Section 7: Virtual Memory, Caching

CS 162

March 12, 2021

Contents

1 Vocabulary

- **Virtual Memory** - Virtual Memory is a memory management technique in which every process operates in its own address space, under the assumption that it has the entire address space to itself. A virtual address requires translation into a physical address to actually access the system's memory.
- **Memory Management Unit** - The memory management unit (MMU) is responsible for translating a process' virtual addresses into the corresponding physical address for accessing physical memory. It does all the calculation associating with mapping virtual address to physical addresses, and then populates the address translation structures.
- **Address Translation Structures** - There are two kinds you learned about in lecture: segmentation and page tables. Segments are linearly addressed chunks of memory that typically contain logically-related information, such as program code, data, stack of a single process. They are of the form (s,i) where memory addresses must be within an offset of i from base segment s. A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. Virtual addresses are used by the accessing process, while physical addresses are used by the hardware or more specifically to the RAM.
- **Translation Lookaside Buffer (TLB)** - A translation lookaside buffer (TLB) is a cache that memory management hardware uses to improve virtual address translation speed. It stores virtual address to physical address mappings, so that the MMU can store recently used address mappings instead of having to retrieve them mutliple times through page table accesses.
- **Cache** - A repository for copies that can be accessed more quickly than the original. Caches are good when the frequent case is frequent *enough* and the infrequent case is not too expensive. ensures locality in two ways: temporal (time), keeping recently accessed data items 'saved', and spatial (space), since we often bring in contiguous blocks of data to the cache upon a cache miss.
- **AMAT** - Average Memory Access Time: a key measure for cache performance. The formula is (Hit Rate x Hit Time) + (Miss Rate x Miss Time) where Hit Rate + Miss Rate = 1.
- **Compulsory Miss** - The miss that occurs on the first reference to a block. This also happens with a 'cold' cache or a process migration. There's essentially nothing that you can do about this type of miss, but over the course of time, compulsory misses become insignificant compared to all the other memory accesses that occur.
- **Capacity Miss** - The miss that occurs when the cache can't contain all the blocks that the program accesses. One solution for capacity misses is increasing the cache size.
- **Conflict Miss** - The miss that occurs when multiple memory locations are mapped to the same cache location (i.e a collision occurs). In order to prevent conflict misses, you should either increase the cache size or increase the associativity of the cache. These technically do not exist in virtual memory, since we use fully-associative caches.
- **Coherence Miss** - Coherence misses are caused by external processors or I/O devices that update what's in memory (i.e invalidates the previously cached data).
- **Tag** - Bits used to identify the block - should match the block's address. If no candidates match, cache reports a cache miss.
- **Index** - Bits used to find where to look up candidates in the cache. We must make sure the tag matches before reporting a cache hit.

- **Offset** - Bits used for data selection (the byte offset in the block). These are generally the lowest-order bits.

- **Direct Mapped Cache** - For a 2^N byte cache, the uppermost (32 - N) bits are the cache tag; the lowest M bits are the byte select (offset) bits where the block size is 2^M. In a direct mapped cache, there is only one entry in the cache that could possibly have a matching block.

- **N-way Set Associative Cache** - N directly mapped caches operate in parallel. The index is used to select a set from the cache, then N tags are checked against the input tag in parallel. Essentially, within each set there are N candidate cache blocks to be checked. The number of sets is X / N where X is the number of blocks held in the cache.

- **Fully Associative Cache** - N-way set associative cache, where N is the number of blocks held in the cache. Any entry can hold any block. An index is no longer needed. We compare cache tags from all cache entries against the input tag in parallel.

2 Paging and Address Translation

2.1 Conceptual Questions

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

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

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

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

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

If the page size (in bytes) is doubled, how does the number of entries in the page table change?

The following table shows the first 8 entries in the page table. 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.

Valid Bit	Physical Page
0	7
1	9
0	3
1	2
1	5
0	5
0	4
1	1

If there are 1024 bytes per page, what is the physical address corresponding to the hexadecimal virtual address 0xF74?

2.2 Page Allocation

Suppose that you have a system with 8-bit virtual memory addresses, 8 pages of virtual memory, and 4 pages of physical memory.

How large is each page? Assume memory is byte addressed.

Suppose that a program has the following memory allocation and page table.

Memory Segment	Virtual Page Number	Physical Page Number
N/A	000	NULL
Code Segment	001	10
Heap	010	11
N/A	011	NULL
N/A	100	NULL
N/A	101	NULL
N/A	110	NULL
Stack	111	01

What will the page table look like if the program runs the following function? Page out the least recently used page of memory if a page needs to be allocated when physical memory is full. Assume that the stack will never exceed one page of memory.

```
#define PAGE_SIZE 1024;

void helper(void) {
    char *args[5];
    int i;
    for (i = 0; i < 5; i++) {
        // Assume malloc allocates an entire page every time
        args[i] = (char*) malloc(PAGE_SIZE);
    }
    printf("%s", args[0]);
}
```

What happens when the system runs out of physical memory? What if the program tries to access an address that isn't in physical memory? Describe what happens in the user program, the operating system, and the hardware in these situations.

2.3 Address Translation

Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?

List the fields of a Page Table Entry (PTE) in your scheme.

Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?

With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?

The pagemap is moved to main memory and accessed via a TLB. Each main memory access takes 50 ns and each TLB access takes 10 ns. Each virtual memory access involves:

- mapping VPN to PPN using TLB (10 ns)
- if TLB miss: mapping VPN to PPN using page map in main memory (50 ns)
- accessing main memory at appropriate physical address (50 ns)

Assuming no page faults (i.e. all virtual memory is resident) what TLB hit rate is required for an average virtual memory access time of 61ns?

Assuming a TLB hit rate of .50, how does the average virtual memory access time of this scenario compare to no TLB?

3 Caching

3.1 Average Read Time

Suppose you have a system that uses a two level page table to translate virtual addresses and a cache for the translation tables and data. Each memory access takes 50ns, the cache lookup time is 5ns, and your cache hit rate is 90%. What is the average time to read a location from memory?

3.2 Average Read Time with TLB

In addition to the cache, you add a TLB to aid you in memory accesses, with an access time of 10ns. Assuming the TLB hit rate is 95%, what is the average read time for a memory operation? You should use the answer from the previous question for your calculations.

3.3 Cached Paging

Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing: 0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

Here is the current state of the page table.

Valid Bit	Physical Page Number
0	NULL
1	2
0	NULL
0	4
0	5
1	6
1	7
0	NULL

Explain what happens on a memory access.

How many TLB hits and page faults are there? What are the contents of the cache at the end of the sequence?