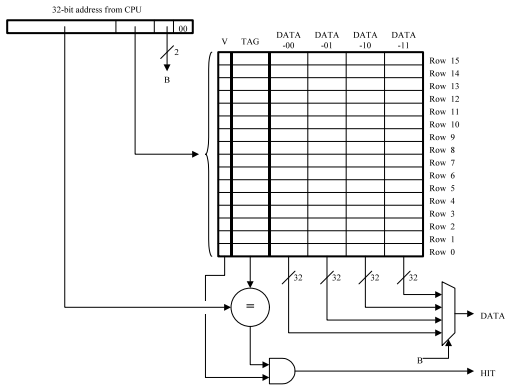
The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 411 Computer Organization
Spring 2012

Solutions Problem Set #6

Problem 1. Cache accounting

The diagram below illustrates a blocked, direct-mapped cache for a computer that uses 32-bit data words and 32-bit byte addresses. The block size is 4 words.



(A) What are the maximum number words of data from main memory that can be stored in the cache at any one time? How many bits of the address are used to select which line of the cache is accessed? How many bits wide is the tag field?

A total of 4 × 16 = 64 words can be present in the cache at any given time. 4 bits of the address <8:4> are used to select the cache line, and the tag size is 24 bits.

(B) Briefly explain the purpose of the one-bit V field associated with each cache line.

The V bit indicates that the corresponding cache line contains valid data.

(C) Assume that memory location 0x3328C was present in the cache. Using the row and column labels from the figure, in what cache location(s) could we find the data from that memory location? What would the value(s) of the tag field(s) have to be for the cache row(s) in which the data appears?

Location 0x3328C would be found in block 3 (Data -11) of line 8, and the tag field would contain 0x000332.

(D) Can data from locations 0x12368 and 0x322F68 be present in the cache at the same time? What about data from locations 0x2536038 and 0x10F4? Explain.

The locations 0x12368 and 0x322f68 map to the same cache line, 6, and therefore could not be in the cache simultaneously. Locations 0x2536038 and 0x000010f4 could be in cache simultaneously since they map to different lines, (3 and F respectively)

(E) When an access causes a cache miss, how many words need to be fetched from memory to fill the appropriate cache location(s) and satisfy the request?

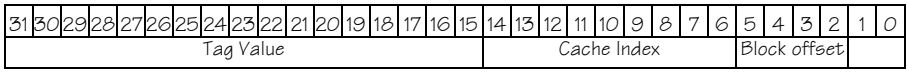
On a cache miss 4 sequential words need to be fetched to fill the line.

Problem 2. Where are the Bits?

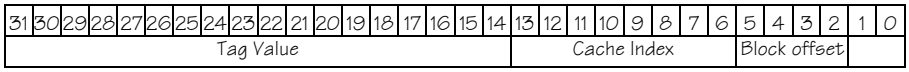
AMD's Sempron processor is advertised as having a total on-chip cache size of 384 Kbytes. It is broken down into 128Kbytes of Level 1 (L1) cache and 256Kbytes of Level 2 (L2) cache. The first level cache is composed of separate instruction and data caches, each with 64 Kbytes. All caches use a common block size of 64 bytes (16, 32-bit words). The L1 instruction and data caches are both 2-way set associative, and the unified (used for both instructions and data) L2 cache is 16-way associative. Both L1 caches employ an LRU replacement strategy, while the L2 cache uses RANDOM replacement. The L1 data cache uses one dirty bit for each group of 4 words in a line (4 for each line). None of the caches, employ a valid bit, instead a fault handler is responsible for executing reads from a reserved region of memory to initialize the cache's contents.

(A) Determine which bits of a 32-bit address are used as tags, cache indices, and block selection for each of the three caches.

The block size for all caches is 64-bytes or 16 words thus the lower 4 + 2 address bits are used as block offsets. Each L1 caches is 64 Kbytes divided over a 2-way sets, making only 32K bytes addressable. This means that 15 – (6) = 9 bits are needed as the cache index, leaving the upper 17 bits as a cache tag.



