For More Practice

Locality

7.6 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

7.7 [10] <§7.2> Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

7.8 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

Cache Performance

7.15 [10] <§7.2> Suppose a processor with a 16-word block size has an effective miss rate per instruction of 0.5%. Assume that the CPI without cache misses is 1.2. Using the memories described in Figure 7.11 on page 489 and Exercise 7.14, how much faster is this processor when using the wide memory than when using narrow or interleaved memories?

Cache Configurations

7.23 [10] <§7.2, B.5> You have been given 18 32K × 16-bit SRAMs to build an instruction cache for a processor with a 32-bit address. What is the largest size (i.e., the largest size of the data storage area in bytes) direct-mapped instruction cache that you can build with one-word (64-bit) blocks? Show the breakdown of the address into its cache access components (for an example, see Figure 7.8) and describe how the various SRAM chips will be used. (Hint: You may not need all of them.)

7.24 [10] <§7.2, B.5> This exercise is similar to Exercise 7.23, except that this time you decide to build a direct-mapped cache with four-word blocks as in Figure 7.10. Once again show the breakdown of the address and describe how the chips are used.

Cache Operation

7.25 [10] <§7.3> Using the series of references given in Exercise 7.9, show the hits and misses and final cache contents for a two-way set-associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

7.26 [10] <§7.3> Using the series of references given in Exercise 7.9, show the hits and misses and final cache contents for a fully associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

7.27 [10] <§7.3> Using the series of references given in Exercise 7.9, show the hits and misses and final cache contents for a fully associative cache with four-word blocks and a *total size* of 16 words. Assume LRU replacement.

Cache Configurations

7.30 [10] <§7.3> This exercise concerns caches of unusual sizes. Can you make a fully associative cache containing exactly 3K words of data? How about a set-associative cache or a direct-mapped cache containing exactly 3K words of data? For each of these, describe how or why not. Remember that $1K = 2^{10}$.

7.31 [10] <§7.3> This exercise is similar to Exercise 7.30, except replace 3K with 300. Remember that $300 = 3 \times 10^2$.

7.36 [10] <§§7.3, B.5> This exercise is similar to Exercise 7.23, except that this time you decide to build a three-way set-associative cache with one-word blocks. Once again show the breakdown of the address (see Figure 7.17 for an example of a four-way set-associative cache) and describe how the chips are used. Note that each SRAM will only perform a single read per cache access.

Memory Hierarchy Interactions

7.37 [5] <§§7.2–7.4> Rank each of the possible event combinations appearing in Figure 7.26 on page 527 according to how frequently you think they would occur.

7.43 [15] <§7.4> Page tables require fairly large amounts of memory (as described in the Elaboration on page 519), even if most of the entries are invalid. One solution is to use a hierarchy of page tables. The virtual page number, as described in Figure 7.20 on page 513, can be broken up into two pieces, a "page table number" and a "page table offset." The page table number can be used to index a first-level page table that provides a physical address for a second-level page table, assuming it resides in memory (if not, a first-level page fault will occur and the page table itself will need to be brought in from disk). The page table offset is used to index into the second-level page table to retrieve the physical page number. One obvious way to arrange such a scheme is to have the second-level page tables occupy exactly one page of memory. Assuming a 32-bit virtual address space with 4 KB pages and 4 bytes per page table entry, how many bytes will each program need to use to store the first-level page table (which must always be in memory)? Provide figures similar to Figures 7.19, 7.20, and 7.21 (pages 512–517) that demonstrate your understanding of this idea.

Hierarchical Page Tables

7.44 [15] <§7.4> Assuming that we use the two-level hierarchical page table described in Exercise 7.43 and that exactly one second-level page table is in memory and exactly half of its entries are valid, how many bytes of memory in our virtual address space actually reside in physical memory? (Hint: The second-level page table occupies exactly one page of physical memory.)