Set Associative Caches

Similar to Direct Mapped, except that multiple blocks can be stored at each Index. Must look at ALL tags at a given Index to determine if hit or miss. Must invoke the **replacement policy** on a miss if a given set is full.

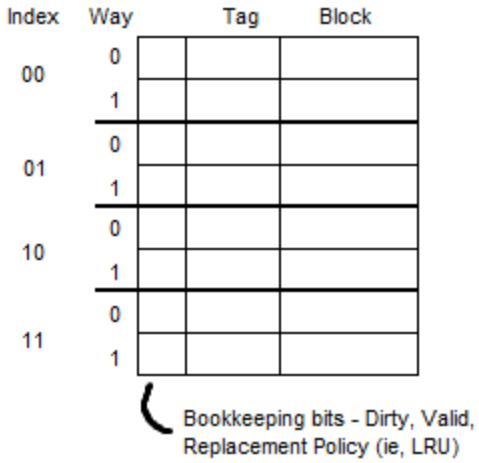


Figure 1. An 8 block, 2 way set associative cache (4 sets).

1. How big should the T, I, and O fields be on a system with …

   i) 32-bit addressed memory, 64KB fully associative cache, 4-byte blocks

30/0/2

   ii) 8-bit addressed memory, 32 B 2-way set associative cache, 4-byte blocks

4/2/2

   iii) 8-bit addressed memory, 32 B 4-way set associative cache, 4-byte blocks.

5/1/2

2. A 32kB cache has a line size of 16 bytes and is 4-way set-associative. How many bits of an address will be in the Tag, Index, and Offset? Assume 32 bit addresses.

19 Tag, 9 Index, 4 Offset

3. In a 2-way set-associative cache, three addresses, A, B, and C, all have the same index but distant tags. What is the minimum sequence of access which, if repeated, will maximize the miss rate in the cache if it uses the LRU replacement policy?

A B C. Every time the newly loaded element will kick the oldest element out. We always get a miss in this scheme.

4. If the above sequence is repeated for a long period of time, what will the miss rate be if the cache uses an LRU replacement policy?

100% miss rate

5. If the hit time is 1 cycle, and the miss penalty is 3 cycles, what will be the average memory access time (in clock cycles) for the LRU replacement policy using the above sequence?

AMAT = hit time + miss time = 1 + 1*3 = 4.

6. What would be a better replacement policy?

MRU (most recently used), random.

7. What will the miss rate be for LRU replacement when the sequence is ABC CBA ABC CBA …?

Amortized over time, the miss rate will be 1/3 = 33% (first two of the three hit, last misses).

Memory Management in C

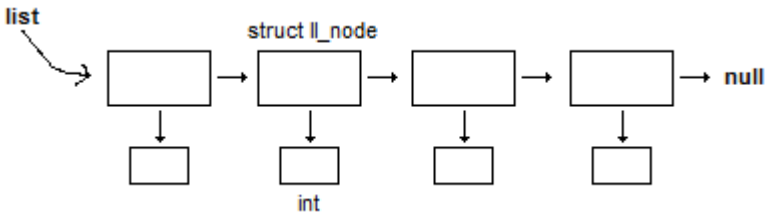
8. What is wrong with the C code below:

```
int* ptr = (int*) malloc(4 * sizeof(int));
if(extra_large) {
ptr = (int*) malloc(10 * sizeof(int));
}
return ptr;
```

If `extra_large` is true, we have a memory leak (we lose the pointer to the memory initially allocated).

9. For the singly linked list implementation below, fill out `free_ll`, which frees all of the memory allocated for the linked list.

```
struct ll_node {
  struct ll_node* next;
  int *element;
}
```




```
void free_ll(struct ll_node* list) {
    /* YOUR CODE HERE */
    if(list) {
        free_ll(list->next);
        free(list->element);
        free(list);
    }
}
```