The L2 cache is 256 Kbytes divided over 16-way sets, making 16 Kbytes addressable, which leads once again to a 14 – 6 = 8 bit index and 18 bit tag.



(B) Compute how many actual bits are needed each cache, accounting for tags, data, replacement, dirty, and any other overhead.

The L1 instruction cache has:
512 * (1 LRU bit + 2 * (17 bits/tag + (64 * 8) bits/block)) = 542208 bits
The L1 data cache has:
512 * (1 LRU bit + 2 * (4 dirty bits + 17 bits/tag + (64 * 8) bits/block)) = 546304 bits
The L2 cache has:
256 * (16 * (4 dirty bits + 18 bits/tag + (64 * 8) bits/block)) = 2187264 bits

Both L1 caches have a 3-cycle access latency, the L2 cache requires 5 additional cycles (after an L1 miss), and main memory requires 20-cycles to load a line. A popular benchmark suite reports that this cache design achieves an instruction cache hit rate of 95%, and a data cache hit rate of 80%. A hit rate of 98% has been reported for the L2 cache for the same benchmarks.

(C) What is the average effective access time for instruction fetches in this cache architecture (in cycles)? What is the average effective access time for data fetches (loads and stores) in this cache architecture (in cycles)? If you assume that about 20% of executed instructions access memory, what is the overall effective access time?

For instruction fetches,
0.95*3 + (1-0.95)*(0.98*(3+5) + (1-0.98)*(3+5+20)) = 3.27 cycles

For data access,
0.80*3 + (1-0.80)*(0.98*(3+5) + (1-0.98)*(3+5+20)) = 4.08 cycles

Overall effective access time,
Instruction*100/120 + data access * 20/120 = 3.405 cycles

(D) AMD also sells a version of the chip with a L2 cache that is half the size (128 Kbytes). In that chip the L2 hit rate is only 92% for the same benchmark, what is its overall effective access time in cycles?

For instruction fetches,
0.95*3 + (1-0.95)*(0.92*(3+5) + (1-0.92)*(3+5+20)) = 3.33 cycles

For data access,
0.80*3 + (1-0.80)*(0.92*(3+5) + (1-0.92)*(3+5+20)) = 4.32 cycles

Overall effective access time,
Instruction*100/120 + data access * 20/120 = 3.495 cycles

(E) Assuming that your on-chip memory budget is fixed at 384 Kbytes, what might you suggest to improve the performance of this cache architecture?

The performance is largely dictated by the instruction access time, as seen by its contribution to the average access time. Therefore, the first place to start is to improve the L1 instruction cache. Since the total cache size is fixed, to increase the hit ratio, we need to either increase associativity (we'd need to be careful because doing so might require more LRU bits) or increase the blocksize (which would save us of tag bits). Increasing the blocksize for the instruction cache is particularly attractive, since loading large blocks effectively prefetches instructions that are likely to be needed soon. The best choice is probably a combination of these two modifications.

One last note, notice that the I-cache hit rate is already quite high, meaning that we may have already have reached a point of diminishing returns. Thus, making a small change to the D-cache might, in the end, provide a greater overall improvement. The only way to really know is to apply analysis based on instruction traces from the same benchmarks or applications.

Problem 3. Trip Around the Old Block

Lee Hart is assigned the task of designing the cache architecture for an upcoming processor. He is given the following design constraints. Cache accesses, for both hits and misses, require 1 clock cycle. On cache misses, an entire cache block (multiple words) is fetched from main memory. Main memory accesses are not initiated until after the cache is inspected. A main-memory access consists of 2 cycles to send the address, 3 cycles of idle time for main-memory access, and then 1 cycle to transfer word to the cache. Moreover, the processor will stall until the last word of the block has arrived. Thus, if the block size is B words (32 bits), a cache miss will cost 1 + 2 + 3 + B cycles. The following table gives the average cache miss rates of a 1 Mbyte cache for various block sizes:

Block size (B)	*Miss ratio (m), %*
1	3.2
4	1.6
16	0.8
64	0.4
256	0.3

(A) Write an expression for the average memory access time for a 1-Mbyte cache and a B-word block size (in terms of the miss ratio *m* and B). (5 pts)

(1-m)*1 + m*(1+2+3+B) = 1 + 5m + Bm

(B) What block size yields the best average memory access time? (5 pts)

4

(C) If main memory bus width is expanded to 128 bits (4 words), reducing the number of cycles for accessing the block proportionately, what is the optimal block size? (5 pts)

The expression becomes 1 + 5m + Bm/4, and the best blocksize becomes 16.

