4.5 One field identifies a unique word or byte within a block of main memory. The remaining two fields specify one of the blocks of main memory. These two fields are a line field, which identifies one of the lines of the cache, and a tag field, which identifies one of the blocks that can fit into that line.

4.6 A tag field uniquely identifies a block of main memory. A word field identifies a unique word or byte within a block of main memory.

4.7 One field identifies a unique word or byte within a block of main memory. The remaining two fields specify one of the blocks of main memory. These two fields are a set field, which identifies one of the sets of the cache, and a tag field, which identifies one of the blocks that can fit into that set.

4.1 The cache is divided into 64 sets of 2 lines each. Therefore, 6 bits are needed to identify the set number. Main memory consists of 8K = 2^{13} blocks. Therefore, the set plus tag lengths must be 13 bits and therefore the tag length is 7 bits. Each block contains 256 words. Therefore, 8 bits are needed to specify the word.

	TAG	SET	WORD
Main memory address =	7	6	8

4.2 There are a total of 4 kbytes/32 bytes = 128 lines in the cache. Thus the cache consists of 32 sets of 4 lines each. Therefore 5 bits are needed to identify the set number. For the 32-Mbyte main memory, a 25-bit address is needed. Main memory consists of 32-Mbyte/32 bytes = 2^{20} blocks. Therefore, the set plus tag lengths must be 20 bits, so the tag length is 15 bits and the word field length is 5 bits.

	TAG	SET	WORD
Main memory address =	15	5	5

4.3

Address	111111	666666	BBBBBB
a. Tag/Line/Word	11/444/1	66/1999/2	BB/2EEE/3
b. Tag /Word	44444/1	199999/2	2EEEEE/3
c. Tag/Set/Word	22/444/1	CC/1999/2	177/EEE/3

4.4 a. Address length: 24; number of addressable units: 2^{24}; block size: 4; number of blocks in main memory: 2^{22}; number of lines in cache: 2^{14}; size of tag: 8.

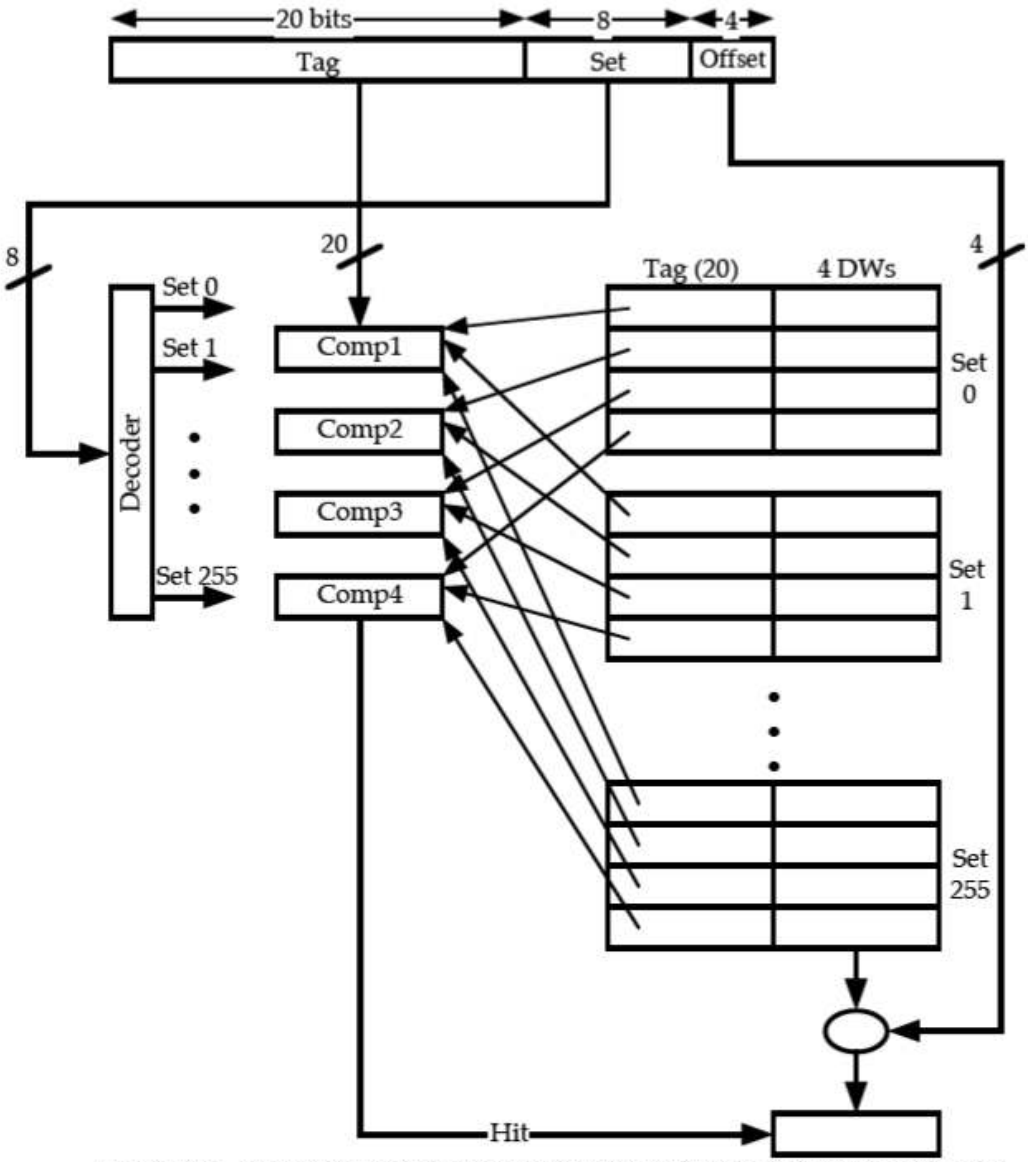
b. Address length: 24; number of addressable units: 2^{24}; block size: 4; number of blocks in main memory: 2^{22}; number of lines in cache: 4000 hex; size of tag: 22.

c. Address length: 24; number of addressable units: 2^{24}; block size: 4; number of blocks in main memory: 2^{22}; number of lines in set: 2; number of sets: 2^{13}; number of lines in cache: 2^{14}; size of tag: 9.

