Dan Shawlson
CSE 5441, T/Th 2:20 P.M.

Homework 1

1. (a) C1:

$C : 64$	$B : 4$	$E : 1$	$S : 16$
$m : 16$	$b : 2$	$t : 10$	$s : 4$

C2:

$C : 64$	$B : 16$	$E : 1$	$S : 4$
$m : 16$	$b : 4$	$t : 10$	$s : 2$

(b)

	BA00	BA04	AA08	BA05	AA14	AA11	AA13	AA38
C1	miss	miss	miss	hit	miss	miss	hit	miss
C2	miss	hit	miss	miss	miss	hit	hit	miss

	AA09	AA0B	BA04	AA2B	BA05	BA06	AA09	AA11
C1	hit	hit	hit	miss	hit	hit	hit	hit
C2	miss	hit	miss	miss	hit	hit	miss	hit

Final content:
C1:

Set 0 : BA00-BA03	Set 1 : BA04-BA07	Set 2 : AA08-AA0B	Set 3 : ?
Set 4 : AA10-AA13	Set 5 : AA14-AA17	Set 6 : ?	Set 7 : ?
Set 8 : ?	Set 9 : ?	Set 10 : AA28-AA2B	Set 11 : ?
Set 12 : ?	Set 13 : ?	Set 14 : AA38-AA3B	Set 15 : ?

C2:

Set 0 : AA00-AA0F	Set 1 : AA10-AA1F	Set 2 : AA20-AA2F	Set 3 : AA30-AA3F

(c)

	0x0004	0xF008	0x0005	0xF009
C1	miss	miss	hit	hit
C2	miss	miss	miss	miss

2. (a) 2^{34} bytes (0x000000000-0x3FFFFFFFF)
   (b) 4096 bytes
   (c) Implementing our cache will require 8 bits$*$(4096 data bytes)+1 valid bit$*$(256 sets)+22 tag bits$*$(256 sets), for a total of 38656 bits.
   (d) 2^{22} blocks

3. If we use the following cache parameters

$C : 8192$	$B : 32$	$E : 1$	$S : 256$
$m : 64$	$b : 5$	$t : 51$	$s : 8$

Then this code will have a cache hit ratio of at least 87.5%

```
register int min = MAX_INT;
for (int i = 0; i < 1000000; i++) {
  if (array[i] < min) {
    min = array[i];
  }
}
```

4.

```
// Given an x by y matrix

// Good locality of reference
int sum = 0;
for (int i = 0; i < x; i++) {
  for (int j = 0; j < y; j++) {
    sum += matrix[i][j];
  }
}
int average = sum / (x * y);

// Bad locality of reference
int sum = 0;
for (int j = 0; j < y; j++) {
  for (int i = 0; i < x; i++) {
    sum += matrix[i][j];
  }
}
int average = sum / (x * y);
```

Because C compilers store 2-D arrays as row-major, elements in the same row of a matrix are stored contigously in memory. Hence, whenever we try to access a member of a row and have a cache miss, we bring into cache other elements from the row after the initial member we requested. So in the "Good locality of reference" example above, by reading all the elements in a row before moving to the next row, we will have mostly cache hits. However, in the example of bad locality of reference, we read all elements in a column before moving to the next column. Unless our cache lines are longer than the rows of the matrix, we will likely have a cache miss every access.

In Fortran we would see the opposite effect, since it stores 2-D arrays as column-major.

5.
   - Set 0: valueA[2046-2047], followed by 8 unknown bytes
     Set 1: valueA[1026-1029]
     Set 2: valueA[1030-1033]
     Set 3: valueA[1034-1037]
     . . .
     Set 255: valueA[2042-2045]
   - (1 memory access per cache load)(256 + 256 + 1 sets loaded) = 513 memory accesses