CSC 462 Homework #7 answer key

1. Miss penalty is largely controlled by the speed of DRAM. Cache improvements will not impact DRAM speed. There are things we could do with the cache to limit the DRAM access time's impact such as early restart/critical word first. Improving DRAM speed itself requires improving that technology, which differs greatly from cache technology. Improving miss rate was a common goal for cache architects but today miss rates are 10% for a 4KB cache all the way down to under 1% for a 512KB cache. Improving miss rate below 1% is not practical without continuing to increase the cache size.

2.
   a. Larger block sizes improve hit rate, specifically compulsory misses because more will have been loaded into the cache. However, the larger the block size, the fewer blocks a cache will have and this can impact conflict miss rates in a negative way. Additionally, larger block sizes take more time to load so the miss penalty will increase unless you are using a non-blocking cache and early-restart/critical word first.
   b. Increased associativity will improve hit rate, particularly conflict hit rate because there will be fewer items discarded because of conflicts. The downside is that associativity requires additional hardware which causes the cache to respond slower and that will impact hit time. If the increased associativity takes place on an L2 (or L3) cache, this will impact miss penalty instead of hit time.
   c. Adding the L3 cache improves (reduces) miss penalty. It has no negative impact aside additional cost.

3.
   a. As the memory access stages of a pipeline are the slowest, by pipelining the cache access, we can increase the clock rate. For instance, imagine in our original 5-stage pipeline, the IF and MEM stages took 1 ns and the ID, EX and WB stages took .5 ns. If we pipeline the IF and MEM stages, we can reduce the clock cycle time from 1 ns to .5 ns (approximately). A faster clock rate will improve CPU time as we saw with the CPU execution time formula (CPI * IC * Clock Cycle Time).
   b. The first stage is the cache access, the second stage is the tag check (although for a data write, the access cannot start until the tag check is performed).
   c. In a superscalar, we might have 2 memory accesses in one cycle. If these two accesses were made to a non-banked cache, we would have to serialize them and so postpone the second instruction one cycle. If the two accesses can be made to two separate, banked caches, they can occur simultaneously.

4. Address translation is the process of converting a virtual address to a physical address. This is required for virtual memory usage as the address generated by the CPU is not the physical location of the item being accessed. A page table is used to map a page number to a frame number. Address translation then requires page table access, which is stored in main memory. To avoid a main memory access every time we have a cache access, we use a small cache (TLB) to store the most recently accessed page to frame translations. But even so, any memory access now requires at least two cache accesses, the TLB and the L1 cache. In question 3, we talked about reducing the clock cycle time by pipelining cache accesses but now because of

address translation, we would have to double the clock cycle time to accommodate two accesses in a cycle. We would prefer to avoid address translation. The very reason we need address translation is because the address the CPU generates does not equal the address that the item is stored in in main memory because of virtual memory. However, we do not have to store items in caches using their main memory addresses. So, to avoid address translation, we store items in the L1 cache (and possibly the L2 cache) using their virtual address. Now, we no longer need to consult the TLB unless we have a cache miss. Then we consult the TLB and then use the physical address for main memory (or L2 or L3). Because we have an L1 cache miss, we accrue a miss penalty, so the added time to consult the TLB is not a major impact. If however there is a TLB miss and address translation requires consulting the page table in main memory, then the impact is more significant.

5. Note that the L1 data cache stores 1024 total words (int values) in 256 blocks. Any miss brings in 4 new int values. The array consists of 2048 int values.
    a. With no values initially being in the cache, and each block storing 4 int values, we will have a total of 2048 / 4 = 512 misses per outer loop iteration. As we cannot store the entire array in the cache, each successive outer loop iteration results in the same number of misses. So we wind up with 512 * 5 = 2560 total misses.
    b. If the cache could store the entire array, we would only have the initial 512 misses. But the array is twice the size of the cache, and being a direct-mapped cache, we have to discard every element of the array that we have loaded into the cache to make room for a later array reference. So there's nothing we can do about the initial 512 misses. However, the inner outer loop is essentially repeating the inner loop with a different value of j to be used in adding to a[i]. That is, the outer loop's only purpose is changing a value we are adding to each a[i]. We can improve on the code in one of two ways, first we could simply change the order of the loops (we also have to reset c to 1 each time through the outer loop), or we could remove the outer loop entirely and use a more elaborate assignment statement. Both versions are shown below.

```
for(i=0;i<2048;i++)   {
        c=1;                    // needed because the inner loop is doing c++
        for(j=2;j<=10;j+=2)  {
                a[i]=a[i]+j*c;
                c++;
        }
}
for(i=0;i<2048;i++)
        a[i]=a[i]+2*c+4*(c+1)+6*(c+2)+8*(c+3)+10*(c+4);   // or a[i]=a[i]+110;
```

    c. Both revisions of the code result in the array being brought into the cache only once instead of 5 complete times. We wind up with 512 misses in all.

6. A blocking cache is one that, when there is a cache miss, does not permit another access until the cache miss is handled. As this takes time because the cache is waiting for the next level of the memory hierarchy to respond, it would postpone other cache accesses. A non-blocking cache is one that can accommodate later cache accesses in spite of waiting to handle a cache miss. This could be useful in cases where, for instance, we had a Tomasulo-style architecture and multiple loads/stores are taking place within a few cycles, or we are running multiple

threads and an instruction miss that results in a cache miss does not postpone the processor's ability to start fetching instructions of another thread. Why postpone a later load or store or instruction fetch because of an earlier miss? The non-blocking cache supports critical word first and early restart as well as compiler-controlled prefetching. There are no downsides to a non-blocking cache other than expense, however, to continue not blocking upon further misses becomes more challenging so most non-blocking caches start blocking if there is a second miss. For instance, a datum is sought and there is a cache miss. While the non-blocking cache is handling that miss, other requests come in. As long as those accesses are all hits, the non-blocking cache continues to work fine. But if there is a second miss while handling the first miss, the cache either has to start keeping track of which memory reference is for which miss (so that upon a response coming in from lower in the hierarchy, the cache knows which miss this corresponds to) or starts blocking at this point.

7. First, I would select the two 0-cost items: larger block sizes and compiler techniques. I would select a block size of 64 bytes (8 or 16 words depending on the word size) because this is optimal for caches larger than 4 KB. The compiler techniques would include loop interchange and blocking at a minimum even though these are hard to implement (but because I'm an architect and not a compiler writer, I don't have to do it myself!) In terms of the items that "cost", I would start with a non-blocking cache and early restart/critical word first. Even though this pair of enhancements costs me 5 of my 9 units, they are critical in reducing miss penalty and supporting a superscalar architecture. I would also select multibanked caches to similarly support a superscalar. Next, I would select pipelined access so that I could increase my clock rate and thus reduce hit time. I would use multilevel caches as this will impact both miss rate and miss penalty. This leaves me 1 more unit of "cost". I would prefer no address translation to again keep the hit time down. If I didn't select this one, I would probably pick way prediction.