Solution of homework 7

8.10

Direct mapping with block size of one word:

15	7C, 13C
14	78, 38
13	74, 34
12	
11	AC
10	
9	
8	A0
7	
6	
5	
4	
3	38C, 8C, 18C
2	88, 388
1	84
0	

74: 01110100, A0: 10100000 , 78: 01111000, 38C: 001110001100, AC: 10101100, 84: 10000100, 88: 10001000 , 8C: 10001100, 7C: 01111100, 34: 00110100, 38: 00111000,, 13C: 000100111100, 388: 001110001000, and 18C: 000110001100

Miss rate: 11/14

Fully associative with block size of two words:

For fully associative map with block size of two words and a cache of only 16 words. There will be a total of 8 different locations to be filled. Unfortunately, once the cache is filled you will need to kick some blocks out and substitute them with new blocks. The only good news is 7C&78, 88&8C, and 38C&388 are the same block.

Miss rate: 11/14

Two-way associative block size of 2 words

74: 01110100, A0: 10100000 , 78: 01111000, 38C: 001110001100, AC: 10101100, 84: 10000100, 88: 10001000 , 8C: 10001100, 7C: 01111100, 34: 00110100, 38: 00111000,, 13C: 000100111100, 388: 001110001000, and 18C: 000110001100

3	7C&78, 38	13C
2	74	34
1	38C&388, 88&8C	AC,18C
0	A0	84

Miss rate 6/14

Direct mapping with block size of four words

74: 01110100, A0: 10100000 , 78: 01111000, 38C: 001110001100, AC: 10101100, 84: 10000100, 88: 10001000 , 8C: 10001100, 7C: 01111100, 34: 00110100, 38: 00111000,, 13C: 000100111100, 388: 001110001000, and 18C: 000110001100

3	74&78&7C,34&38,13C
2	A0&AC
1	
0	38C&388, 84&88&8C, 18C

Miss rate is 6/14

8.11

a. $2^{10}/2^3=128$

b. Independent of the pattern of memory access. Since, this is the first time we will get 100% miss rate, except if the block size is big enough to fit multiple memory accesses at the same time. Unfortunately, having a block size of 8 bytes and for the following memory accesses each one of them will be mapped into a different block as shown below.

0: 00000000, 8: 00001000, 10:00010000, 18: 00011000, 20: 00100000, 28, 00101000

c. Increasing the block size to 16 bytes help in reducing the miss rate to 50% because 0&8, 10&18, 20&28 will fit in three different blocks instead of six.

Exercise 8.13

(a)

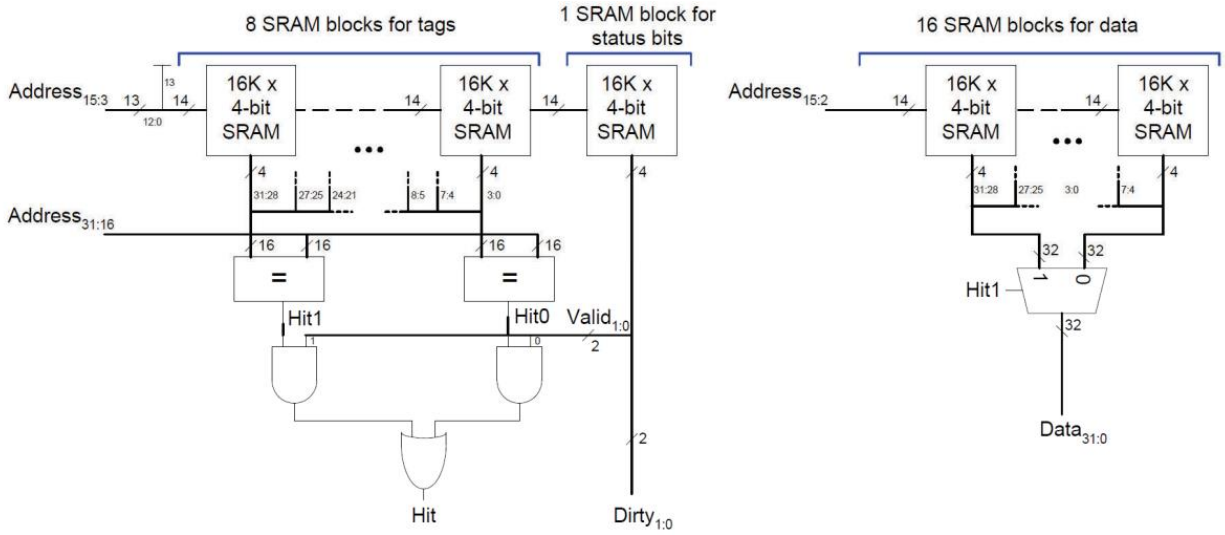
(b) Each tag is 16 bits. There are 32Kwords / (2 words / block) = 16K blocks and each block needs a tag: 16 × 16K = **256 Kbits** of tags.

(c) Each cache block requires: 2 status bits, 16 bits of tag, and 64 data bits, thus each set is 2 × 82 bits = **164 bits**.

(d) See figure below. The design must use enough RAM chips to handle both the total capacity and the number of bits that must be read on each cycle. For the data, the SRAM must provide a capacity of 128 KB and must read 64 bits per cycle (one 32-bit word from each way). Thus the design needs at least 128KB / (8KB/RAM) = 16 RAMs to hold the data and 64 bits / (4 pins/RAM) = 16 RAMs to supply the number of bits. These are equal, so the design needs exactly 16 RAMs for the data.

For the tags, the total capacity is 32 KB, from which 32 bits (two 16-bit tags) must be read each cycle. Therefore, only 4 RAMs are necessary to meet the capacity, but 8 RAMs are needed to supply 32 bits per cycle. Therefore, the design will need 8 RAMs, each of which is being used at half capacity.

With 8K sets, the status bits require another 8K × 4-bit RAM. We use a 16K × 4-bit RAM, using only half of the entries.



Bits 15:2 of the address select the word within a set and block. Bits 15-3 select the set. Bits 31:16 of the address are matched against the tags to find a hit in one (or none) of the two blocks with each set.

Exercise 8.15

(a) **FIFO:** FIFO replacement approximates LRU replacement by discarding data that has been in the cache longest (and is thus least likely to be used again). A FIFO cache can be stored as a queue, so the cache need not keep track of the least recently used way in an N-way set-associative cache. It simply loads a new cache block into the next way upon a new access. FIFO replacement doesn't work well when the least recently used data is not also the data fetched longest ago.
Random: Random replacement requires less overhead (storage and hardware to update status bits). However, a random replacement policy might randomly evict recently used data. In practice random replacement works quite well.

(b) FIFO replacement would work well for an application that accesses a first set of data, then the second set, then the first set again. It then accesses a third set of data and finally goes back to access the second set of data. In this case, FIFO would replace the first set with the third set, but LRU would replace the second set. The LRU replacement would require the cache to pull in the second set of data twice.