6.004 Computation Structures
Spring 2009

Virtual memory

Problem 1. Consider a virtual memory system that uses a single-level page map to translate virtual addresses into physical addresses. Each of the questions below asks you to consider what happens when one of the design parameters of the original system is changed.

A. ★ If the physical memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

increases by 1 bit. Assuming the page size remains the same, there are now twice as many physical pages, so the physical page number needs to expand by 1 bit.

B. ★ If the physical memory size (in bytes) is doubled, how does the number of entries in the page map change?

no change. The number of entries in the page table is determined by the size of the virtual address and the size of a page -- it's not affected by the size of physical memory.

C. ★ If the virtual memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

no change. The number of bits in a page table entry is determined by the number of control bits (usually 2: dirty and resident) and the number of physical pages -- the size of each entry is not affected by the size of virtual memory.

D. ★ If the virtual memory size (in bytes) is doubled, how does the number of entries in the page map change?

the number of entries doubles. Assuming the page size remains the same, there are now twice as many virtual pages and so there needs to be twice as many entries in the page map.

E. ★ If the page size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

each entry is one bit smaller. Doubling the page size while maintaining the size of physical memory means there are half as many physical pages as before. So the size of the physical page number field decreases by one bit.

F. ★ If the page size (in bytes) is doubled, how does the number of entries in the page map change?

there are half as many entries. Doubling the page size while maintaining the size of virtual memory means there are half as many virtual pages as before. So the number of page table entries is also cut in half.

G. ★ The following table shows the first 8 entries in the page map. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

Virtual page	Valid bit	Physical page
0	0	7
1	1	9
2	0	3
3	1	2
4	1	5
5	0	5
6	0	4
7	1	1

If there are 1024 (2^{10}) bytes per page, what is the physical address corresponding to the decimal virtual address 3956?

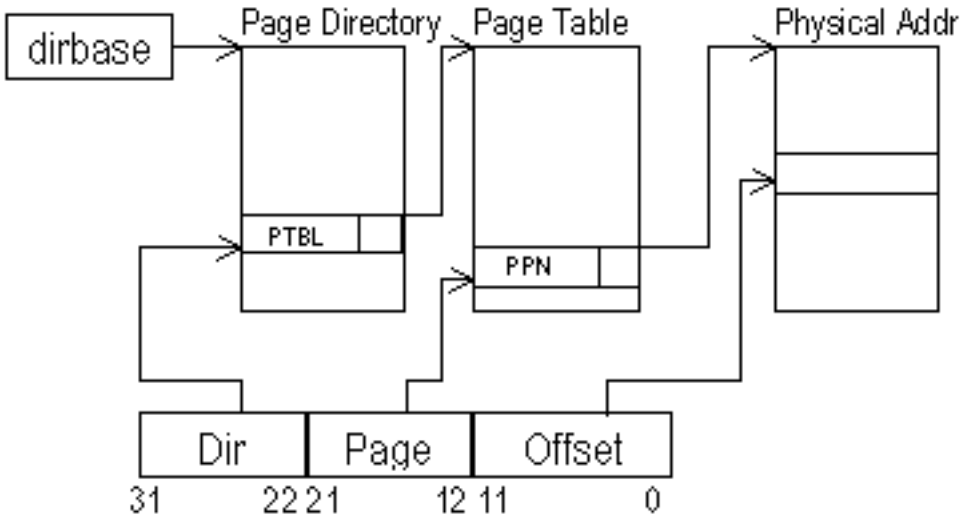
3956 = 0xF74. So the virtual page number is 3 with a page offset of 0x374. Looking up page table entry for virtual page 3, we see that the page is resident in memory (valid bit = 1) and lives in physical page 2. So the corresponding physical address is (2<<10)+0x374 = 0xB74 = 2932.

Problem 2. A particular 32-bit microprocessor includes support for paged virtual memory addressing with 2^{12} byte pages. The mapping of virtual to physical addresses requires two translation steps:

1. The most significant 10 bits of the virtual address (the Dir field) are multiplied by 4 and appended to the 20 most significant bits of the dirbase (directory base) register to get the address in main memory of a page directory entry. Each entry in the page directory is a 32-bit record composed of a 20-bit PTBL field and various control bits (Present, Dirty, Read-only, etc.).

