1. An application, running on a system that uses 16-bit addresses, generates the following sequence of address references:

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td>00C5</td>
<td>0000 0000 1100 0101</td>
</tr>
<tr>
<td>166</td>
<td>00A6</td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>00E5</td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>0117</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>003D</td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>00EE</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>00BE</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>00A5</td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>00EF</td>
<td></td>
</tr>
<tr>
<td>279</td>
<td>0117</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>003C</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>00ED</td>
<td></td>
</tr>
</tbody>
</table>

a) [12 points] Assume the system uses a direct-mapped cache that consists of 8 sets, each holding a single block (line) of 16 bytes.

   For each address reference above, show the binary tag, the binary set number to which the reference maps, the binary byte offset of the reference, and indicate whether the reference will lead to a cache hit or a cache miss.

   For example, the first reference yields (format your answer as shown):

<table>
<thead>
<tr>
<th>Addr</th>
<th>Tag</th>
<th>Set</th>
<th>Idx</th>
<th>H/M?</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C5</td>
<td>0000</td>
<td>0000</td>
<td>0101</td>
<td>M</td>
</tr>
</tbody>
</table>

b) [24 points] Assume the system uses a set-associative cache that consists of 4 sets, each holding 4 blocks (lines) of 8 bytes. Replacements, if necessary, are performed according to a strict LRU policy.

   For each address reference above, show the binary tag, the binary set number to which the reference maps, the binary byte offset of the reference, and indicate whether the reference will lead to a cache hit or a cache miss.

   Hint: The solutions to these questions can be generated tediously by hand, more conveniently by writing a short C program to simulate the operation of the cache. I did so in just over 300 lines of code.

2. A system uses 32-bit addresses, and a direct-mapped cache, incorporating a valid bit and a modified bit per cached block. Byte offsets are determined using bits 0-6, set numbers using bits 7-16, and tags using bits 17-31.

   a) [8 points] How many bytes are in a user data block?

   b) [10 points] How many blocks of user data can the cache hold?

   c) [10 points] What is the ratio between the number of bits of user data the cache can hold and the total number of bits in the cache, including any "administrative" overhead?
3. Given the following type definition

```c
struct _Motion {
    int deltaP[3];
    int deltaV[3];
};
typedef struct _Motion Motion;
```

and the following declaration of an array

```c
#define N 1000
Motion M[N];
```

a C programmer is considering three possible designs for a function to initialize all the elements of all the `struct` variables to hold zeros:

```c
void Init1(Motion* pM, int Dim) {
    int i, j;
    for (i = 0; i < Dim; i++) {
        for (j = 0; j < 3; j++) {
            pM[i].deltaP[j] = 0;
            pM[i].deltaV[j] = 0;
        }
    }
}
```

```c
void Init2(Motion* pM, int Dim) {
    int i, j;
    for (i = 0; i < Dim; i++) {
        for (j = 0; j < 3; j++) {
            pM[i].deltaP[j] = 0;
        }
        for (j = 0; j < 3; j++) {
            pM[i].deltaV[j] = 0;
        }
    }
}
```

```c
void Init3(Motion* pM, int Dim) {
    int i, j;
    for (j = 0; j < 3; j++) {
        for (i = 0; i < Dim; i++) {
            pM[i].deltaP[j] = 0;
        }
        for (i = 0; i < Dim; i++) {
            pM[i].deltaV[j] = 0;
        }
    }
}
```

a) [8 points] A C compiler would be expected to store the loop counters i and j in registers persistently, rather than swapping their values in and out from memory. Considering the issue of locality of reference, why is this a reasonable way for the compiler to treat these variables? Be very specific.

b) [12 points] Consider the issue of spatial locality in the three functions given above. Rank the three functions from best to worst, with respect to spatial locality. Explain clearly why you ranked the functions in the order you decide on.
4. [8 points] A system has an L1 cache and an L2 cache. The L1 cache has an average hit rate of 92%, and takes 1 clock cycle to access. The L2 cache has an average hit rate of 98%, and takes 10 clock cycles to access. The access time for DRAM is 120 clock cycles.

What is the average memory access time for this system? Round your answer to the nearest tenth of a clock cycle.

5. [8 points] A system has an L1 cache with an access time of 1 clock cycle and DRAM with an access time of 100 clock cycles. The cache miss rate for instructions is 4%, and the cache miss rate for data references is 12%. On average, 30% of instructions are loads or stores. The base CPI, assuming ideal cache performance, is 3.

What is the actual CPI for this system? Round your answer to the nearest tenth of a clock cycle.