**Week 4 – Homework Problems**

Eric Vara

The University of Arizona Global Campus

CPT 301: Computer Organization & Architecture

Professor Austine Ohwobete

December 4, 2023

**Chapter 5  
Exercise 5.1.1**

Four 32-bit integers can be stored in a 16-byte cache block.

**Exercise 5.1.2**

The variables I, J, and the array elements A[I][J] exhibit temporal locality.

**Exercise 5.1.3**

The array elements A[I][J] and B[I][J] exhibit spatial locality.

**Exercise 5.1.4**

From the code provided, it appears that the matrix has 8 rows and 18,000 columns, and each element is a 32-bit integer. In computing, a 32-bit integer is typically 4 bytes in size.

The total size of the matrix in bytes is calculated by multiplying the number of elements by the size of each element. Since the matrix has 8 rows and 18,000 columns, it has 8×18,000 elements. Multiplying this by 4 bytes gives us the total size of the matrix in bytes.

The number of 16-byte blocks needed is then calculated by dividing the total size of the matrix by 16 bytes per block. To store all the 32-bit matrix elements being referenced, you would need 36,000 cache blocks that are 16 bytes each.

**Exercise 5.2.1**

Each address results in a miss because the cache starts empty and each new address loads a new block into the cache.

**Exercise 5.2.2**

* Address 3: Binary – 00000000000000000000000000000011  
  Tag - 0000000000000000000000000000, Index - 0011
* Address 180: Binary – 00000000000000000000000010110100

Tag - 0000000000000000000000001011, Index - 0100,

* Address 43: Binary - 00000000000000000000000000101011

Tag - 0000000000000000000000000010, Index - 1011,

* Address 2: Binary - 00000000000000000000000000000010

Tag - 0000000000000000000000000000, Index - 0010,

* Address 191: Binary - 00000000000000000000000010111111

Tag - 0000000000000000000000001011, Index - 1111,

* Address 88: Binary - 00000000000000000000000001011000

Tag - 0000000000000000000000000101, Index - 1000,

* Address 190: Binary - 00000000000000000000000010111110

Tag - 0000000000000000000000001011, Index - 1110,

* Address 14: Binary - 00000000000000000000000000001110

Tag - 0000000000000000000000000000, Index - 1110,

* Address 181: Binary - 00000000000000000000000010110101

Tag - 0000000000000000000000001011, Index - 0101,

* Address 44: Binary - 00000000000000000000000000101100

Tag - 0000000000000000000000000010, Index - 1100,

* Address 186: Binary - 00000000000000000000000010111010

Tag - 0000000000000000000000001011, Index - 1010,

* Address 253: Binary - 00000000000000000000000011111101

Tag - 0000000000000000000000001111, Index - 1101,

**Exercise 5.2.3**

For each of the given memory addresses, I’ve calculated the binary address, the tag, the index (with and without the word offset), and determined whether each reference is a cache hit or a miss in a direct-mapped cache with 8 two-word blocks. Here are the details for each address:

* Address 3: Index with Offset - 0011, Index - 001, Miss
* Address 180: Index with Offset - 0100, Index - 010, Miss
* Address 43: Index with Offset - 1011, Index - 101, Miss
* Address 2: Index with Offset - 0010, Index - 001, Hit
* Address 191: with Offset - 1111, Index - 111, Miss
* Address 88: Index with Offset - 1000, Index - 100, Miss
* Address 190: Index with Offset - 1110, Index - 111, Hit
* Address 14: Index with Offset - 1110, Index - 111, Miss
* Address 181: Index with Offset - 0101, Index - 010, Hit
* Address 44: Index with Offset - 1100, Index - 110, Miss
* Address 186: Index with Offset - 1010, Index - 101, Miss
* Address 253: with Offset - 1101, Index - 110, Miss

**Exercise 5.2.4**

After calculating the miss rates and average access times for the three cache designs, here are the results:

* C1 has a miss rate of 0.75 and an average access time of 20.75 cycles.
* C2 has a miss rate of 0.75 and an average access time of 21.75 cycles.
* C3 has a miss rate of 1.0 and an average access time of 30.0 cycles.

Based on these calculations, C1 is the best cache design among the three options, as it has the lowest average access time. This suggests that despite the faster access time of C3, its high miss rate results in a higher average access time when considering the miss penalty.

**Exercise 5.2.5**

For a cache with 16-word blocks, I did some calculations and found out:

* The cache is made up of 2 blocks.
* To find the right block, we use 1 bit (because there are only 2 blocks to choose from).
* The part of the address that tells us which block to look at has 27 bits (since a full address has 32 bits, and we use 4 for the word in the block and 1 for the block number).
* The cache can store 1024 bits of actual data.
* It also uses an extra 56 bits for keeping track of which blocks are in use and what data they're holding.
* So, the total size of the cache is 1080 bits.

Even though this cache can hold a lot of information at once, it's not very quick because it's constantly busy checking and swapping information in its two big sections.

**Exercise 5.2.6**

Normally, we just divide the address by the number of spots to find where to put the data.

The new idea is to mix some of the address bits up (using XOR) to pick a cache spot. But the way it mixes only gives us 32 different options, and we need 1024 options for all the spots in the cache. So, this new trick doesn't give us enough different options to use all the spots in the cache.

To make it work, we'd need to mix more bits or use another method alongside the XOR to get enough options for the 1024 spots.

**Exercise 5.3.1**

Assuming a word is 4 bytes (which is typical for a 32-bit system), the block size would be 16 bytes per block divided by 4 bytes per word which will equal 4 words per block 16 bytes per block/4 bytes per word=4 words per block.

**Exercise 5.3.2**

The index bits are used to determine which cache line (or entry) an address maps to. With 5 index bits (address bits 5 to 9), there are 2 to the power of 5 = 32 different indices possible, which means the cache has 32 entries.

**Exercise 5.3.3**

The ratio of total bits required for the cache implementation over the data storage bits is approximately 1.18. This means for every bit of data, an additional 0.18 bits are used for the tag and valid bits in the cache

**Exercise 5.3.4**

To determine this, we need to calculate the index for each address and see if that index in the cache is being used by another block. If it is, that block is replaced. Three blocks are replaced during the cache simulation.

**Exercise 5.3.5**

The hit ratio is the number of hits divided by the total number of cache accesses. A hit occurs when the data corresponding to an address is already in the cache. The hit ratio is 0.25, which means 25% of the cache accesses were hits.

**Exercise 5.3.6**

The final state of the cache, with each valid entry represented as a record of <index, tag, data>, is as follows:

* Index: 0, Tag: 4, Data (Address): 1024
* Index: 1, Tag: 4, Data (Address): 3100
* Index: 8, Tag: 4, Data (Address): 2180
* Index: 14, Tag: 4, Data (Address): 232
* Index: 10, Tag: 4, Data (Address): 160
* Index: 11, Tag: 4, Data (Address): 180

The 'data' is represented by the last address that was stored in each cache block.