Problem1

Problem 1.

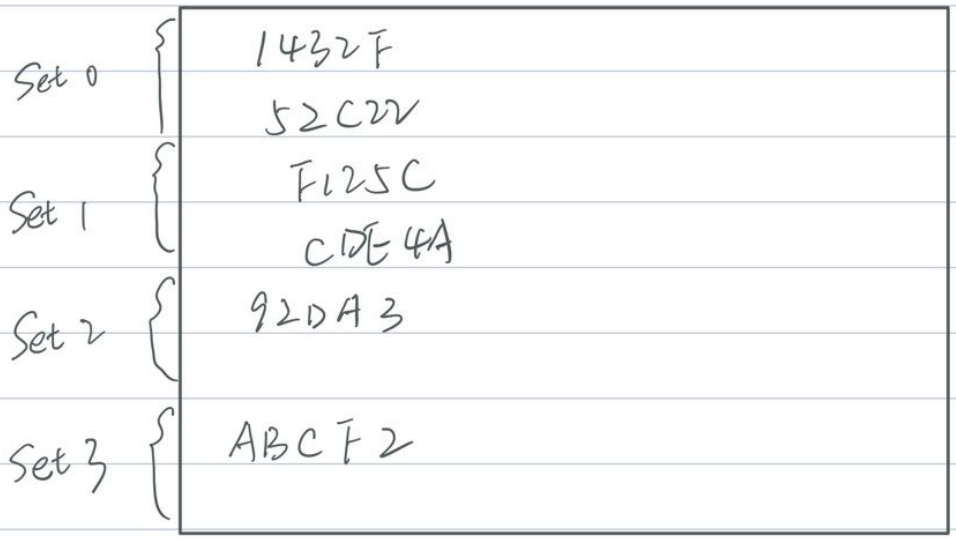
As every cache block is 64B and total size is 512 bytes and it is a 2 way set.

So. there are $\frac{512}{64 \times 2} = 4$ set of Cache.

Every cache set has 2 blocks.

Transform the address from hex to binary.

HEX	Binary	Miss or Hit
ABCDE	1010 1011 1100 11 01 1110	miss
14327	0001 0100 0011 00 10 0111	miss
D7148	1101 0111 0001 01 00 1000	miss
8F220	1000 1111 0010 00 10 0000	miss
CDE4A	1100 1101 1110 01 00 1010	miss
14327	0001 0100 0011 00 10 0111	hit
52C22	0101 0010 1100 00 10 0010	miss
ABCF2	1010 1011 1100 11 11 0010	hit
920A3	1001 0010 1010 00 10 0011	miss
F125C	1111 0001 0010 01 01 1100	miss



the Address above represents the corresponding block.

Problem2

Problem2:

(a).

AAT = 1 + (15 + 30% × 300) × 3%

= 4.15 (CPU cycles)

(b)

AAT = 1 + (15 + 5% × 300) × 10%

= 4 (CPU cycles)

Problem 3

(C)

Following the instruction in Q3, I find the L1 Cache Size = 32k. So I imitate the cache_test program in class using an uint_64 array with 4096 elements, which is exactly the size of L1 Cache.

For write only, I initialize the array with 0 and write the 1 into the array.

```
index = num_traversals;
  clock_gettime(CLOCK_MONOTONIC, &start_time);
  for(;index > 0 ; --index){
    for (i=0; i < num_elements; i++) {
      array[i] = 1;
    }
  }

  clock_gettime(CLOCK_MONOTONIC, &end_time);
```

My Program needs one parameter for traversal time. Like this: ./cache_test_write1 200

To compute the bandwidth:

```
double elapsed_ns = calc_time(start_time, end_time);
  printf("Time = %f\n", elapsed_ns);
  printf("Bandwidth for write = %f GB/s \n ", (((uint64_t)num_elements * (uint64_
t)num_traversals * 8) / (elapsed_ns)));
```

Result:

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_write1 10000000

Time = 7451535155.000000

Bandwidth for write = 43.974831 GB/s

For write 1 read 1, I add one statement of read based on the write only.

```
clock_gettime(CLOCK_MONOTONIC, &start_time);
  int read = 0;
  index = num_traversals;
  for(;index > 0 ; --index){
    for (i=0; i < num_elements; ++i) {
      read = array[i];
      array[i] = 1;
    }
  }

  clock_gettime(CLOCK_MONOTONIC, &end_time);
```

