Example Cache Problems

Problem 1:

Consider the following program:

```
typedef struct {
  int val;              // The value we care about
  float alpha;
  float beta;
  float gamma;
  double eta;
  double teta;
} cell;

#define N 4096
#define STEPS 2

cell cvalues[N];
cell tcvalues[N];

int values[N];
int tvalues[N];

void jacobi1()
{
    int i, k;
    cell *ptr1, *ptr2, *tmp;

    ptr1 = cvalues;
    ptr2 = tcvalues;

    for(k = 0; k < STEPS; k++)
    {
        for(i = 1; i < N-1; i++)
        {
            ptr2[i].val = (ptr1[i-1].val+ptr1[i].val+ptr1[i+1].val)/3;
        }
        tmp = ptr1;
        ptr1 = ptr2;
        ptr2 = tmp;
    }
}
```

```
void jacobi2()
{
    int i, k;
    int *ptr1, *ptr2, *tmp;

    ptr1 = values;
    ptr2 = tvalues;

    for(k = 0; k < STEPS; k++)
    {
        for(i = 1; i < N-1; i++)
        {
            ptr2[i] = (ptr1[i-1]+ptr1[i]+ptr1[i+1])/3;
        }
        tmp = ptr1;
        ptr1 = ptr2;
        ptr2 = tmp;
    }
}
```

In all problems, assume that global variables are allocated adjacent to each other in memory (so &cvalues[N] == &tcvalues[0]) and that the values of the global arrays are not already in the cache. You answer does not have to be exact (ie, 1023 instead of 1024.. an answer of 1K is sufficient).

A. Assuming the code is run on a machine with a 16K **direct mapped** level one data cache with 32 byte cache lines, what is the number of cache misses incurred during a run of **jacobi1**?

If the code is run on a machine with a 16K **2-way associative** level one data cache with 32 byte cache lines, what is the number of cache misses incurred during a run of **jacobi1**?

Assuming the code is run on a machine with a 16K **direct mapped** level one data cache with 32 byte cache lines, what is the number of cache misses incurred during a run of **jacobi2**?

If the code is run on a machine with a 16K **2-way associative** level one data cache with 32 byte cache lines, what is the number of cache misses incurred during a run of **jacobi2**?

B. Suppose we're just interested in finding the value of the first (index 1) element in the array and, to make our lives easier we don't do a strict Jacobian computation (we use new and old values):

```
int just_first_value()
{
    int i, k;
    int *ptr1, *ptr2, *tmp;

    ptr1 = values;

    for(k = STEPS; k > 0; k--)
    {
for(i = 1; i <= k; i++)
{
    ptr1[i] = (ptr1[i-1]+ptr1[i]+ptr1[i+1])/3;
}
    }
    return *ptr1;
}
```

If the code is run on a machine with a 16K **2-way associative** level one data cache with 32 byte cache lines, what is the number of cache misses incurred during a run of **just_first_value**?

How many more cache misses would be incurred if the code is changed to compute the first two values?

Imagine this approach is adapated to compute an arbitrary number of first values and we run it on a machine with a fully associative cache with 4byte cache lines. What is the minimum size of such cache that will ensure this algorithm will never evict a cache line that it will want to use later?

Problem 2:

Part 1

Consider a computer with a 12-bit address space and a two level cache. Both levels use a LRU replacement policy. The parameters of the caches are as follows:

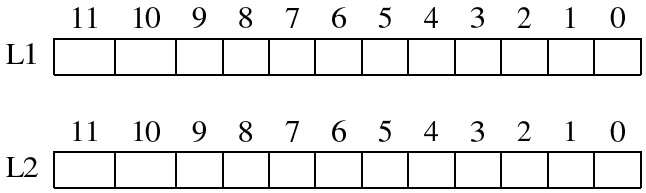
- **L1:** 32 bytes, direct mapped, 8-byte cache lines.
- **L2:** 512 bytes, 4-way set associative, 32-byte cache lines.

The boxes below represent the bit-format of a physical address. In each box, for each cache (L1 and L2), indicate which field that bit represents (it's possible that a field doesn't exist). Here are the fields:

O: Byte offset within the cache line

I: The cache (set) index

T: The cache tag



The table below shows a trace of memory accesses (loads) made by the processor. For each access specify whether it is a level 1 cache hit (Ll), a level 2 cache hit (L2), or a miss (M). If the access is a hit, specify which previous access (by line number) loaded the value into the cache. **Assume that initially all cache lines are invalid.**

Load No.	Binary Address	L1, L2 or M	Which line loaded?
1	1011 0101 0111		
2	1100 0111 0101		
3	1110 1111 1100		
4	1011 0101 0000		
5	1011 0101 1100		
6	1011 0101 0010		
7	1011 0111 1010		
8	0001 0111 1001		
9	1100 0101 0000		
10	1111 0111 1111		
11	1100 0111 0111		

Suggestion: Work out the L1 hits before dealing with the L2 hits.

Recommended Book Practice Problems: 6.9,6.10,6.14,6.15,6.16,6.18
Solutions are at the end of the chapter.