Problems 1, 3, and 5 will be graded. There are 70 points on these problems. *Note: You must do all the problems, even the non-graded ones.* If you do not do some of them, half as many points as they are worth will be subtracted from your score on the graded problems.

**Problem 1.** (15 points) You have designed a system with split instruction and data caches. The data cache uses writeback, and 50% of all replaced blocks are dirty. The miss rate for instructions is 1%, and the miss rate for data is 2%. Your measurements have shown that 15% of instructions are loads and 5% are stores, and that the average CPI (of all instructions) is 1.3 assuming no memory stalls (a "perfect" memory system). If the miss penalty for reads and writes to main memory is 25 cycles, what is the overall CPI including memory stalls?

**Problem 2.** (20 points) As we have seen in class, in order to perform TLB lookup at the same time a set-associative cache is beginning to be searched, some restrictions on the length of the displacement field are necessary. Assume we desire to do TLB lookup concurrently with cache search, and answer the following questions.

(a) If both the (main-memory) page size and the (cache) line size are held fixed, how does an increase in cache size (the number of lines in the cache) affect the number of lines required in each set (the "set size")? Justify your answer.

(b) If the cache contains 16K words, pages are 1K words long, and lines contain 16 words, what range of set sizes will allow simultaneous TLB and cache access?

(c) Does the required set size depend on the line size? (For example, if you redid part (b) assuming a line size of 32 words, or 64 words, etc., would your answer change?)

(d) Suppose now that the set size and line size are held fixed. How does an increase in cache size affect the required page size? Justify your answer.

**Problem 3.** (20 points) Set-associative and sectored caches are two compromises between direct mapping and full association. A cyclic reference string of order $n$ is a sequence of references to blocks

$$0, 1, 2, \ldots, n-2, n-1, 0, 1, 2, \ldots, n-2, n-1, 0, 1, \ldots$$

Assume that LRU replacement is used in both kinds of caches, and that

- there are $f$ lines in each cache,
- there are $b$ blocks per set in the set-associative cache,
- there are $s$ sectors in the sectored cache,
- there are $n$ distinct blocks in the cyclic reference string.

(a) Suppose $b = s$. Which kind of cache has a higher hit ratio? Does the answer depend on $f$ or $n$? How?

(b) Suppose $b \neq s$. Is the answer still the same as in part (a)? Why?

**Problem 4.** (10 points) *Hennessy & Patterson 5.13* McFarling [1989] found that the best memory-hierarchy performance occurred when it was possible to prevent some instructions from entering the cache.
(a) <5.5> Explain why McFarling’s result could be true.  *(Hint: The case where it is best not to cache a particular instruction may be relatively rare.)*

(b) <5.2, 5.5> The four memory-hierarchy questions (Section 5.2) form a model for describing cache designs. Where does a cache that does not always read-allocate fit into this model?

**Problem 5.** *(35 points)* Compare 0-, 1-, 2-, and 3-address machines by writing programs to compute

\[ g := (a + b \times c) ÷ (d - e \times f) \]

for each of the four machines. The instructions available for use are as follows:

<table>
<thead>
<tr>
<th>0-address</th>
<th>1-address</th>
<th>2-address</th>
<th>3-address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH M</td>
<td>LOAD M</td>
<td>MOV (X := Y)</td>
<td>MOV (X := Y)</td>
</tr>
<tr>
<td>POP M</td>
<td>STORE M</td>
<td>ADD (X := X + Y)</td>
<td>ADD (X := Y + Z)</td>
</tr>
<tr>
<td>ADD</td>
<td>ADD M</td>
<td>SUB (X := X - Y)</td>
<td>SUB (X := Y - Z)</td>
</tr>
<tr>
<td>SUB</td>
<td>SUB M</td>
<td>MUL (X := X * Y)</td>
<td>MUL (X := Y * Z)</td>
</tr>
<tr>
<td>MUL</td>
<td>MUL M</td>
<td>DIV (X := X / Y)</td>
<td>DIV (X := Y / Z)</td>
</tr>
<tr>
<td>DIV</td>
<td>DIV M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) M is a 16-bit memory address, and X, Y, and Z are either 16-bit addresses or 4-bit register numbers. The zero-address machine uses a stack, the 1-address machine uses an accumulator, and the other two have 16 registers and instructions operating on all combinations of memory locations and registers. SUB X, Y subtracts Y from X, and SUB X, Y, Z subtracts Z from Y and puts the result in X. Assuming 8-bit opcodes and instruction lengths that are multiples of 4 bits, how many bits does each machine need to compute g? *(All of the variables a, b, c, d, e, and f are initially stored in main memory, not registers, and g must be stored back to memory.)*

(b) Show that for each machine, there is some program that it can represent more compactly than all the other three machines. That is, display one program for which the zero-address architecture “beats” all the others, one program on which the one-address architecture is best, and so forth. Show the code for these programs in all four machines. In the event that it is not possible to find different programs for which each of the machines are optimal, explain how you have arrived at this conclusion. *(Hint: One of the architectures cannot be optimal for a program P unless one of the other architectures “runs out of registers” in which to store temporary values generated during the execution of P. Which architectures are these? Why?)*