My program needs one element as traversal parameter. Like: ./cache_test_read1_write1 10000000

Result:

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_read1_write1 10000000

Time = 7328985697.000000

Bandwidth for write = 89.420286 GB/s

To Compute:

```
double elapsed_ns = calc_time(start_time, end_time);
  printf("Time = %f\n", elapsed_ns);
  printf("Bandwidth for write = %f GB/s \n ", (((uint64_t)num_elements * (uint64_
t)num_traversals * 8*2) / (elapsed_ns)));
```

For write 1 read 2, I add one statement of read based on the write1 read1.

```
int temp;
  index = num_traversals;
  clock_gettime(CLOCK_MONOTONIC, &start_time);
  for(;index > 0 ; --index){
    for (i=0; i < num_elements; i++) {
      temp = array[i];
      array[i] = 1;
      temp = array[i];
    }
  }

  clock_gettime(CLOCK_MONOTONIC, &end_time);
```

My program needs one parameter as traversal times. Like: ./cache_test_read2_write1 10000000

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_read2_write1 10000000

Time = 7442069100.000000

Bandwidth for write = 132.092297 GB/s

To Compute:

```
double elapsed_ns = calc_time(start_time, end_time);
  printf("Time = %f\n", elapsed_ns);
```

```
  printf("Bandwidth for write = %f GB/s \n ", (((uint64_t)num_elements * (uint64_
t)num_traversals * 8* 3) / (elapsed_ns)));
```

Write	43.974831 GB/s
Write1 read1	89.420286 GB/s
Write1 read2	132.092297 GB/s

Based on the table, we can come a conclusion that read is faster than write. As read will use less instructions.

(d)

L3 Cache is 22528K, so I will use array with more than 22528k / 8 = 2816k elements. I will use 2820k elements.

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_write1 1000

Time = 1562594937.000000

Bandwidth for write = 14.437523 GB/s

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_read1_write1 1000

Time = 1591084697.000000

Bandwidth for write = 28.358013 GB/s

kc426@vcm-21370:~/ECE565/hw2/problem3$ ./cache_test_read2_write1 1000

Time = 1560566068.000000

Bandwidth for write = 43.368878 GB/s

Write	14.437523 GB/s
Write1 read1	28.358013 GB/s
Write1 read2	43.368878 GB/s

I believe this arising from that cpu must wait for the data so the total latency will be large. As a result, the Bandwidth is smaller than the above. This match my expectation.

Problem 4

(b)

ijk	ijk_Time = 1.757477s
jki	jki_Time = 18.576692s
ikj	ikj_Time = 0.559666s

Runtime Result:

kc426@vcm-21370:~/ECE565/hw2/problem4$ ./matrix_test ijk

ijk_Time = 1.757477

kc426@vcm-21370:~/ECE565/hw2/problem4$ ./matrix_test ikj

ikj_Time = 0.559666

kc426@vcm-21370:~/ECE565/hw2/problem4$ ./matrix_test jki

jki_Time = 18.576692

As C++ store the data in row-major order, so the First one in B matrix will experience some cache miss. And the jki will experience most cache miss. And the ikj will experience the least cache miss. So T(ikj) < T(ijk) < T(jki)

(d)

In my VM, the L2 Cache is 1M which is 1024kB,

That is:

blockSize * blockSize * 3 * 8 < 1024K

Then I choose Blocksize = 128 and block = 64 as experiment parameter.

The block = 64 get the best result.

Ijk	ijk_Time = 1.769311
ijk	ijk_Time = 1.769311

Runtime Result:

kc426@vcm-21370:~/ECE565/hw2/problem4$ ./matrix_test ijk_tiling

ijk_tiling_Time = 1.643415

kc426@vcm-21370:~/ECE565/hw2/problem4$ ./matrix_test ijk

ijk_Time = 1.769311

It is faster than ijk and jki but slower than ikj. I believe this arising from that L1 Cache is faster than L2 Cache. So the ijk tiling will be faster than ijk but slower than ikj.