Program 3 is due at the beginning of class on **February 28th**. I reserve the right to have you “demo” your program for me and explain various parts of your code. Such a demo will take about 10 minutes. Late submissions will be penalized.

In program 3 you will design one or more 'C' programs to measure the effect of cache memory on program execution speed. *Cache* is a layer of the memory hierarchy that stands between main memory and the CPU. Understanding how to write a program to take advantage of cache is important in optimizing performance. A memory reference that can be satisfied from cache is easily 100 times faster than one that must go to main memory.

Semiconductor memory comes in various speeds, and faster memory costs more. A (not recent) advertisement lists 32KB of 12 ns memory for $12.75. For $11.50, the same ad offers 256KB of 70 ns memory. If you built an 8 Meg memory for your PC entirely with the faster memory, you’d spend about $3000 just on the memory, whereas 8 MB of the slower memory would cost only $368. For some applications, you may want to spend more money. For example, in designing a “supercomputer”, which is designed to get the biggest bang regardless of cost, you would choose the more expensive memory solution.

Caching cleverly combines fast and slow memory to achieve advantages of each. Cache exists in nearly all computers built today, from PCs to large compute/storage servers. The clever cache organization is as follows: Main memory consists of cheaper, slower chips. We use the fast, expensive memory (i.e. cache) to store copies of (selected) recently used values and their addresses in a cache. A cache controller is interposed between the CPU and the two different memory technologies. When the CPU stores a value, a copy of the value and the address is saved in the fast memory. When the CPU asks for the contents of an address, the cache controller checks to see if it has that address stored in fast memory, and if so, it can return the value more quickly than if the value is only stored in the slow memory.

If 90 out of 100 requests can be satisfied by reference to the cache, and a cache reference takes 12 ns, and the remaining 10 requests take 80 ns to complete, the average time for a memory request will be $0.90 \times 12 \text{ ns} + 0.10 \times 80 \text{ ns} = 18.8 \text{ ns}$. Of course, because we now have to store the address as well as the value, we’ll lose big if we have to have 7MB (+28MB of address) of expensive fast memory in order to reach a 90% hit rate.

It turns out that most programs exhibit good **temporal locality**, that is, they tend to refer to the same words of memory over and over, either in code loops, or as data accesses. So a PC cache of 128K is big enough to achieve a 90% hit rate for most programs. In order to avoid having to store another 512KB of addresses, several different schemes are used; one frequent scheme is to organize adjacent bytes into cache blocks that can share the same address. For example, if the cache is organized into 128-byte cache blocks, each of the 128 bytes in a block will have the same high-order bits, and so we need store only 1/128 as many addresses. For our example, we could get away with storing only 1024 addresses.

Most current operating systems have no direct role in the implementation of caching; the hardware tries to hide the details from the software, and the trade-offs change rapidly. But if you were attempting to maximize the performance of your computer, you might want to take cache properties into account. Note that if you are running a program on our hypothetical PC (mentioned above) that never hits cache, your program can run 70/12 times longer than a program which is carefully tuned so that it always hits cache.
The computer system you are using probably has a cache system. In this programming assignment, you’ll try to discover what its parameters are. You’ll attempt to answer the questions:

1. What is the difference in execution time between a reference to main memory and a reference to cache?
2. How big is a cache block?
3. How big is the cache?
4. How long does a reference to main memory take to complete?
5. How long does a reference that can be satisfied from cache take to complete?

The basic idea is to compare the execution time of two loops, one of which tries to force reading a new value from main memory at each iteration. Because we are talking only a few ns difference between the two read times, you’ll have to execute a lot of reads in order to be able to measure the difference.

The test program(s) for this problem would be easier to program in assembly language. I don’t expect you to do that. However, you should probably determine what machine language instructions occur in the loops you are timing, because the loops will obviously take different amounts of time to run if they include different instructions. On UNIX systems, one way to do this is with the -S switch to the C compiler. Some PC compilers will not provide a similar switch; in this case you may be able to use a debugger to examine the program.

Each of your final submitted program(s) won’t be very long, although you may write and discard several versions “along the way”, because they may not work. This is a difficult assignment. So give me a clear explanation of how you propose to perform the measurements, and what you think may have worked or gone wrong. Put this explanation in a file called, “README.txt”, along with your answers to the five questions listed above.

Many modern computer systems have several layers of cache. For example most current microprocessors have a significant amount of cache on the chip with the CPU (L1 cache), and also have an off-chip cache(s) that are significantly faster than main memory (L2 and L3 caches), but slower than L1 cache. If you are still full of vim after dealing with the first five questions, you might attempt to consider how you would design a 'C' program to determine whether there are two (or more) levels of cache in your system. I will give extra credit points for developing a correct 'C' solution to answer the question regarding multiple levels of cache.

You must submit (using submit on stono) the following files:

1. README.txt
2. 'C' programs which demonstrate the answers to the five questions listed above
3. Makefile

The idea for this assignment was taken from Stallings, “Programming Projects” supplement to Operating Systems, 7th Edition.