

# Homework 4 Solutions

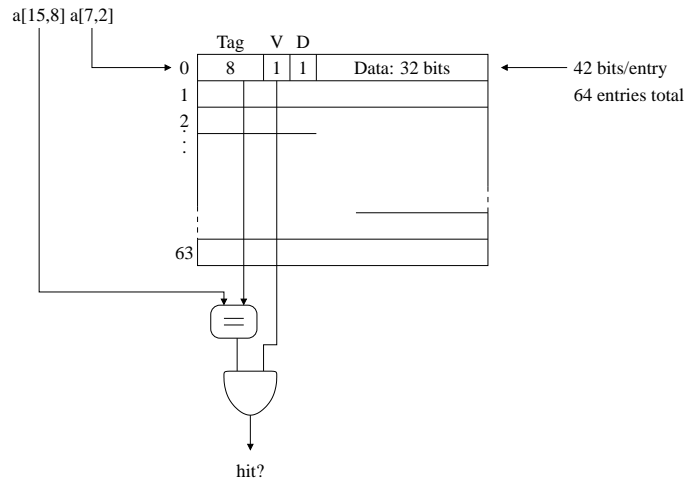
Prof. Loh

CS3220 - Processor Design - Spring 2005

1. **Caches** In the following parts, all caches will have a capacity of 256 bytes, and 4-byte (32-bit) words. Assume an address of 16-bits ( $A[0..15]$ ). You do not need to draw each and every single entry in the cache, but instead, use ellipses (“...”) when appropriate to reduce the amount of tedious drawing that would otherwise be required.

(a) Design a direct mapped cache (with the specifications given above: 256 byte capacity, 4 byte words), where each cache line stores only a single word. Be sure to label which bits are used to index into the cache, to compare to the tag, etc. How large is the cache in bits (including the bits for the tag, a valid bit, and a dirty bit)? If a cache hit takes 1ns, and a cache miss takes 10ns to service, how long will it take to access the following sequence of addresses, assuming that all entries of the cache are initially invalid?

4, 8, 12, 16, 4, 4, 20, 4, 16, 68, 132, 72, 8, 12, 68, 4, 100, 52, 20, 4, 8, 12, 16, 4, 4, 256, 4, 16, 516

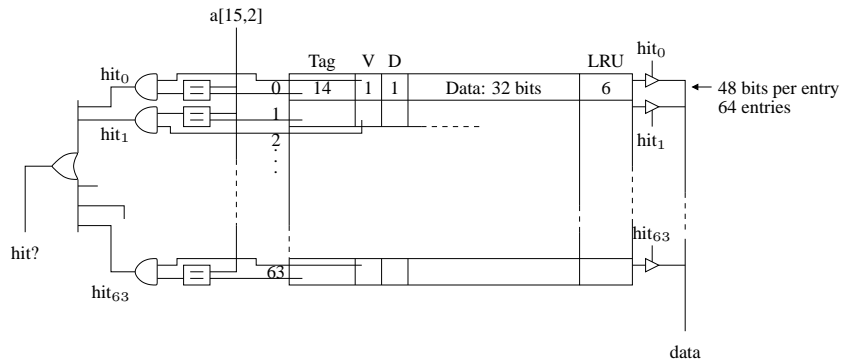


With 4-byte words, we divide the address by four to get a word index. We then take the word index modulo 64 (since there are 64 rows in the cache) to get a row index.

4→1 (M), 8→2 (M), 12→3 (M), 16→4 (M), 4→1 (H), 4→1 (H), 20→5 (M), 4→1 (H), 16→4 (H), 68→17 (M), 132→33 (M), 72→18 (M), 8→2 (H), 12→3 (H), 68→17 (H), 4→1 (H), 100→25 (M), 52→13 (M), 20→5 (H), 4→1 (H), 8→2 (H), 12→3 (H), 16→4 (H), 4→1 (H), 4→1 (H), 256→0 (M), 4→1 (H), 16→4 (H), 516→1 (M)

