Name: Georges Hatem

CS 550 Homework 4

Due Date: December 5, 2021

Problem 1:

Part A:

Assuming each word is a 64-bit integer as stated in the instructions:

First, let's convert byte to bits:

$$1\ byte = 8\ bits$$

So,

$$16\ bytes = (16) * (8) = 128\ bits$$

So, to find out how many 64-bit integers can be stored in a 16-byte cache block, we need to divide 128 bits by 64 bits:

$$\frac{128\ bits}{64\ bits} = 2$$

So, we can store two (2) 64-bit integers in a 16-byte cache block.

Part B:

Temporal locality is defined as follows:

The locality principle stating that if a data location is referenced, then it will tend to be referenced again soon.

In the code provided in Problem 1, we can see the following:

1) J keeps getting called everytime we are accessing A[j][i] and C[i][j] when looping and when it gets incremented by 1 when accessing the outer loop.
2) I keeps getting called everytime we are incrementing its value by 1 in the inner

loop, and everytime we are accessing A[j][i], B[i][0], and C[i][j] when looping.

We are not accessing the same row and index values of arrays A, B, and C right away.

For array A:

We go as follows:

1st Iteration: A[0][0], A[0][1], A[0][2],….., A[0][299]
2nd Iteration: A[1][0], A[1][1], A[1][2],….., A[1][299]
.
.
.
.
Last Iteration: A[99][0], A[99][1], A[99][2],……, A[99][299]

As you can see, we are not calling the same row and index value of array A when looping. We are looping through array A. So, array A does not exhibit temporal locality.

For array B:

We go as follows:

1st Iteration	2nd Iteration ……	Last Iteration
B[0][0]	B[0][0]	B[0][0]
B[1][0]	B[1][0]	B[1][0]
B[2][0]	B[2][0]	B[2][0]
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
B[299][0]	B[299][0]	B[299][0]

As you can see, we are not calling the same row and index value of array B when looping. We are looping through array B. So, array B does not exhibit temporal locality.

For array C:

We go as follows:

1st Iteration	2nd Iteration …….	Last Iteration
C[0][0]	C[0][1]	C[0][99]
C[1][0]	C[1][1]	C[1][99]
C[2][0]	C[2][1]	C[2][99]
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
C[299][0]	C[299][1]	C[299][99]

As you can see, we are not calling the same row and index value of array C when looping. We are looping through array C. So, array C does not exhibit temporal locality.

From my analysis above, i and j are the variable references in the code given in Problem 1 that exhibit temporal locality.

Part C:

Spatial locality is defined as follows:

The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

In the code provided in Problem 1 above, we can see the following:

i and j do not exhibit spatial locality:

i keeps getting called everytime we are incrementing its value by 1 in the inner loop, and everytime we are accessing A[j][i], B[i][0], and C[i][j] when looping.

And j keeps getting called everytime we are accessing A[j][i] and C[i][j] when looping and when it gets incremented by 1 when accessing the outer loop.

So, i and j do not fit the definition of spatial locality, and therefore, they do not exhibit spatial locality.

For array A:

We go as follows:

1st Iteration: A[0][0], A[0][1], A[0][2],….., A[0][299]
2nd Iteration: A[1][0], A[1][1], A[1][2],….., A[1][299]
.
.
.
.
Last Iteration: A[99][0], A[99][1], A[99][2],……, A[99][299]

As you can see, the accesses A[0][0], A[0][1],……., A[0][299]; A[1][0], A[1][1],……, A[1][299]; A[99][0], A[99][1],……, A[99][299] appear sequentially in memory. Therefore, A[j][i] exhibits spatial locality.

For array B:

We go as follows:

1st Iteration	2nd Iteration ……	Last Iteration
B[0][0]	B[0][0]	B[0][0]
B[1][0]	B[1][0]	B[1][0]
B[2][0]	B[2][0]	B[2][0]
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
B[299][0]	B[299][0]	B[299][0]

As you can see, the accesses B[0][0], B[1][0],…., B[299][0] do not appear sequentially in memory. Therefore, B[i][0] does not exhibit spatial locality.

For array C:

We go as follows:

1st Iteration	2nd Iteration …….	Last Iteration
C[0][0]	C[0][1]	C[0][99]
C[1][0]	C[1][1]	C[1][99]
C[2][0]	C[2][1]	C[2][99]
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
C[299][0]	C[299][1]	C[299][99]

As you can see, the accesses C[0][0], C[1][0],….., C[299][0]; C[0][1], C[1][1],….., C[299][1];

C[0][99], C[1][99],….., C[299][99] do not appear sequentially in memory. Therefore, C[i][j] does not exhibit spatial locality.

From my analysis above, I can deduce that the variable references that exhibit spatial locality in the code in Problem 1 is A[j][i].

Problem 2:

Let's get the binary word address, tag, and the index for each of the references given in Problem 2 given a direct mapped cache with 16 one-word blocks:

0x43:

Transforming 0x43 to binary, we get the following:

01000011

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 01000011 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0011 and the tag would be 0100.

