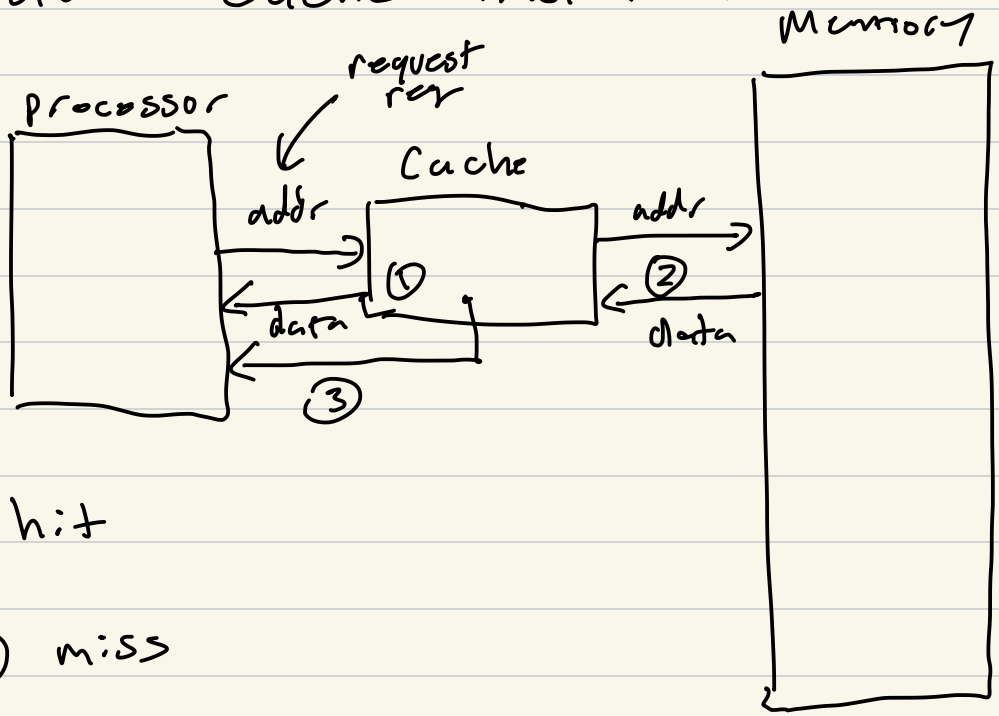
CS315-01 Cache Simulation



① hit

②③ miss

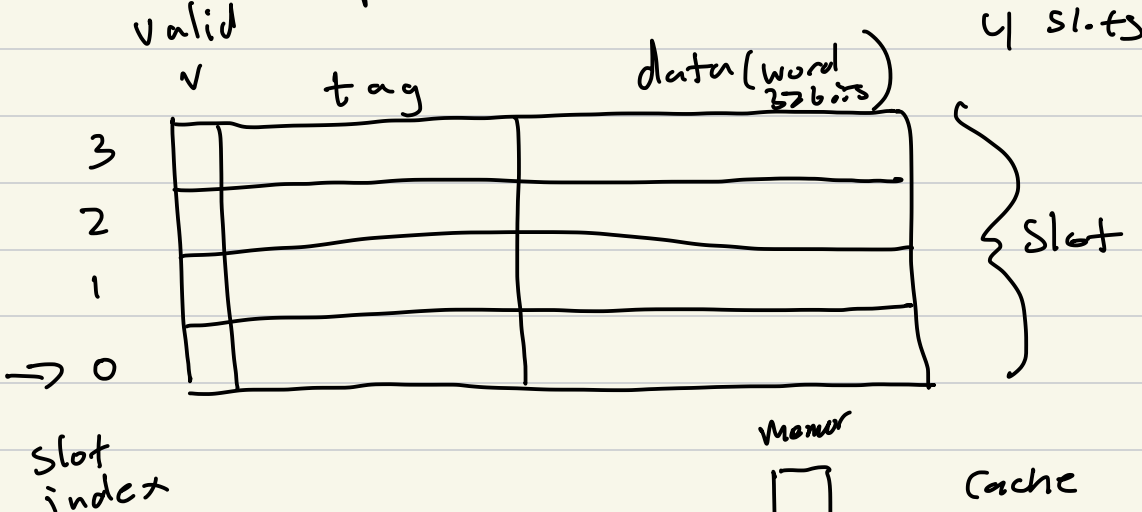
total number of memory requests (reqs)

