Problem 1:

Question 1.

With a 32-bit address, how many total Tag bits are required for a direct-mapped cache with 256 bytes of data and 16-byte blocks?

$2^4 \text{ blocks } \times (32 - 4 - 4) = 384 \text{ bits}$

Question 2.

Give the block number to map byte address $0x0018$ to, when we have a 16-block direct-mapped cache, with 16-byte blocks and Tag bits are the high order bits.

$0x0018 \Rightarrow 24$

floor $(24/16) = \text{floor (1.5) } = 1$

$1 \text{ modulo } 16 = 1$

Question 3.

Suppose we have a 2-Way Set Associative cache with 256 bytes of data, 16-byte blocks, and 32-bit physical address, instead of a direct-mapped cache. Fill in the address breakdown table for this cache with the following fields: Index, Tag, and Byte offset and their corresponding address bits.

<table>
<thead>
<tr>
<th>Address Bits</th>
<th>31</th>
<th>7</th>
<th>6</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Tag</td>
<td>Index</td>
<td>Byte offset</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Byte offset => 16 bytes = $2^4$ bytes so 4 bits
Index => $256/32 = 8, 2^3$ so 3 bits
Tag => $32-4-3 = 25$ bits
**Question 4.**

During the design phase of his memory system, Ben generates a sequence of block addresses: 0x0200, 0x0018, 0x0200, 0x00B6, 0x0018. He runs them through a 2-way set associative, consisting of four 4-byte blocks. Please fill in the rest of table 1, to help Ben analyze his cache performance. Ben is using FIFO for his cache line replacement policy.

**Table 1: 2-Way Set Associative**

<table>
<thead>
<tr>
<th>Address of memory accessed</th>
<th>Hit or Miss</th>
<th>Content of cache blocks after reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Set 0</td>
</tr>
<tr>
<td>0x0200</td>
<td>Miss</td>
<td>M[0x0200]</td>
</tr>
<tr>
<td>0x0018</td>
<td>Miss</td>
<td>M[0x0200]</td>
</tr>
<tr>
<td>0x0200</td>
<td>Hit</td>
<td>M[0x0200]</td>
</tr>
<tr>
<td>0x00B6</td>
<td>Miss</td>
<td>M[0x00B6]</td>
</tr>
<tr>
<td>0x0018</td>
<td>Hit</td>
<td>M[0x00B6]</td>
</tr>
</tbody>
</table>

**Problem 2: Exercise 2.1 (p. 132-133)**

a. Each element is 8B. Since a 64B cacheline has 8 elements, and each column access will result in fetching a new line for the non-ideal matrix, we need a minimum of 8x8 (64 elements) for each matrix. Hence, the minimum cache size is 128 × 8B = 1KB.

b. The blocked version only has to fetch each input and output element once. The unblocked version will have one cache miss for every 64B/8B = 8 row elements. Each column requires 64Bx256 of storage, or 16KB. Thus, column elements will be replaced in the cache before they can be used again. Hence the unblocked version will have 9 misses (1 row and 8 columns) for every 2 in the blocked version.

c.  

```c
for(i=0; i<256; i=i+B) {
    for (j = 0; j < 256; j=j+B) {
        for(m=0; m<B; m++) {
            for(n=0; n<B; n++) {
                output[j+n][i+m] = input[i+m][j+n];
            }
        }
    }
}
```
d. 2-way set associative. In a direct-mapped cache the blocks could be allocated so that they map to overlapping regions in the cache.

e. You should be able to determine the level-1 cache size by varying the block size. The ratio of the blocked and unblocked program speeds for arrays that do not fit in the cache in comparison to blocks that do is a function of the cache block size, whether the machine has out-of-order issue, and the bandwidth provided by the level-2 cache. You may have discrepancies if your machine has a write-through level-1 cache and the write buffer becomes a limiter of performance.

**Problem 3: Exercise 2.20 (p. 140-141)**

a. Yes. The application and production environment can be run on a VM hosted on a development machine.

b. Yes. Applications can be redeployed on the same environment on top of VMs running on different hardware. This is commonly called business continuity.

c. No. Depending on support in the architecture, virtualizing I/O may add significant or very significant performance overheads.

d. Yes. Applications running on different virtual machines are isolated from each other.

e. Yes.

**Problem 4: Exercise 2.21 (p. 141)**

a. Programs that do a lot of computation but have small memory working sets and do little I/O or other system calls.

b. The slowdown above was 60% for 10%, so 20% system time would run 120% slower.

c. The median slowdown using pure virtualization is 10.3, while for para-virtualization the median slowdown is 3.76.

d. The null call and null I/O call have the largest slowdown. These have no real work to outweigh the virtualization overhead of changing protection levels, so they have the largest slowdowns.