2. The bits of the Page field (virtual address bits 21 to 12) are multiplied by 4 and appended to the PTBL field to form the page-table address. This page table address references a 32-bit page table entry. Each page table entry is composed of a 20-bit physical page number (PPN) and a series of control bits.

All page-table entries and the page directory are stored in main memory. The results of these translations are cached in a fully-associative translation look-aside buffer (TLB) with a total of 64 entries, and a LRU replacement strategy is used on TLB misses.



A. ★ Given a computer system with 2^{27} bytes of physical memory that uses the virtual-to-physical address translation scheme described, how many pages of physical memory are there?

$2^{15} = 2^{27}/2^{12}$ = the size of physical memory divided by the size of each page.

B. ★ How many memory pages does the Page Directory occupy?

We are told that the Page Directory index is 10 bits, implying $2^{10} = 1024$ entries. Each entry occupies 4 bytes, so the total size of the of the Page Directory is $4*2^{10} = 2^{12}$ bytes, or exactly one page.

C. ★ What is the approximate maximum size for a process's working set that still achieves a 100% TLB hit rate?

The TLB has 64 entries, so to achieve 100% hit rate in the TLB we can access only 64 different pages as part of our working set. 64 pages = $64*2^{12} = 2^{18}$ bytes.

D. ★ How large must the tag field of the TLB be?

The tag field should contain all the bits of the virtual page number, i.e., bits 12 through 31, a total of 20 bits.

E. ★ A control bit, C, in each page table entry determines if memory references to that page are cacheable. In order to support this feature, which of the following statements concerning the interaction between virtual-to-physical address translations and caching must be true?

A. The cache tags must contain physical addresses
B. Each memory access requires a virtual-address translation to take place in parallel with the cache access
C. The status of the cacheable bit, C, needs only to be considered on a cache miss
D. Page table entries with their dirty bit set should clear their cacheable bit
E. All of the above

C. We only need to worry if a page is cacheable if we're considering bringing some of its entries into the cache, and we only do this if the access can't be satisfied from current contents of the cache.

Problem 3. Consider two possible page-replacement strategies: LRU (the least recently used page is replaced) and FIFO (the page that has been in the memory longest is replaced). The merit of a page-replacement strategy is judged by its hit ratio.

Assume that, after space has been reserved for the page table, the interrupt service routines, and the operating-system kernel, there is only sufficient room left in the main memory for *four* user-program pages. Assume also that initially virtual pages 1, 2, 3, and 4 of the user program are brought into physical memory in that order.

A. ★ For each of the two strategies, what pages will be in the memory at the end of the following sequence of virtual page accesses? Read the sequence from left to right: (6, 3, 2, 8, 4).

LRU:

start: 4 3 2 1
access 6: replace 1 => 6 4 3 2
access 3: reorder list => 3 6 4 2
access 2: reorder list => 2 3 6 4
access 8: replace 4 => 8 2 3 6
access 4: replace 6 => 4 8 2 3

FIFO:

start: 4 3 2 1
access 6: replace 1 => 6 4 3 2
access 3: no change => 6 4 3 2
access 2: no change => 6 4 3 2
access 8: replace 2 => 8 6 4 3
access 4: no change => 8 6 4 3

B. ★ Which (if either) replacement strategy will work best when the machine accesses pages in the following (stack) order: (3, 4, 5, 6, 7, 6, 5, 4, 3, 4, 5, 6, 7, 6, ...)?

LRU misses on pages 3 & 7 => 2/8 miss rate.

FIFO doesn't work well on stack accesses => 5/8 miss rate.

C. Which (if either) replacement strategy will work best when the machine accesses pages in the following (repeated sequence) order: (3, 4, 5, 6, 7, 3, 4, 5, 6, 7, ...).

Both strategies have a 100% miss rate in the steady state.

D. Which (if either) replacement strategy will work best when the machine accesses pages in a randomly selected order, such as (3, 4, 2, 8, 7, 2, 5, 6, 3, 4, 8, ...).

