Homework 8 Key (New Version)

8 pts to be added automatically if you submit your HW8, which makes 20pts in total. (Make sure you answered all questions Q1 – Q3. A non-answered HW8 does not count 8pts.)

Q1. (8 pts) Cache and Memory mapping
Suppose a byte-addressable memory has a 2M-byte capacity and CPU cache consists of 64 blocks, where each block contains 32 bytes.
Q1-1. Direct Mapping
Q1-1-i. Divide the bits in main memory into tag, block and offset bits. (2pts)

Memory size: 2^21
Cache Size: 64= 2^6
Cache Block Size: 32= 2^5

10 = 21-6-5

Tag	Block	Offset
10 bits	6 bits	5 bits

Q1-1-ii. What is the tag, line and offset for the address 0x123A63, in hexadecimal? (2pts)
0001 0010 0011 1010 0110 0011
Tag (12 bits) : **0010 0100 0111**
Offset (3 bits) : **0 0011**
Block (6 bits): **01 0011**

Tag	Block	Offset
0x 247	0x 13	0x3

Q1-2. 4-way set associative mapping
Q1-2-i. Divide the bits in main memory into tag, set and offset bits (2pts)

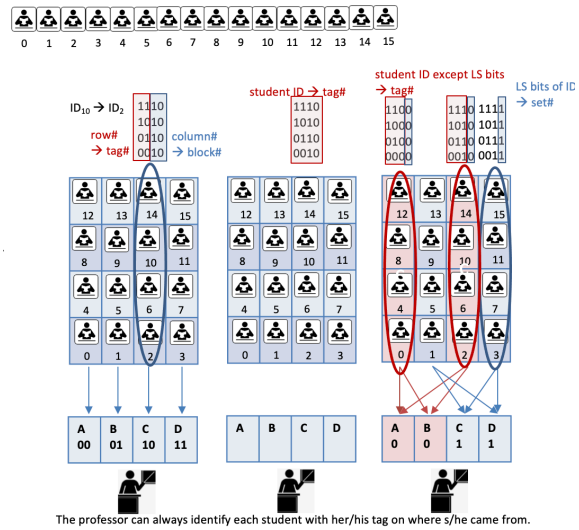
Memory size: 2M = 2^21
Cache Size: 64 /4=16 16=2^4
Cache Block Size: 32= 2^5

Tag	Set	Offset
12 bits	4 bits	5 bits

Q1-2-ii. What is the tag, set and offset for the address 0x123A63, in hexadecimal? (2pts)

Tag	Set	Offset
0x 91D	0x 3	0x 3

Q2. (2pts) Cache and Programming: direct-mapping and set associative mapping
Look at 15.Cache-Direct.pptx's p13.



Each student represents an integer space, (i.e., 4 bytes). Therefore, each student's address will be:
student[0] at 0000 00
student[1] at 0001 00
student[2] at 0010 00
…
student[14] at 1110 00
student[15] at 1111 00

In direct mapping on this slide p13, the address is partitioned

Tag	**Block**	**Offset**
2 bits	2 bits	2 bits

In two-way set associative mapping on this slide p13, the address is partitioned

Tag	**Set**	**Offset**
3 bits	1 bits	2 bits

Given the following three loops, say loop A, loop B, and loop C, which loop can two-way set associative mapping benefit? This means that the other two loops observe no performance difference between direct mapping and two-way set associative mapping.

```
// Loop A
for ( int i = 0; i < 10; i++ ) {
  sum += student[0]++ + student[2]++ + student[4]++ student[6]++;
 }

// Loop B
for ( int i = 0; i < 10; i++ ) {
  sum += student[0]++ + student[2]++;
 }

// Loop C
for ( int i = 0; i < 10; i++ ) {
  sum += student[0]++ + student[4]++;
 }
```

Your answer: Loop C as it benefits associative mapping compared to loop A and B.

Q3. (2pts) Cache and Programming: write invalidate and write update
Let's assume that two CPUs, (say CPUs A and B) execute the following two loops (i.e., loop A and B), respectively. Also assume that "int array[N]" is shared among these two CPUs and zero-initialized by CPU A before CPU B gets started.

```
int array[N];
bzero( array, N * 4 );

// Loop A to be executed by CPU A
for ( int i = 0; i < N; i++ )
  array[i] = i + 1;

// Loop B to be executed by CPU B
for ( int i = 0; i < N; i++ ) {
  if ( array[i] == 0 );
  sum += array[i];
 }
```

Do not worry about any CPU synchronization (which you'll study about in CSS430 OS). Just focus on cache memory. Which policy of "write invalidation" or "write update" would work better or faster?

Your answer: Write update will work more faster, improving both efficiency as well as reduces the cache number of misses.