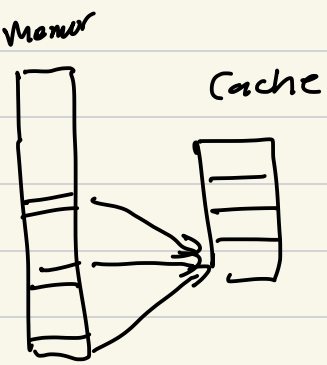
$$\text{hit rate} = \frac{\#\text{hits}}{\#\text{reqs}}$$

$$\text{miss rate} = \frac{\#\text{misses}}{\#\text{reqs}}$$

$$\text{hit rate} = 1 - \text{miss rate}$$

Direct Mapped Cache





addr
byte addr

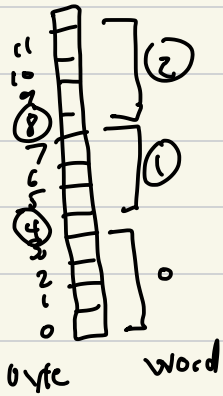
assume addr is word aligned

addr_word = addr_byte / 4

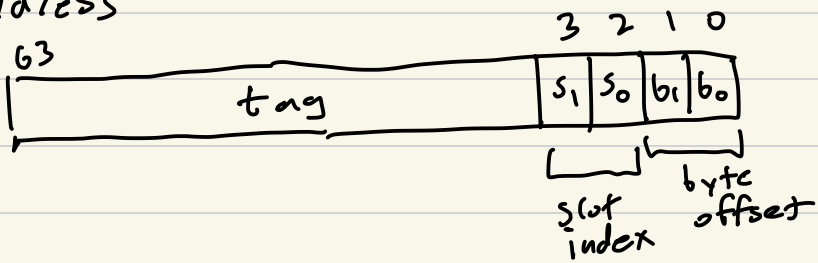
slot_index = addr_word % 4

e.g. 4 word
$4 \% 4 = 0$

24
$17 \% 4 = 1$



address



slot_index = (addr >> 2) & 0b11

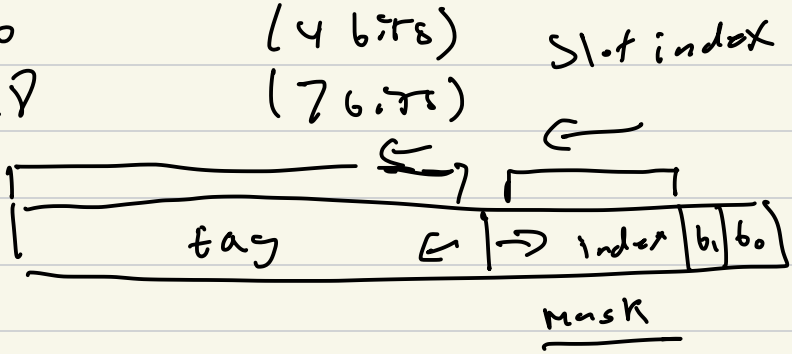
tag = addr >> 4

Direct Mapped Pseudo Code