Therefore, the binary word address for 0x43 is 01000011, the tag for 0x43 is 0100, and the index for 0x43 is 0011.

<u>0xc4:</u>

Transforming 0xc4 to binary, we get the following:

11000100

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 11000100 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0100 and the tag would be 1100.

Therefore, the binary word address for 0xc4 is 11000100, the tag for 0xc4 is 1100, and the index for 0xc4 is 0100.

0x2b:

Transforming 0x2b to binary, we get the following:

00101011

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 00101011 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1011 and the tag would be 0010.

Therefore, the binary word address for 0x2b is 00101011, the tag for 0x2b is 0010, and the index for 0x2b is 1011.

<u>0x42:</u>

Transforming 0x42 to binary, we get the following:

01000010

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 01000010 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0010 and the tag would be 0100.

Therefore, the binary word address for 0x42 is 01000010, the tag for 0x42 is 0100, and the index for 0x42 is 0010.

<u>0xc5:</u>

Transforming 0xc5 to binary, we get the following:

11000101

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 11000101 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0101 and the tag would be 1100.

Therefore, the binary word address for 0xc5 is 11000101, the tag for 0xc5 is 1100, and the index for 0xc5 is 0101.

<u>0x28:</u>

Transforming 0x28 to binary, we get the following:

00101000

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 00101000 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1000 and the tag would be 0010.

Therefore, the binary word address for 0x28 is 00101000, the tag for 0x28 is 0010, and the index for 0x28 is 1000.

<u>0xbe:</u>

Transforming 0xbe to binary, we get the following:

10111110

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 10111110 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1110 and the tag would be 1011.

Therefore, the binary word address for 0xbe is 10111110, the tag for 0xbe is 1011, and the index for 0xbe is 1110.

<u>0x05:</u>

Transforming 0x05 to binary, we get the following:

00000101

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 00000101 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0101 and the tag would be 0000.

Therefore, the binary word address for 0x05 is 00000101, the tag for 0x05 is 0000, and the index for 0x05 is 0101.

<u>0x92:</u>

Transforming 0x92 to binary, we get the following:

10010010

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 10010010 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 0010 and the tag would be 1001.

Therefore, the binary word address for 0x92 is 10010010, the tag for 0x92 is 1001, and the index for 0x92 is 0010.

<u>0x2a:</u>

Transforming 0x2a to binary, we get the following:

00101010

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 00101010 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1010 and the tag would be 0010.

Therefore, the binary word address for 0x2a is 00101010, the tag for 0x2a is 0010, and the index for 0x2a is 1010.

<u>0xba:</u>

Transforming 0xba to binary, we get the following:

10111010

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 10111010 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1010 and the tag would be 1011.

Therefore, the binary word address for 0xba is 10111010, the tag for 0xba is 1011, and the index for 0xba is 1010.

<u>0xbd:</u>

Transforming 0xbd to binary, we get the following:

10111101

Considering a direct mapped cache with 16 one-word blocks, the index will be the first 4 bits of the binary word address 10111101 because 2^4 is equal to 16 so we take the first 4 bits of the binary word address as the index and the last remaining bits of the binary word address as the tag.

So, the index would be 1101 and the tag would be 1011.

Therefore, the binary word address for 0xbd is 10111101, the tag for 0xbd is 1011, and the index for 0xbd is 1101.

I illustrated the table below for miss and hit:

Address (Hex)	Address (Binary)	Tag	Index	Hit/Miss
0x43	01000011	0100	0011	Miss
0xc4	11000100	1100	0100	Miss
0x2b	00101011	0010	1011	Miss
0x42	01000010	0100	0010	Miss
0xc5	11000101	1100	0101	Miss
0x28	00101000	0010	1000	Miss
0xbe	10111110	1011	1110	Miss
0x05	00000101	0000	0101	Miss
0x92	10010010	1001	0010	Miss
0x2a	00101010	0010	1010	Miss
0xba	10111010	1011	1010	Miss
0xbd	10111101	1011	1101	Miss

Part B:

I illustrated below the table representing the binary word address, the tag, the index, the offset, and the hit/miss given a direct-mapped cache with two-word blocks and a total size of eight blocks:

Address (Hex)	Address (Binary)	Tag	Index	Offset	Hit/Miss
0x43	01000011	0100	001	1	Miss
0xc4	11000100	1100	010	0	Miss
0x2b	00101011	0010	101	1	Miss
0x42	01000010	0100	001	0	Hit
0xc5	11000101	1100	010	1	Hit
0x28	00101000	0010	100	0	Miss
0xbe	10111110	1011	111	0	Miss
0x05	00000101	0000	010	1	Miss
0x92	10010010	1001	001	0	Miss
0x2a	00101010	0010	101	0	Hit
0xba	10111010	1011	101	0	Miss
0xbd	10111101	1011	110	1	Miss

