EN164: Design Of Computer Systems

Homework 1

Andrew Chin *April 6, 2007*

Problem 1:

Consider a virtual memory system with 40-bit virtual byte address, 16KB pages, and 36-bit physical address. What is the total size of the page table for each process, assuming the valid, protection, and dirty markers take 4 bits and all virtual pages are in use?

16KB $= 2^{14} \Rightarrow$ 14 bit page offset

$40 - 14 = 26$ bit address

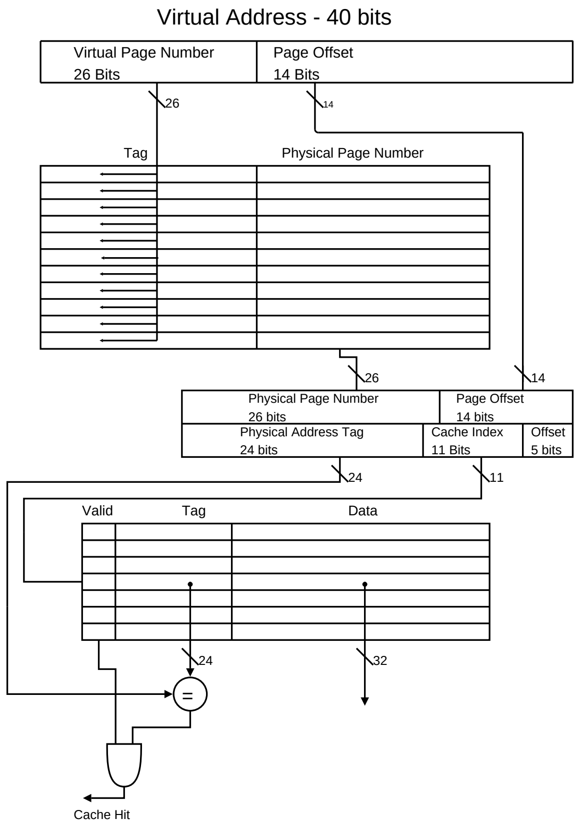
2^{26} page table entries.

(4 bit markers + 36 bit physical address) * $2^{26} = 320$ megabytes.

Problem 2:

Assume a 2-way TLB with a total of 256 TLB entries. The physical address is for a 64KB direct mapped cache with 32B blocks.

a)



b) *What is the main advantage of using a virtually-indexed, physically tagged scheme to access data from a level 1 cache? Is it possible to use such a scheme for the cache described here? Explain why or why not? If not, suggest how the cache configuration may be modified so this scheme can work*

   The advantage is that the page offset from the virtual address can potentially be used, without translation, as the index for the directly mapped cache. This allows us to index into the cache at the same time as the TLB is looking up the physical page number (compare this to the case when we need to translate the page offset before we can use it to index into the cache. In this case, we must wait for the TLB lookup to complete before we can start the cache lookup).

   No, it is not possible to use such a scheme for this cache. In the cache described in this problem the page offset from the virtual address of 14 bits (bits 0 to 13). The index for the cache is 11 bits wide. Given that we have 5 bits of data offset, this means that the index is bits 5 to 15. This means we have to wait for bits 14 and 15 from the TLB (the lower two bits from the TLB).

   The cache configuration in this problem needs to be modified in such a way that the page offset (the lower 14 bits from the virtual address) can be used directly (without translation) as the index into the directly mapped cache. Currently, we need to wait for the TLB lookup to complete before we can generate the cache index: the lower 2 bits from the physical page number (which the TLB outputs) are used as the upper 2 bits of the cache index. In order to remedy this, we have two options: a) increase the number of bits in the page offset by 2, or b) decrease the size of the cache index by 2 bits, or c) use a combination of both (a) and (b).

   In case (a) we can do this by increasing the size of each page. In case (b) we can do this by decreasing the number of blocks in the directly mapped cache. In case (c), we can do both. An example for case (c): Increase the page size from 16KB to 32KB and increase the size of the blocks in the cache from 32B to 64B, or decease the size of the cache from 64K to 32K.

Problem 3:

a) *How can way prediction reduce the number of pins needed on this chip to read L2 tags and data, and what is the impact on the performance compared to a package with a full complement of pins to interface to the L2 cache?* Without way prediction, the L2 cache will needs to send back the tag and data info for both ways (which will then be used by the on-chip L2 tag comparison circuitry).

   With way prediction, the L2 cache only needs to send back the tag and data info for one of the ways, cutting the number of cache-to-chip pins in half. There will also be the addition of one chip-to-cache pin (for way prediction)

   Using the reduced number of chips, there is a performance hit when the way-prediction circuitry predicts incorrectly. With the reduced number of pins, the chip will have to go off chip again to get the correct data from the correct way. Using the full set of pins, this would not have been necessary, as the tag comparison circuitry would have simply chosen the correct data from the correct way.

b) *What is the performance drawback of just using the same smaller number of pins but not including way prediction?* If way prediction is not used, but there still is the smaller number of pins, then the chip will have to first get the tag/data from the first way, and compare tags. If the tags don't match, then it will have to go off chip again to the cache to get the data from the second way.

Problem 4:

[See Attached Spreadsheet]