Introduction to Computer Engineering (E114)

Harris Spring 1999

Problem Set 9 **Due: Friday, April 16**

Reading: Chapter 6.1-6.7, 7.1-7.9

1) Cache Finger Exercises

a) Come up with a sequence of addresses for a MIPS processor for which a direct-mapped cache of size 64 bytes, block size 16 bytes, outperforms a fully-associative cache with the same block size, using LRU replacement.

Suppose the block size is B bytes, S is the number of sets, D is the degree of associativity (also called the set size or the number of ways), and addresses are made up of A bits. Assume that the cache is word addressed, i.e., the low two bits of the address are always 0.

b) In terms of the parameters described above, what is the cache size in bytes? This number should include just the data portion of the cache.

c) In terms of the parameters above, what is the total number of bytes required to store the tags?

d) What are S and D for a fully-associative cache of size x bytes, with block size b?

e) What is S for a direct-mapped cache of size x bytes, with block size b?

Consider the following repeating sequence of `lw` addresses (in hex):

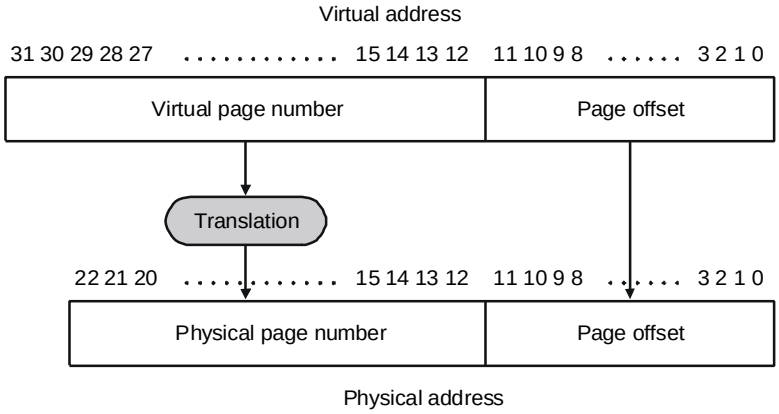
repeat 60 64 68 6C 70 74 78 7C 80 84 88 8C 90 94 98 9C 0 4 8 C 10 14 18 1C 20

f) Assuming LRU replacement, determine the effective miss rate if the sequence is input to the following caches, ignoring startup effects (i.e. compulsary misses).

- Direct-mapped cache (D=1), S= 16, B = 4 bytes
- Fully-associative cache (D=16), B=4 bytes
- Two-way, set-associative cache (D=2), S=8, B=4 bytes
- Direct-mapped cache (D=1), S=8, B=8 bytes

2) Virtual Memory

Suppose we wish to design a virtual memory system. We would like to be able to address a total of 2^{32} bytes. We've got as much disk space as we want, but are limited to only 8 MB of semiconductor memory, i.e. to physical addresses that are 23 bits long. Assume in the following that we have decided to divide blocks of memory (both virtual and physical) into 4 KB pages.



a) How many virtual pages are there in our system?

b) How many physical pages are there in our system?

c) Think of physical memory as a cache for virtual memory; virtual pages are "pulled in" to physical memory at different locations. We must come up with a way of mapping each virtual page into a physical page. If a "direct-mapped" scheme is used, i.e., if only the low order bits of the virtual page number are used to determine the physical page number, how many virtual pages are mapped to each physical page? Why is this "direct mapping" a bad plan? (HINT: try to provide an example of thrashing.)

d) Clearly, a more flexible and dynamic scheme for translating virtual addresses into physical addresses is required. Suppose we use a page table to store mappings (translations from virtual page number to physical page number). How many page table entries will the page table for our system contain?

e) Assume that in addition to the physical page number, each page table entry also contains some status information in the form of an R bit (set if this particular page table entry refers to a virtual page that is actually resident in physical memory), and an M bit (which is set if a particular physical page has been modified). How many bytes long is each page table entry? Sketch the layout of the page table. What is the total size of the page table in bytes? (Round up to an integer number of bytes).

f) Our page table is large enough that we don't want to keep the entire table resident in main memory. We want to implement a *paged* page table which can be stored primarily on disk, with only the pages of the page table needed by a running program resident in physical memory at any particular time. One way to do this is to use a two-level page table. With the first level always resident in memory, pages of the second level of the table can be swapped in on demand. For our system, how many pages long will the first level page table be? Note that you must first figure out how many pages the page table itself takes up and then divide this by the number of locations contained in one page. (Optional: make a sketch of the two-level page table, indicating the function of the entries in the first and second level).

g) Translating the virtual address into a physical address in our system is potentially very painful. In order to do the translation the system must:

- Use the top N bits of the virtual address to look in the first level page table to see if the relevant page of the second level page table is in physical memory.
- Look on the disk for this page and swap it in if it's not in physical memory.
- Once the relevant page of the second level page table is in physical memory use the next M bits of virtual memory to index into this page and find the relevant page table entry for the physical page number of the data we are after.
- If the page containing this data is not resident in physical memory, swap it in from the disk.
- Once the page containing the data we're after is in physical memory, read the data from this page using the final P bits from the virtual address.

What are the values for N, M, and P mentioned above?

Suppose we wish to enhance the performance of our virtual memory system by adding a fully associative 64 entry cache for caching VPN->PPN translations (called a translation lookaside buffer, or TLB). Each entry in the TLB holds one address tranlation. If a particular translation that we need is resident in this cache, we can bypass the complicated procedure outlined above: the physical page number is quickly available. See Figure 7.24 for a picture of a TLB.

h) How many bits are in each entry of the TLB (including the valid bit). What is the total size of the TLB in bits?

3) Pipelining

Do Exercises 6.1, 6.11, 6.12, and 6.13.

4) Time

Please indicate how many hours you spent on this problem set. This will not affect your grade, but will be helpful for calibrating the workload for next semester's class.

Some of these problems are courtesy old MIT 6.004 (Computation Structures) problem sets.