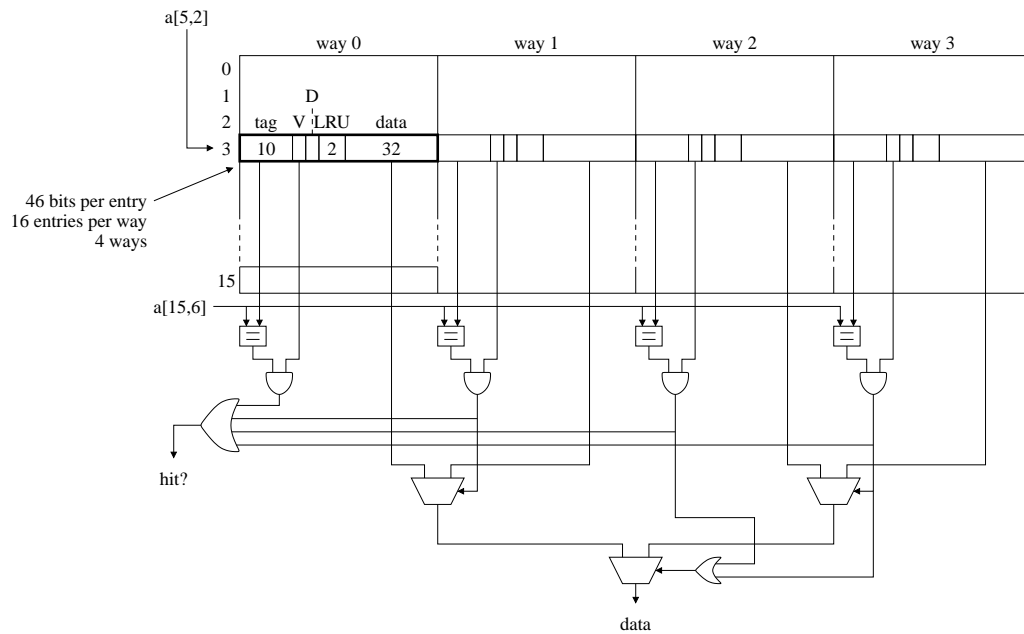
Total: 12 misses, 17 hits, 137 ns.

(b) Repeat for a fully associative cache (remember to include LRU bits for the non-direct-mapped caches!).



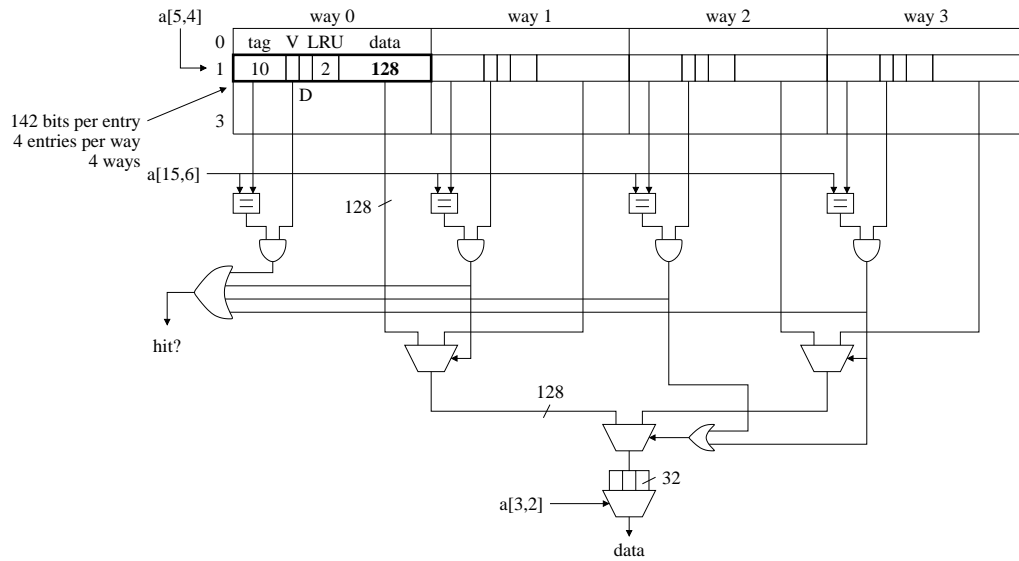
Hit/miss pattern is the same: 137ns.

(c) Repeat for a 4-way set-associative cache.



Hit/miss pattern is the same: 137ns.

(d) Repeat for a 4-way set-associative cache with a cache line size of four words. The total capacity should still be 256 bytes.



With 16-byte entries, we divide the address by sixteen to get an entry index. We then take the entry index modulo 4 (since there are 4 rows in the cache) to get a row index.

4→0 (M), 8→0 (H), 12→0 (H), 16→1 (M), 4→0 (H), 4→0 (H), 20→1 (H), 4→0 (H), 16→1 (H), 68→0 (M),  
 132→0 (M), 72→0 (H), 8→0 (H), 12→0 (H), 68→0 (H), 4→0 (H), 100→2 (M), 52→3 (M), 20→1 (H),  
 4→0 (H), 8→0 (H), 12→0 (H), 16→1 (H), 4→0 (H), 4→0 (H), 256→0 (M), 4→0 (H), 16→1 (H), 516→0 (M)

Total: 8 misses, 21 hits, 101ns.

2. **Pipeline Control** Refer to the online notes on pipeline control for this problem. Assuming we have the pipeline organization shown on page 2 of the notes, show the cycle-by-cycle state of the scoreboard to execute the following instructions:

- A. ADD R1 R3
- B. SUB R2 R4
- C. XOR R1 R2
- D. MUL R5 \*R2
- E. ADD R4 R5
- F. ADD R1 R4

Assume all “regular” arithmetic operations take one cycle to execute. Assume multiplication takes 2 cycles. Assume that all memory accesses hit in the cache and take one cycle. Complete all cycles up to and including when instruction F reaches the Load/Exec1 stage. Also, draw in the bypass paths used to route the register values from producers to consumers (in the mini-pipeline diagrams on the right on the next page). This should look similar to the examples from the notes. Use the following template:

