Chapter 5 Exercises

5.1 In this exercise, we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 32-bit integer:

```
for (i = 0; i < 8; i++)
    for (j = 0; j < 8000; j++)
        A[i][j] = B[i][0] + A[i][j];
```

5.1.1 [5] ⟨5.1⟩ How many 32-bit integers can be stored in a 16-byte cache block?
Assuming the addresses are 32 bits in length (1 word), we can store four 32-bit integers in a 16 byte (4 word) cache.

5.1.2 [5] ⟨5.1⟩ References to which variables exhibit temporal locality?
Temporal locality refers to locality in time. If information was recently referenced, it is likely to be referenced again in the near future. In the C code above, the loop variables `i` and `j` are accessed multiple times in each iteration. Additionally, the value of `B[i][0]` does not change within the inner loop, so here the element is an example of temporal locality.

5.1.3 [5] ⟨5.1⟩ References to which variables exhibit spatial locality?
Spatial locality refers to locality in space. Information which is spatially near information recently referenced is likely to be referenced in the near future. A good example of spatial locality is iterating linearly through an array, as array elements are stored sequentially in memory. The next element to be accessed, is directly next to the last accessed element. In the C example, `A[i][j]` demonstrates usage of spacial locality. Note, `B[i][0]` does not exhibit spacial locality, as C compilers use row major order. For example, `B[0][0]` is 8000 (assuming row size of 8000) memory locations away from `B[1][0]`.

5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses:

```
3, 180, 42, 2, 191, 88, 190, 14, 181, 44, 186, 253
```

5.2.1 [10] ⟨5.3⟩ For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Address (dec)	**3**	**180**	**42**	**2**	**191**	**88**	**190**	**14**	**181**	**44**	**186**	**253**
Address (bin)	00000011	10110100	00101010	000000010	10111111	01011000	10111110	00001110	10110101	00101100	10111010	11111101
Tag (28)	0000	1011	0010	0000	1011	0101	1011	0000	1011	0010	1011	1111
Index (4)	0011	0100	1010	0010	1111	1000	1110	1110	0101	1100	1010	1101
Hit or Miss	M	M	M	M	M	M	M	M	M	M	M	M

5.2.2 [10] ⟨5.3⟩ For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with two-word blocks and a total size of 8 blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Memory (dec)	**3**	**180**	**42**	**2**	**191**	**88**	**190**	**14**	**181**	**44**	**186**	**253**
Memory (bin)	00000011	10110100	00101010	000000010	10111111	01011000	10111110	00001110	10110101	00101100	10111010	11111101
Tag (28)	0000	1011	0010	0000	1011	0101	1011	0000	1011	0010	1011	1111
Index (3)	001	010	101	001	111	100	111	111	010	110	101	110
Hit or Miss	M	M	M	H	M	M	H	M	H	M	M	M

5.2.3 [10] ⟨5.3, 5.4⟩ You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 8 words of data: C1 has 1-word blocks, C2 has 2-word blocks, C3 has 4-word blocks. In terms of miss rate, which cache design is the best? If the miss stall time is 25 cycles, and C1 has an access time of 2 cycles, C2 takes 3 cycles, and C3 takes 5 cycles, which is the best cache design?
There are many different design parameters that are important to a cache's overall performance. Below are listed parameters for different direct-mapped cache designs:
Cache Data Size: 32 KiB
Cache Block Size: 2 words
Cache Access Time: 1 cycle

C1: Miss rate 100%

Memory (dec)	**3**	**180**	**42**	**2**	**191**	**88**	**190**	**14**	**181**	**44**	**186**	**253**
Hit or Miss	M	M	M	M	M	M	M	M	M	M	M	M

C2: Miss rate 83.3%

Memory (dec)	**3**	**180**	**42**	**2**	**191**	**88**	**190**	**14**	**181**	**44**	**186**	**253**
Hit or Miss	M	M	M	M	M	M	H	M	H	M	M	M

C3: Miss rate 91.7%

Memory (dec)	**3**	**180**	**42**	**2**	**191**	**88**	**190**	**14**	**181**	**44**	**186**	**253**
Hit or Miss	M	M	M	M	M	M	H	M	M	M	M	M

(a) Exclusively comparing the miss rates of each design, C2 would be the best for this set of data with a miss rate of 83.3%. This would be followed by C2 (91.7%) then C1 (100%).

(b) Accounting for the miss stall time and the access times:

$$\text{total cycles} = (\text{number of accesses} \times \text{access time}) + \text{misses} \times \text{miss stall time}$$

$$\text{total cycles}_{\text{C1}} = (12 \times 2 \text{ cycles}) + (12 \times 25 \text{ cycles}) = 324 \text{ cycles}$$

$$\text{total cycles}_{\text{C2}} = (12 \times 3 \text{ cycles}) + (10 \times 25 \text{ cycles}) = 286 \text{ cycles}$$

$$\text{total cycles}_{\text{C3}} = (12 \times 5 \text{ cycles}) + (11 \times 25 \text{ cycles}) = 335 \text{ cycles}$$

C2 is the best cache design for this set of data.

5.7 This exercise examines the impact of different cache designs, specifically comparing associative caches to the direct-mapped caches from Section 5.4. For these exercises, refer to the address stream shown in Exercise 5.2.

5.7.1 [10] ⟨5.4⟩ Using the sequence of references from Exercise 5.2, show the final cache contents for a three-way set associative cache with two-word blocks and a total size of 24 words. Use LRU replacement. For each reference, identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss.

5.7.2 [10] ⟨5.4⟩ Using the references from Exercise 5.2, show the final cache contents for a fully associative cache with one-word blocks and a total size of 8 words. Use LRU replacement. For each reference, identify the index bits, the tag bits, and if it is a hit or a miss.

5.9 This exercise examines the single error-correcting, double error-detecting (SEC/DED) Hamming code.

5.9.1 [10] ⟨5.5⟩ What is the minimum number of parity bits required to protect a 128-bit word using the SEC/DED code?