(D) Suppose that expanding the memory bus to 128 bits increases the main memory access latency by 6 clocks (for a total of 12 cycles of overhead before the first access); what then is the optimal block size? (5 pts)

The expression becomes (1-m)*1 + m*(1+2+9+B/4), and the best blocksize becomes 64.

Problem 4. Approximate LRU Replacement

Immediately after the cache-replacement-algorithm lecture in Comp 411, Lori Acan has an epiphany. It dawns on her that it is possible to approximate LRU replacement in a 4-way set with only 3 bits per set, 2 less than an exact LRU implementation and only 1 bit more than the FIFO algorithm. Her scheme works as follows. She partitions the 4-way set into two halves and treats each half as a 2-way set, thus, allotting one replacement bit for computing the LRU line in each half. This requires 2 bits, one for each half. The third bit is used to keep track of the most recently used half of the cache. On a cache miss, the least recently used line in the least recently used half of the cache is replaced.

(A) Simulate the behavior of this replacement algorithm on a 4-word fully associate cache using the instruction sequence given in lecture and repeated below where $t3 = 0x4000. Assume the initial values of the replacement bits are 0-00. Estimate the miss rate for this code sequence in the steady state while noting of the values of the 3 replacement bits on each cache access.

```
Address     Instruction
  400:      lw   $t0,0($t3)
  404:      addi $t3,$t3,4
  408:      bne  $t0,$0,400
```

Addr	Line #	Miss?	Replacement bits
100	2	M	1,01
1000	0	M	0,11
101	3	M	1,10
102	1	M	0,00
100	2		1,01
1001	0	M	0,11
101	3		1,10
102	1		0,00
100	2		1,01
1002	0	M	0,11
101	3		1,10
102	1		0,00
100	2		1,01
1003	0	M	0,11
101	3		1,10
102	1		0,00

The miss ratio is ¼.

(B) Explain how Lori's approximate LRU algorithm can be used for any N-way set where $N=2^k$. How many bits are required per set in these cases?

Half the lines step by step until there are only two lines in each small group. The first bit is used to indicate which half is most recently used, the second is used to indicate which ¼ among the half is used most recently, and so on. The last bit is used to indicate which line is least recently used in the pair of lines. There are totally log(N) levels, the number of bits needed is

$$1 + 2 + 4 + \ldots + N/2 = N-1$$

(C) Give a memory reference string for which Lori's 4-way LRU replacement algorithm differs from an exact LRU replacement.

If the replacement bits are (0,00), line pair (2, 3) is accessed less recently than pair (0, 1), and 2 is accessed before 3 while 0 is accessed before 1. There are three possible access pattern which are (2, 3, 0, 1), (2, 0, 3, 1), and (0, 2, 3, 1). If the true access order is (0, 2, 3, 1), then the second least recently accessed line 2 is more likely to be replaced than line 0.

Addr	Line #	Miss?	Replacement bits
100	2	M	1,01
101	0	M	0,11
102	3	M	1,10
103	1	M	0,00
101	0		0,11
102	3		1,10
104	1	M	0,00

When the last miss happens, line 2 is least recently used, but in Lori's algorithm, the second least used line 1 will be replaced.

(D) In the worse case, what will be the highest element in an ordered list of recent accesses that Lori's approximate LRU replacement algorithm would throw out?

The case given above is the worst case. The third least recently used value will either be the most recently used element in the "other" least recently used half, or it will be the least recently used element in the current, most recently used "half". In neither case can it be thrown out.