Part C:

For C1:

C1 has 1-word blocks. So, number of offset is as follows:

Number of offset = 0 since 2^0 = 1

So, for C1 we obtain the table below:

Address (Hex)	Address (Binary)	Tag	Index	Hit/Miss
0x43	01000011	0100	0011	Miss
0xc4	11000100	1100	0100	Miss
0x2b	00101011	0010	1011	Miss
0x42	01000010	0100	0010	Miss
0xc5	11000101	1100	0101	Miss
0x28	00101000	0010	1000	Miss
0xbe	10111110	1011	1110	Miss
0x05	00000101	0000	0101	Miss
0x92	10010010	1001	0010	Miss
0x2a	00101010	0010	1010	Miss
0xba	10111010	1011	1010	Miss
0xbd	10111101	1011	1101	Miss

As you can, see for C1 miss rate is 100 percent.

For C2:

C2 has 2-word blocks. So, number of offset is as follows:

Number of offset = 1 since 2^1 = 2

So, for C2, we obtain the table below

Address (Hex)	Address (Binary)	Tag	Index	Offset	Hit/Miss
0x43	01000011	0100	001	1	Miss
0xc4	11000100	1100	010	0	Miss
0x2b	00101011	0010	101	1	Miss
0x42	01000010	0100	001	0	Hit
0xc5	11000101	1100	010	1	Hit
0x28	00101000	0010	100	0	Miss
0xbe	10111110	1011	111	0	Miss
0x05	00000101	0000	010	1	Miss
0x92	10010010	1001	001	0	Miss
0x2a	00101010	0010	101	0	Hit
0xba	10111010	1011	101	0	Miss
0xbd	10111101	1011	110	1	Miss

As you can see from the table above, miss rate for C2 is as follows:

Miss Rate of C2 = (9/12)*(100) = 75 percent

For C3:

C3 has 4-word blocks. So, number of offset for C3 is as follows:

Number of Offset = 2 since 2^2 = 4

So, for C3 we obtain the table below:

Address (Hex)	Address (Binary)	Tag	Index	Offset	Hit/Miss
0x43	01000011	0100	00	11	Miss
0xc4	11000100	1100	01	00	Miss
0x2b	00101011	0010	10	11	Miss
0x42	01000010	0100	00	10	Hit
0xc5	11000101	1100	01	01	Hit
0x28	00101000	0010	10	00	Hit
0xbe	10111110	1011	11	10	Miss
0x05	00000101	0000	01	01	Miss
0x92	10010010	1001	00	10	Miss
0x2a	00101010	0010	10	10	Hit
0xba	10111010	1011	10	10	Miss
0xbd	10111101	1011	11	01	Hit

As you can see from the table above, the miss ratio is as follows:

Miss Ratio for C3 = (7/12)*(100) = 58.33 percent

So, from my analysis above, C3 is better for the above sequence of accesses since it has the lowest miss rate and highest hit rate.

Problem 3:

Part A:

Let's calculate AMAT (Average Memory Access Time)

AMAT = (Time for a hit) + (Miss Rate)*(Miss Penalty)

Since CPI is given as 1, then for a hit the time is 1 cycle

For Block Size 8:

AMAT = (1) + (0.05)*(30*8) = 13 Cycles

For Block Size 16:

AMAT = (1) + (0.03)*(30*16) = 15.4 Cycles

For Block Size 32:

AMAT = (1) + (0.02)*(30*32) = 20.2 Cycles

For Block Size 64:

AMAT = (1) + (0.015)*(30*64) = 29.8 Cycles

For Block Size 128:

AMAT = (1) + (0.011)*(30*128) = 43.24 Cycles

A Block Size of 8 has the lowest AMAT. So, the optimal block size for a miss penalty of 30*B Cycles is Block Size 8.

Part B:

If miss latency was constant (independent of B), then the optimal block size would be the block size with the lowest miss rate, which is in this case Block Size 128 with a miss rate of 1.1 percent.

So, if miss latency was constant (independent of B), then the optimal block size would be Block Size 128.

Problem 4:

Memory address is of 64 bit, so size of word is as follows:

$$\frac{64}{8} = 8\ Byte$$

Part A:

size of offset = 3 bits (as block size is 1 word)
Size of index = 9 bits (as number of blocks =512)

Size of tag = 64 - 12 = 52 bits

Part B:

size of offset = 8*8 Byte = 2^6 = 6 bits
size of index = 64 blocks = 2^6=6 bits

Size of tag = 64-12= 52 bits

Part C:

ratio in question a

3:64

ratio in question b

6:64

Part D:

size of block = 1 word = 8 bytes

number of blocks = size of cache / size of word

So,

number of blocks = 512 word / 1 word

number of blocks = 512 blocks

In a two-way set associative cache, 2 blocks will form 1 set.

So, number of index = 512/2=256=2^8

Size of offset = 3 bits

size of index = 8 bits

Size of tag = 64-11=53 bits