CHAPTER 12

12-1.

	Binary	a	b	c
54	0010 1 01 00	M	M	M
58	0010 1 10 00	M	M	M
104	1000 0 01 00	M	M	M
5C	0010 1 11 00	M	M	M
108	1000 0 10 00	M	M	M
60	0011 0 00 00	M	M	M
F0	0111 1 00 00	M	M	M
64	0011 0 01 00	M	M	M
54	0010 1 01 00	H	H	M
58	0010 1 10 00	H	H	H
10C	1000 0 11 00	M	M	M
5C	0010 1 11 00	H	H	H
110	1000 1 00 00	M	M	M
60	0011 0 00 00	H	H	M
F0	0111 1 00 00	M	H	M
64	0011 0 01 00	H	H	H

12-2.

	Binary	a	b	c
20	0010 0000	M	M	M
04	0000 0100	M	M	M
28	0010 1000	M	M	M
60	0110 0000	M	M	M
20	0010 0000	M	H	H
04	0000 0100	H	H	H
28	0010 1000	H	H	H
4C	0001 1100	M	M	M
10	0001 0000	M	M	M
6C	0110 1100	M	M	M
70	0111 0000	M	M	M
10	0001 0000	M	H	H
60	0110 0000	M	H	M
70	0111 0000	M	H	M

12-3*.

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

12-4.

(a) Number of rows = 1M/(4 bytes/line × 4 words/line × 2 lines/set) = 2^{15} = 32,768
The number of bits for addressing the word and byte within a line is 4 bits.
Index = 15 bits Tag = 32 – 15 – 4 = 13 bits
(b) (Assumes main memory address to bytes, not words)
0C00
4AC8
2CF0
4CF0
2CF0
(c) Yes. Although 7142CF0F and F83ACF04 have the same index, the cache is two-way set associative, so both of them can fit in the cache simultaneously.

12-5.*

a) See Instruction and Data Caches section on page 636 of the text.
b) See Write Methods section on page 633 of the text.

12-6.

Any sequence of instructions and data that share the same indices will cause a unified cache to work poorly and a separate instruction cache and data cache to work well. Recall that the index for direct mapping are the bits from location 2 through 4.

12-7.*

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.

12-8.

The use of write-allocate with a write-through cache defeats the purpose of write-allocate. The idea behind write-allocate is to load a cache line when a write miss occurs to a word in the line, thus if future writes occur to the same line, no writes to main memory will be required. If write-through is implemented in the cache, a write to main memory is performed anyway, thus write-allocate is a waste of design effort in these caches.

12-9

a) Number of sets = 1K/(4×4) = 64
b) Index = 6 bits, Tag = 32 – 6 -2 (word) -2 (byte) = 22 bits
c) Number of sets = 1K/$(4 \times 4 \times 4)$ = 16, Index = 4 bits, Tag= 32 – 4 -2 (word) -2 (byte) = 24 bits
d) Number of sets = 1K/(2×4)= 128, Index = 7 bits, Tag = 32 – 7 -1 (word) -2 (byte) = 22 bits

12-10.

