in this configuration. Since N=S, the whole cache is one single set composed of all the blocks.With this configuration, therefore, all conflict misses are avoided leaving just compulsory and capacity misses to consider. The process of replacement, upon full cache, is done by replacing the block that has been least recently used in place of the new coming data. In analyzing cache performance, it is very vital that the tracking of the manner of access and management of blocks be based on an LRU replacement policy.

(c) Each set in this two-way set associative cache has two blocks to reduce the number of conflicts compared to a direct mapped cache. Misses depend on how addresses are mapping to the sets and also on the Least Recently Used (LRU) policy. Here, $N = 2$, that is, two blocks per set. To find out the total number of sets:

S = Total Blocks / 2
The set index is determined by the following equation:

SetIndex = (Address / b) mod S

Conflict misses can arise if more than two blocks are mapped to the same set. In order to manage block replacements in each set, the LRU policy is implemented.

(d) Each block holds two words, thereby reducing the number of blocks and reducing miss rates on sequential accesses. Also, most of the time, consecutive addresses map to the same block.

When b = 2, the number of blocks is reduced by half. The mapping of address to set can be obtained by

SetIndex = (Address / b) mod S
Access to consecutive addresses within the same block results in hits after the initial access.
Steps to Calculate Miss Rate
1. Parse the sequence of addresses into blocks based on, and map these addresses to sets.
2. For each cache configuration:
Keep track of hits and misses using the replacement policy and address mapping.
3. Compute the miss rate as
MissRate = Misses / Total Accesses.

2nd Question

a) We assume that the hit time is negligible.
AMAT can be computed by the following equation:
AMAT = Hit time + (Miss rate × Miss penalty)
Data cache miss rate is 15%, Miss penalty is 200 ns. Access time to main memory (hit time) = 0
as the instruction cache always hits
AMAT = 0 + (0.15 × 200) = 30 ns
The AMAT is 30 ns

b) Given:
Processor speed = 1 GHz (1 cycle per ns)
Data cache miss penalty = 200 ns, equivalent to 200 cycles
Data cache miss rate = 15% (0.15 in decimal form) The formula to calculate the CPI is given by: CPI = 1 + (Miss rate × Miss penalty in cycles) Putting the values:
CPI = 1 + (0.15 × 200)
Simplifying the equation: CPI = 1 + 30 = 31
Thus, the CPI is 31.
On average, load and store word instructions incur an extra 30 cycles due to data cache misses, bringing the Total CPI of 31 for those instructions.

c) The average CPI depends on the weighted sum of the frequencies of each type of instruction and their respective CPIs. The formula used is: The formula to calculate the average CPI is: Average CPI = Σ(Frequency of each type × CPI of that type) This formula considers the weighted contribution of each instruction type based on its frequency and CPI.
Load/store instructions: 25% of the total, with a CPI of 31 (as calculated in part Other instructions: 75% of the total, with a CPI of 1 Substituting the values into the formula:
The average CPI is calculated as follows:
Average CPI = 0.25 × 31 + 0.75 × 1
Simplify the terms: Average CPI = 7.75 + 0.75 Final result:
Average CPI = 8.5
The average CPI for the benchmark application is 8.5. The load/store instructions, that account for 25% of all the instructions executed, drive up the overall CPI very high due to its large miss penalty while all other instructions have a CPI of 1.

d) Now, assume an instruction cache miss rate of 10% (0.10 as a decimal) combined with the data cache miss rate and penalty.

CPI = Base CPI + (Instruction cache miss rate × Miss penalty)

This equation takes into account the base CPI and adds the additional delay due to instruction cache misses.

Here, the miss penalty for both instruction and data cache misses is 200 cycles.

Non-Load/Store Instructions:

The CPI for non-load/store instructions is computed as:

CPI for non-load/store = 1 + (0.10 × 200)

Simplifying the equation:

CPI for non-load/store = 1 + 20

Therefore,

CPI for non-load/store = 21

Overall Average CPI:

Now, the overall average CPI can be obtained by taking a weighted contribution of load/store instructions and non-load/store instructions:

The average CPI is given by:

Average CPI = 0.25× 31 + 0.75 × 21

Average CPI = 7.75 + 15.75

Average CPI = 23.5

When instruction cache misses are taken into account, all instructions suffer a further delay because of the miss penalty. This raises the general average CPI to 23.5.