Solutions to Problems Marked with a * in
Logic and Computer Design Fundamentals, 4th Edition

Chapter 13

13-3.*

Since the lines are 32 bytes, 5 bits are used to address bytes in the lines.

Since there are 1K bytes, there are $1024/32 = 2^5$ cache lines.

a) Index = 5 Bits,

b) Tag = 32 – 5 – 5 = 22 Bits

c) $32 \times (32 \times 8 + 22 + 1) = 8928$ bits

13-5.*

a) See Instruction and Data Caches section on page 635 of the text.

b) See Write Methods section on page 631 of the text.

13-7. *

000000 00 00 (i0)	000001 00 00 (i4)	000001 10 00 (i6)	000010 10 00 (i10)
000000 01 00 (i1)	000011 00 00 (d)	00001 11 00 (i7)	000011 10 00 (d)
000000 10 00 (i2)	000001 01 00 (i5)	000010 00 00 (i8)	000010 11 00 (i11)
000000 11 00 (i3)	000011 01 00 (d)	000010 01 00 (i9)	000011 11 00 (d)

Addresses of instructions (i) and Data (d) in sequence down and then to the right with the instructions in a loop with instruction i0 following i11. For the split cache, the hit - miss pattern for instructions is (assuming the cache initially empty and LRU replacement) M, M, M, M, M, M, M, M, M, M, M, M, M, ... since there are only eight locations available for instructions. For the unified cache, the hit-miss pattern for instructions with the same assumptions is M, M, M, M, M, M, M, M, M, M, M, M, H, H, H, ... since there are 12 locations indexed appropriately for instructions and four indexed appropriately for data.

13-10.*

a) Effective Access Time = 0.91 * 4ns + 0.09 * 40 ns = 7.24 ns

b) Effective Access Time = 0.82 * 4ns + 0.18* 40 ns = 10.48 ns

c) Effective Access Time = 0.96 * 4ns + 0.04 * 40 ns = 5.44 ns

13-13.*

a) Each page table handles 512 pages assuming 64-bit words. There are 4263 pages which requires 4263/512 8.33 page tables. So 9 page tables are needed.

b) 9 directory entries are needed, requiring 1 directory page.

c) 4263 - 8*512 = 167 entries in the last page table.

13-17.*

In section 14-3, it is mentioned that write-through in caches can slow down processing, but this can be avoided by using write buffering. When virtual memory does a write to the secondary device, the amount of data being written is typically very large and the device very slow. These two factors generally make it impossible to do write-through with virtual memory. Either the slow down is prohibitively large, or the buffering cost is just too high.