a) $2^{64}/2^{3} = 2^{61}$
b) Byte = 5, Index = 14, Tag = 45

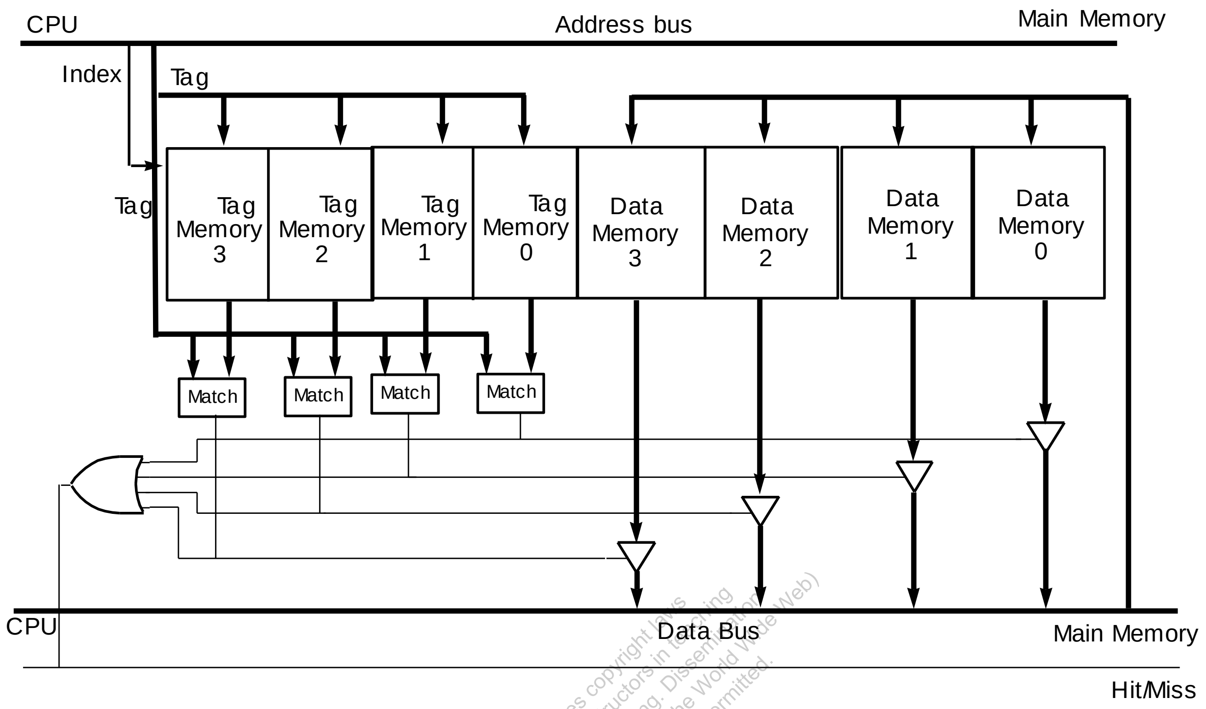
12-11.*

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

12-12.

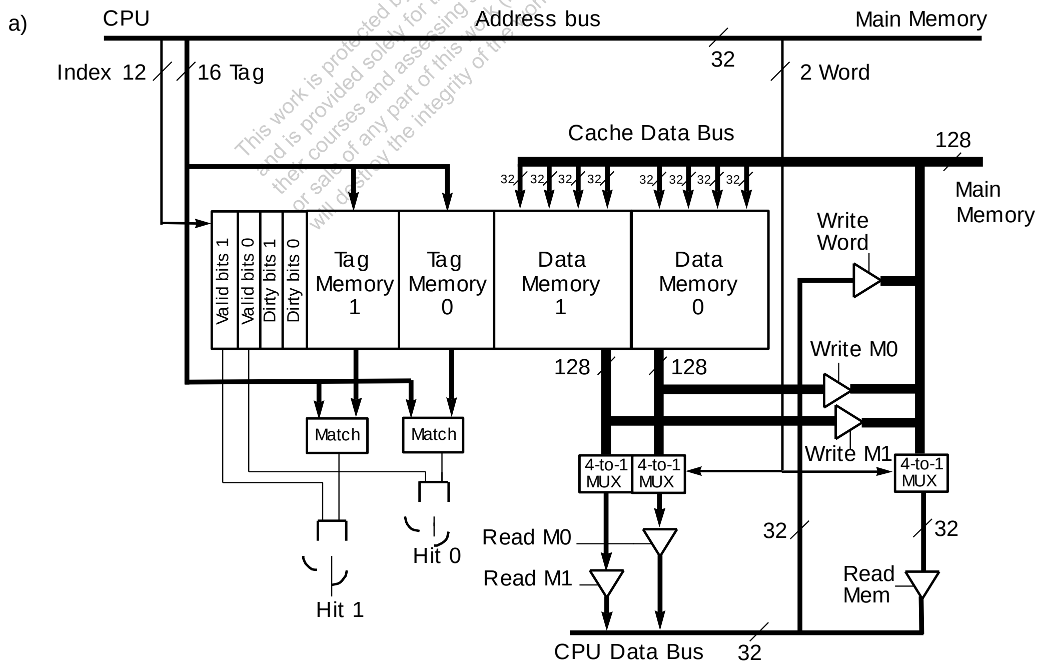
a) Effective Access Time = 0.91 * 1ns + 0.09 * 20 ns = 2.71 ns
b) Effective Access Time = 0.82 * 1ns + 0.18* 20 ns = 4.42 ns
c) Effective Access Time = 0.96 * 1ns + 0.04 * 20 ns = 1.76 ns

