Homework 3

Memory and Cache

1. Reorder the fields in this structure so that the structure will (a) consume the most space and (b) consume the least space on an IA32 machine on Linux.

```
struct foo {
  char *a;
  short *b;
  char c;
  unsigned d;
  char e
  double f;
  short g;
}
```

2. The IA32 architecture has a notion of an *instruction prefix*, which we have not yet used or discussed. One commonly used prefix is "rep". Rep repeats the instruction a fixed number of times, where the number of times is given in the %ecx, which is called the "count register" for this purpose. Rep is typically combined with a *string instruction*. String instructions operate with memory operands pointed at by the %esi (source) and %edi (destination) registers. For example:

```
movl $256, %ecx
movl A, %esi
movl B, %edi
rep movsb
```

will copy 256 byes starting at address A to addresses starting at address B. "movsb" means "Move String of Bytes". What the rep line does is essentially this pseudo code:

```
while (%ecx>0) {                // rep
   %ecx--;                      // rep
   Mem[%edi] = Mem[%esi] ;      // movsb
   %esi++;                      // movsb
   %edi++;                      // movsb
}
```

(a) Give an example (in C) of a loop that could make use of the rep prefix.

(b) Why is the rep prefix potentially helpful in improving memory system performance?

3. Consider a processor which uses 16 bit addresses and can address 2^16=64K bytes of memory. Suppose that it has one level of cache. As in Figure 6.27 of your textbook, the address is split into a t bit tag, an s bit set index, and a b bit block offset. The cache consists of 1024 bytes, with a block size of 64 bytes. Answer each of the following for direct-mapped, 4-way set associative, and fully associative versions of the cache.
   a. How many cache lines are there?
   b. What is b?
   c. What is s?
   d. What is t?

4. For the cache in problem 3, draw the cache given it is structured as follows. You can elide replicated components, but annotate your drawing with how many components there are.
   a. Direct-mapped
   b. 4-way set associative
   c. Fully associative

5. Our company wants to optimize the performance of the following code

```
void vector_add(int n, int *a, int *b, int *c) {
   int i;
   for (i=0;i<n;i++) {
     c[i]=a[i]+b[i];
   }
}
```

   run on the same processor and cache as described in problem 3. The cache is write-back, write-allocate, and has an LRU replacement policy. Integers are 32 bits.
   a. Suppose the cache is direct mapped. Let n=2048, a=0x4000, b=0x8000, c=0xc000. On average, how many times per loop iteration will you load a cache block from main memory? How many times per loop iteration will you flush a cache block back to main memory?
   b. What is the minimum degree of associativity (i.e., the n in n-way) that the cache needs to reduce the answers in (a) to 0.25 cache blocks read per iteration and 0.125 cache blocks written per iteration?
   c. While we're all fired up to buy ultra-cool mega-associative cache hardware (which comes only in machined aluminum), a smart alec programmer claims that we can get the same effect by having a=0x4000, b=0x8020, and c=0xc040. Is he right? Why or why not?