Name: _____

Directions: Work on this sheet (**on both sides**, if needed) only; **do not turn in any supplementary sheets of paper**. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK.**

1. (10) Fill in the blank: A special cache used to store part of the page table is called a(n) _____.

2. (10) Is the mythical machine used for examples in our printed lecture notes on I/O designed for memory-mapped I/O? State clearly why or why not.

3. (15) Consider the 2-way set-associative example which begins at the bottom of p.571 of Patterson and Hennessy. Suppose after the 5 memory accesses shown, we then have 6 more, to memory addresses 4, 1, 3, 1, 0 and 5. How many misses will these latter 6 accesses produce? Remember to show your work.

4. (10) In our printed lecture notes on OSs we discussed the notion of a *system call*. State which specific MIPS instruction performs this operation, and show the page from our assigned reading in Patterson and Hennessy where this is discussed.

5. (10) Fill in the blank: The authors of our textbook neglected to mention that Dirty bits are needed in some kinds of caches too, specifically caches which have a _____ policy.

6. (15) Fill in the blanks (in the first case answering either "hardware" or "software"): Suppose you receive a message like "segmentation fault" or "bus error" while running it program. Such a problem was first discovered by the _____ while checking the _____ bits in the _____.

7. (15) Suppose we are building a set-associative instruction cache for a machine on which we anticipate mainly running a particular program which makes highly frequent calls to a few very short functions. State what implications this has on our choice of degree of set-associativity (high or low) and block size (large or small). Explain your answer with great thoroughness.

8. (15) In this problem, you will write a portion of an OS's page fault handler for the system shown in Fig.7.22 of Patterson and Hennessy. Assume the OS maintains the following global variables:

```
int PTR,  // contents of Page Table Reg
    FaultVAddr,  // virtual memory address
                 // which produced page fault
    EvictVPN,  // virtual page number of page
               // to be evicted
    EvictPPN,  // physical page number of page
               // to be evicted
    EvictDL;  // disk location for evicted page
```

Assume that when the OS is running (*system mode*), all addresses issued by the OS are physical, e.g. MOV AX,[200] really does mean physical location 200. Also assume that each line of the page table takes up one full word of memory, right-justified, and that there is no TLB.

Write the portion of the OS which updates the page table. (Do not include the code for copying the evicted page to disk and loading the new page.) Write your code in C.

Solutions:

1. TLB.

2. The answer was supposed to be no, but due to a typo in the handout, I accepted all answers.

3. 5.

4. Page 597, syscall.

5. Write-back.a

6. Hardware; permission; page table.

7. High, small.

8.

```
FaultVPN = FaultVAddr >> 12;
FaultPgTblEntry = (int *) (PTR + 4*FaultVPN);
*FaultPgTblEntry = EvictPPN + 1 << 19;
EvictPgTblEntry = (int *) (PTR + 4*EvictVPN);
*EvictPgTblEntry = EvictDL;
```