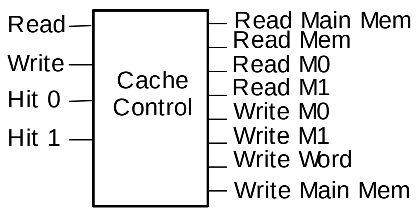
12-13.


CPU
Address bus
Main Memory
Index
Tag
Tag
Tag Memory 3
Tag Memory 2
Tag Memory 1
Tag Memory 0
Data Memory 3
Data Memory 2
Data Memory 1
Data Memory 0
Match
Match
Match
Match
CPU
Data Bus
Main Memory
Hit/Miss

12-14.+

a)


CPU
Address bus
Main Memory
32
Index 12
16 Tag
2 Word
Cache Data Bus
128
32
Main Memory
Valid bits 1
Valid bits 0
Dirty bits 1
Dirty bits 0
Tag Memory 1
Tag Memory 0
Data Memory 1
Data Memory 0
Write Word
128
128
Write M0
Write M1
Match
Match
4-to-1 MUX
4-to-1 MUX
4-to-1 MUX
Hit 0
Hit 1
Read M0
Read M1
32
32
Read Mem
CPU Data Bus
32



b) Read Miss:

at least one valid bit = 0: Read data from memory into one of the unused lines, set valid = 1
both valid bits = 1, both dirty = 0: Randomly select one of the lines for replacement.
both valid bits = 1, one dirty bit = 1: Replace the line that has dirty equal to 0.
both valid bits = 1, both dirty bits = 1: Randomly select one of the sets for replacement, write it back to memory and replace it with the new line from memory. Set dirty = 0.

Write Miss:

All cases are the same as the read misses with the exception that the new value is then writen to the cache and the dirty bit is set for all cases.

12-15.*

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.

12-16.

a) With a page size of 4KB, then the page offset is 12 bits
b) With virtual addresses of 64 bits, then there 2^{64}/4K entries in the page table, = 2^{52} entries.
c) Addressing 1GB of physical memory requires 30 bits for the physical address. So the physical page frame number requires 30-12 = 18 bits.
d) The virtual page number requires 64-12 = 52 bits.
e) With a page size of 16KB instead of 4KB:
The page offset is 14 bits.
There are 2^{64}/16K = 2^{50} page table entries.
The physical page frame number requires 30 - 14 = 16 bits.
The virtual page number requires 64 - 14 = 50 bits.

12-17.

(a) Miss - Address not in cache
(b) Miss - Address in cache: E03B32, but Valid = 0
(c) Miss - Address not in cache
(d) Miss - Address in cache: 657777, but Valid = 0

12-18.

32-bit word implies 4 bytes/word
(a) 4K byte pages implies 12 bit offset which implies 20 bit tags. With 32 entries, associative memory = 32 × 20 = 320 bits.
(b) $2^{28} \le 384\text{ MB} \le 2^{29}$, giving a physical address of 29 bits. With a 12 bit offset, the physical page address is 17 bits. 32 × (3 + 17) = 640 bits.

12-19.

Directory pages		= 4
Pages Program 1	3224/1024	= 4

Pages Program 2	5670/1024	= 6
Pages Program 3	1205/1024	= 2
Pages Program 4	2069/1024	= 3
Total		= 19 pages × 4096 = 77,824 bytes = 76 KB

12-20.*

In section 12-3, 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 buff-ering cost is just too high.

12-21.

If locality of reference did not exist, no benefits would be gained by either virtual memories or caches. Without locality of reference, accesses to both virtual memory and the cache would cause a miss with much greater likelihood. Since any miss incurs more time than direct accesses to cache or memory, system performance would not be significantly improved.