Neither FIFO nor LRU is guaranteed to be the better strategy in dealing with random accesses since there is no locality to the reference stream.

Problem 4. A paged memory with a one-level page table has the following parameters: The pages are 2^P bytes long; virtual addresses are V bits long, organized as follows:

The page-table starts at physical address PTBL; and each page-table entry is a 4-byte longword, so that, given a virtual address, the relevant page-table entry can be found at PTBL + (page number)*4. Answer the following in terms of the parameters P and V:

A. How many bits long is the "offset in page" field?

It takes $\log_2(2^P) = P$ address bits to select a single byte from a page with 2^P bytes.

B. How many bits long is the "virtual page number" field?

Since there a P bits in the offset field, the remaining V-P bits are part of the virtual page number.

C. How many entries does the page table have, and what is the highest address occupied by a page-table entry?

Since the virtual page number field has V-P bits, there are 2^{V-P} virtual pages and each has its own entry in the page table. Each entry is 4 bytes longs, so the highest address occupied by a page table entry is PTBL + $4*(2^{(V-P)}-1)$.

D. How many pages long is the page table?

There are $2^P/4$ page table entries per page and 2^{V-P} pages, so the page table is $2^{V-P}/2^{P-2} = 2^{V-2P+2}$ pages long.

E. What is the smallest value of P such that the page table fits into one page?

Using the formula from the previous question, to make the page table fit in one page, we want V-2P+2 = 0. Solving for P we get P = V/2 + 1.

F. What relationships, if any, must hold between P, V, and the size of physical memory?

Suppose physical memory contained 2^M bytes. Then

- The physical page number must fit in 30 bits since we reserve 2 bits of the 32-bits page table entry for the dirty and resident control bits. So 30 >= M - P.
- It useful to have room in memory for at least one page other than those occupied by the page map. So M > V-P+2.

Problem 5.

A. If virtual addresses are V bits long, physical addresses are A bits long, the page size is 2^P bytes, and a one-level page table is used, give an expression for the size of the page table.

There are 2^{V-P} pages and the page table entry for each page contains a physical page number (A-P bits), a dirty bit (1 bit) and a resident bit (1 bit). So the page table occupies $2^{V-P}(A-P+2)$ bits.

Problem 6. Adverbs Unlimited has recently added a new product, the VIRTUALLY to the product line introduced in an earlier tutorial problem. The VIRTUALLY has a fully-associative cache with 256 entries and a block size of 1, 2^{20} bytes of physical memory, 16-bit virtual addresses, and a 2^6-entry page map. The VIRTUALLY will be used to support multiuser time-sharing. The page map holds the address translation for a single (current) process and must be reloaded (by the kernel) at each process switch. The cache is located between the page map and main memory.

A. What is the page size?

   page size in bytes = size of virtual address divided by number of entries in the page map = $2^{16}/2^6 = 2^{10}$ bytes per page.

B. Which virtual address lines are used to form the index to the page map?

   The virtual page number is used as the index to the page map. The virtual page number includes all virtual address bits that aren't part of the page offset. Since there 2^{10} bytes per page, the page offset requires 10 bits, i.e., address bits 0 through 9. The remaining six bits (bits 10 through 15) form the virtual page number.

C. Can the operation of the cache and page-map be overlapped? Explain in a single sentence.

   No since we can't search the cache until we have the physical page number from the page map.

D. Under what circumstances, if any, must the cache be invalidated (that is, its entries marked as invalid)?

   Since the cache is located after the page map, it caches physical addresses. So it must be invalidated when there is a page replacement due to a page fault, since this operation changes the contents of physical memory.

Problem 7.

A. Program A consists of 1000 consecutive ADD instructions, while program B consists of a loop that executes a single ADD instruction 1000 times. You run both programs on a certain machine

and find that program B consistently executes faster. Give two plausible explanations.

Explanation #1: one would expect the loop to achieve a higher hit rate in the cache since it involves many fewer instruction words.

Explanation #2: the loop, occupying many fewer instruction words, should all fit onto a single page. The 1000 instructions might span several pages and hence their execution may involve some page faults.