4.5 Block frame size = 16 bytes = 4 doublewords

Number of block frames in cache = $\frac{16\,\text{KBytes}}{16\,\text{Bytes}} = 1024$

Number of sets = $\frac{\text{Number of block frames}}{\text{Associativity}} = \frac{1024}{4} = 256\ \text{sets}$



Example: doubleword from location ABCDE8F8 is mapped onto: set 143, any line, doubleword 2:

4.12 a. Because the block size is 16 bytes and the word size is 1 byte, this means there are 16 words per block. We will need 4 bits to indicate which word we want out of a block. Each cache line/slot matches a memory block. That means each cache slot contains 16 bytes. If the cache is 64Kbytes then 64Kbytes/16 = 4096 cache slots. To address these 4096 cache slots, we need 12 bits (212 = 4096). Consequently, given a 20 bit (1 MByte) main memory address:

Bits 0-3 indicate the word offset (4 bits)
Bits 4-15 indicate the cache slot (12 bits)
Bits 16-19 indicate the tag (remaining bits)

F0010 = 1111 0000 0000 0001 0000
Word offset = 0000 = 0
Slot = 0000 0000 0001 = 001
Tag = 1111 = F

01234 = 0000 0001 0010 0011 0100
Word offset = 0100 = 4
Slot = 0001 0010 0011 = 123
Tag = 0000 = 0

CABBE = 1100 1010 1011 1011 1110
Word offset = 1110 = E
Slot = 1010 1011 1011 = ABB
Tag = 1100 = C

b. We need to pick any address where the slot is the same, but the tag (and optionally, the word offset) is different. Here are two examples where the slot is 1111 1111 1111

Address 1:
Word offset = 1111
Slot = 1111 1111 1111
Tag = 0000
Address = 0FFFF

Address 2:
Word offset = 0001
Slot = 1111 1111 1111
Tag = 0011
Address = 3FFF1

c. With a fully associative cache, the cache is split up into a TAG and a WORDOFFSET field. We no longer need to identify which slot a memory block might map to, because a block can be in any slot and we will search each cache slot in parallel. The word-offset must be 4 bits to address each individual word in the 16-word block. This leaves 16 bits leftover for the tag.

F0010
Word offset = 0h
Tag = F001h

CABBE
Word offset = Eh
Tag = CABBh

d. As computed in part a, we have 4096 cache slots. If we implement a two -way set associative cache, then it means that we put two cache slots into one set. Our cache now holds 4096/2 = 2048 sets, where each set has two slots. To address these 2048 sets we need 11 bits (211 = 2048). Once we address a set, we will simultaneously search both cache slots to see if one has a tag that matches the target. Our 20-bit address is now broken up as follows:

Bits 0-3 indicate the word offset
Bits 4-14 indicate the cache set
Bits 15-20 indicate the tag

F0010 = 1111 0000 0000 0001 0000
Word offset = 0000 = 0
Cache Set = 000 0000 0001 = 001
Tag = 11110 = 1 1110 = 1E

CABBE = 1100 1010 1011 1011 1110
Word offset = 1110 = E
Cache Set = 010 1011 1011 = 2BB
Tag = 11001 = 1 1001 = 19

4.13 Associate a 2-bit counter with each of the four blocks in a set. Initially, arbitrarily set the four values to 0, 1, 2, and 3 respectively. When a hit occurs, the counter of the block that is referenced is set to 0. The other counters in the set with values originally lower than the referenced counter are incremented by 1; the remaining counters are unchanged. When a miss occurs, the block in the set whose counter value is 3 is replaced and its counter set to 0. All other counters in the set are incremented by 1.

4.21 a. First, 5 ns are needed to determine that a cache miss occurs. Then, the required line is read into the cache. Then an additional 5 ns are needed to read the requested word.

$$T_{miss} = 5 + 100 + (3)(10) + 5 = 140 \text{ ns}$$

b. The value T_{miss} from part (a) is equivalent to the quantity $(T_1 + T_2)$ in Equation (4.1). Under the initial conditions, using Equation (4.1), the average access time is:

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2)$$

Since the miss rate is 11%, H = 0.89. Thus,

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2)$$
$$= (0.89)(5) + (0.11)(140) = 19.85 \text{ ns}$$

Under the revised scheme, we have:

$$T_{miss} = 5 + 100 + (7)(10) + 5 = 180 \text{ ns}$$

and

$$T_s = H \times T_1 + (1 - H) \times (T_1 + T_2)$$
$$= (0.93)(5) + (0.07)(180) = 17.25 \text{ ns}$$

Yes, it reduces the average memory access time.

4.22 There are three cases to consider:

Location of referenced word	Probability	Total time for access in ns
In cache	0.91	9
Not in cache, but in main memory	(0.09)(0.7) = 0.063	80 + 9 = 89
Not in cache or in main memory	(0.09)(0.3) = 0.027	8ms + 80 + 9 = 8,000,089

So the average access time would be:

Avg = (0.91)(9) + (0.063)(89) + (0.027)(8000089)
= 8.19 + 5.607 + 216002.403 = 216016.2 ns