Inf2C Computer Systems

Tutorial 6, Week 10

Solutions

1. **Fully-associative cache**
   Consider a simple fully-associative cache with 4 lines, each containing 2 bytes of data, and assume that addresses are 6 bits wide. For the following pattern of cache accesses (similar to the direct-mapped cache exercise from last week):

Access Number	Address
1	1 0 1 1 0 0
2	0 0 1 0 1 1
3	0 1 1 0 0 1
4	0 1 0 0 0 1
5	1 0 1 1 0 0
6	0 1 1 0 0 1
7	1 0 0 1 1 0

   draw up a table of form

Access Number	Tag	Line 0	Line 1	Line 2	Line 3	Hit/Miss
1						
2						
3						
4						
5						
6						
7						

   that summarises the contents of the cache *after* each access.

   Assume that empty cache lines are filled in order and assume an LRU (least recently used) replacement policy. Note how the table would change if a FIFO (first in first out) replacement policy were used instead.

Answer: *Each 6-bit address splits into a 5-bit tag and 1-bit offset. Desired table is:*

Access Number	*Tag*	*Line 0*	*Line 1*	*Line 2*	*Line 3*	*Hit/Miss*
1	*22*	*22*				*M*
2	*5*	*22*	*5*			*M*
3	*12*	*22*	*5*	*12*		*M*
4	*8*	*22*	*5*	*12*	*8*	*M*
5	*22*	*22*	*5*	*12*	*8*	*H*
6	*12*	*22*	*5*	*12*	*8*	*H*
7	*19*	*22*	*19*	*12*	*8*	*M*

If a FIFO replacement policy is used instead, only the row for access 7 changes. It becomes:

Access Number	*Tag*	*Line 0*	*Line 1*	*Line 2*	*Line 3*	*Hit/Miss*
7	*19*	*19*	*5*	*12*	*8*	*M*

2. **Virtual memory translation and TLB**
   As discussed in class, TLBs can be used to accelerate the virtual-to-physical address translation process. Recall that TLB is a small and fast table in hardware which caches page table entries,

thereby avoiding slow page table accesses. You have a fully-associative TLB with 64 entries deployed in a system that uses 4KB physical pages. Consider a latency-critical application where TLB misses cannot be afforded. What is the maximum amount of memory the application can use to ensure a 0% TLB miss rate?

Answer: *The TLB can store 64 translations. Each physical page is 4KB. Total memory for which translations can be stored in the TLB = 64 * 4K = 256KB. If the application allocates and uses 256KB of memory, there will be no TLB misses as all the translations can be stored in the TLB.*

3. **Large pages and TLB**

Modern processors and operating systems support a few different physical page sizes: 4KB, 2MB and 1GB. Now consider that you have a fully-associative TLB with 64 entries deployed in a system which supports all three page sizes. The TLB is capable of storing translations for each page size and the 64 TLB entries are divided into:

- 4 entries for 1GB pages
- 16 entries for 2MB pages
- 44 entries for 4KB pages

Comment on when this support for multiple page sizes, more specifically the larger page sizes of 2MB and 1GB, would be beneficial for TLB performance. (*Hint: think about memory allocation in programs*)

What potential problems could arise from this TLB configuration to support multiple page sizes? (*Hint: think both about memory allocation in programs and the TLB structure*)

Answer: *This is a discussion-oriented exercise.*

(i) Large page sizes, 2MB and 1GB, are helpful when a program allocates large chunks of (contiguous) memory. For example, if a program allocates 2MB of memory, 2MB page size support will allow a single translation entry for this chunk of memory. With only 4KB page size support, 512 translations would be required. This would improve TLB performance by improving the TLB hit rate.

Support for multiple page sizes is good if the program allocates both small and large chunks of contiguous memory.

(ii) If a program exclusively allocates small chunks of contiguous memory, only 4KB pages be used. This would render the 4 1GB and 16 2MB TLB entries useless, and reduce the usable TLB capacity. The TLB will, thus, capture fewer 4KB page translations than it would if the whole TLB were dedicated for 4KB entries, and have a lower hit rate. The 1GB and 2MB cannot be re-purposed to store translations for 4KB pages.

Different page sizes require different number of bits to store the translation (physical page number/tag). This may create challenges from an implementation perspective if we would like to make all entries available for translations regardless of page size, because of differences in tag size required for various page sizes. Alternatively, we could partition our TLB, so different entries can only store translations for a given page size, but that would waste TLB capacity if not all three page sizes are used by a given application.