ES5 EE3400 23/04/2025

1. Consider a processor with 32-bit virtual addresses, 4K-byte pages, and 36-bit physical addresses. Assume memory is byte-addressable (i.e., the 32-bit virtual address specifies a byte in memory). Other facts about the system are:
A. LI instruction cache: 64K bytes, 128-byte blocks, four-way set- associative, indexed and tagged with virtual address.
B. LI data cache: 32K bytes, 64-byte blocks, two-way set-associative, indexed and tagged with physical address, writeback.
C. Four-way set-associative TLB with 128 entries in all. Assume the TLB keeps a dirty bit, a reference bit, and three permission bits (read, write, execute) for each entry.

Specify the number of offset, index, and tag bits for each of these structures (i-cache, d-cache and TLB). Also, compute the total size in number of bit cells for each of the tag and data arrays.

2. A CPU accesses the following physical memory addresses in this sequence: 22, 26, 22, 26, 16, 3, 16, 18, 16. Show the tag, index, hit/miss, valid bit for this sequence of access. Assume only 32 locations in main memory and cache to be 2-way set associative, having 4 cache sets with 2 lines in each set.

3. Consider a cache with a total data size of 256 bytes. The word size is 4 bytes, and the cache line size is 16 bytes. Show the values in the cache and tag bits after each of the following memory access operations for the following two cache organizations: direct mapped and two-way associative. Also indicate whether the access was a hit or a miss. Justify. The addresses are in hexadecimal representation. Use LRU (least recently used) replacement algorithm wherever needed.

1. Read 0010
2. Read 001C
3. Read 0018
4. Write 0010
5. Read 0484
6. Read 051C
7. Read 001C
8. Read 0210
9. Read 051C

4. Assuming a virtually-addressed cache, compute the overall miss rate (number of misses divided by number of references). Assume that all variables except array locations reside in registers and that arrays A, B, and C are placed consecutively in memory. Assume that the generated code for the loop body first loads from
B, then from C, and finally stores to A.

```
double A[1024], B[1024], C[1024];
f o r (int i = 0 ; i <1000; i+=2) {
A [ i ] = 2.0 * B [ i ] + C [ i + 1 ] ;
}
```

Do this when:

A. when cache is fully-associative with infinite capacity and having line size of 64B.

B. when cache of capacity 8K-byte and having the same line size.

5.Transposing the rows and columns of a matrix has a reference pattern of data which is both row-wise and column-wise. For example, consider the following transpose routine:

```
Typedef int array[2][2];
void transpose(array dst, array src) {
Int i, j;
for (i=0; i<2; i++) for (j=0; j<2; j++) { dst[j][i] = src[i][j]; }
return;
}
```

Assume

• sizeof(int) = 4

• The src array starts at address 0 and the dst array starts at address 16 (decimal).

• There is a single L1 data cache that is direct-mapped, write-through with a cache line size of 8 bytes.

• The cache has a total size of 16 data bytes and the cache is initially empty.

• Accesses to the src and dst arrays are the only sources of read and write misses, respectively.

Then, for each row and col, indicate whether the access to src[row][col] and dst[row][col] is a hit (h) or a miss (m). For example, reading src[0][0] is a miss and writing dst[0][0] is also a miss.

2. Repeat the problem for a cache with 32 data bytes.