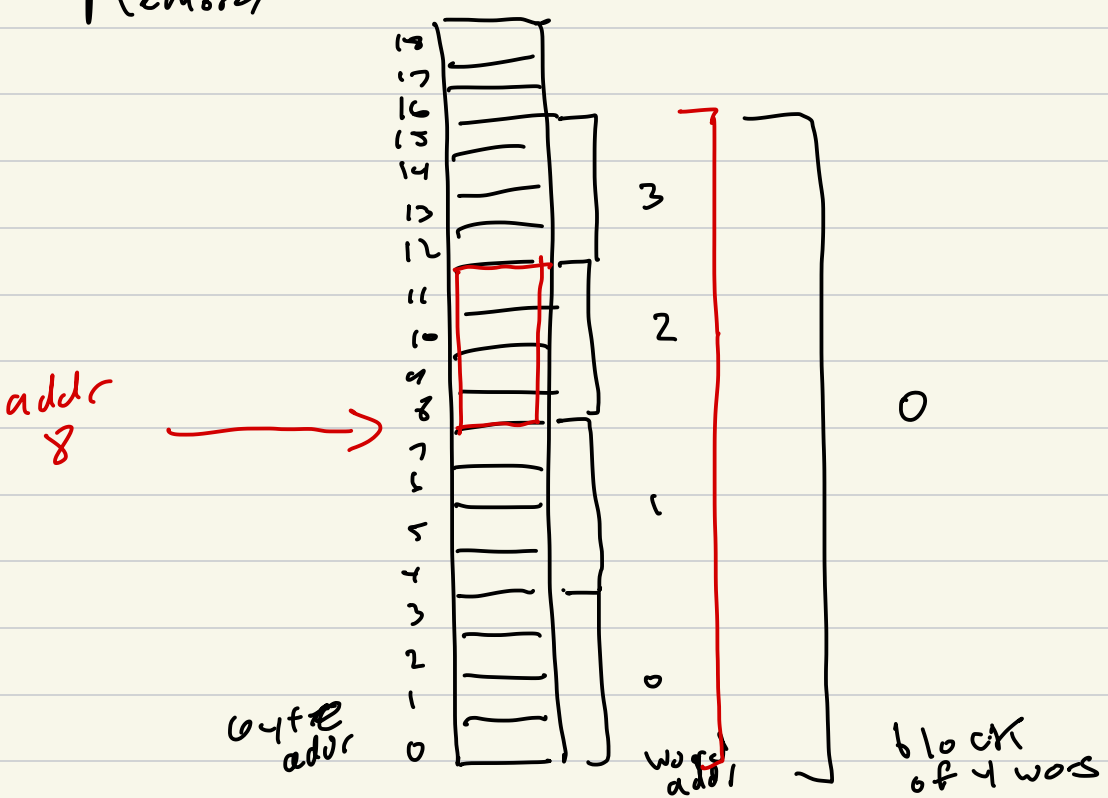
```
Lookup (addr)

    tag = addr >> 4;
    index_mask = 0b11;
    slot_index = (addr >> 2) & index_mask
    slot = cache [slot_index]
    if (slot.valid == 1 && slot.tag == tag) {
        // hit
        return slot.data
    } else {
        // miss
        slot.data = *((uint32_t*) addr);
        slot.tag = tag
        slot.valid = 1
        return slot.data
    }
```

of slots

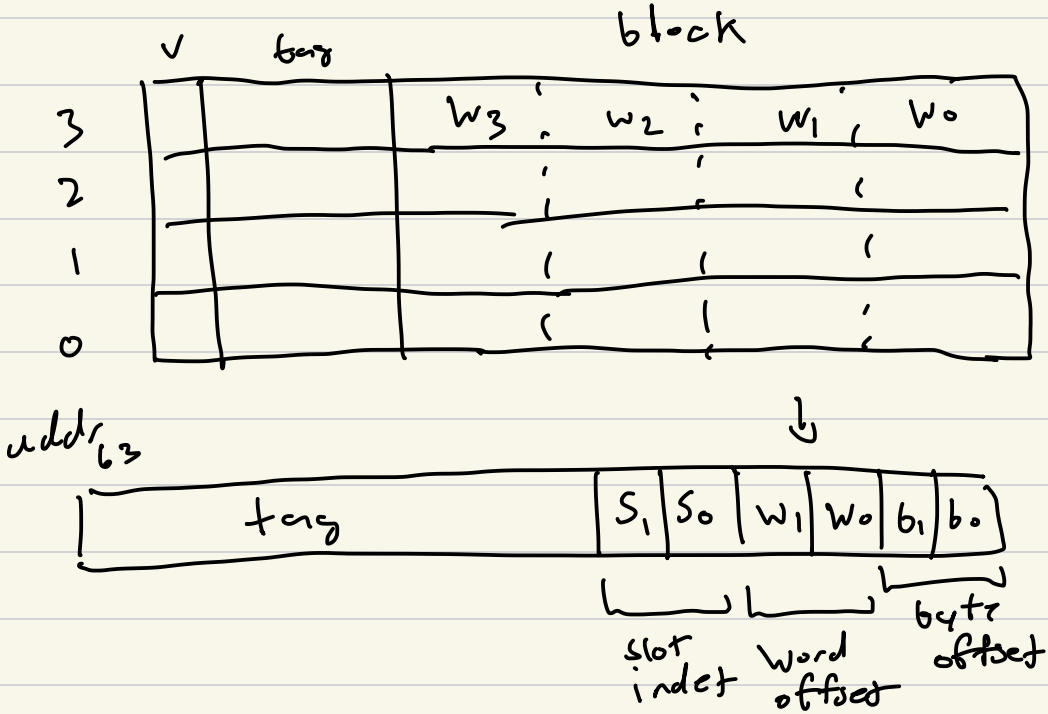
4 slots (2 bits)
8 (3 bits)
16 (4 bits)
128 (7 bits)



Memory



Block Size (size of 4 words)



With Block Size

1) on hit

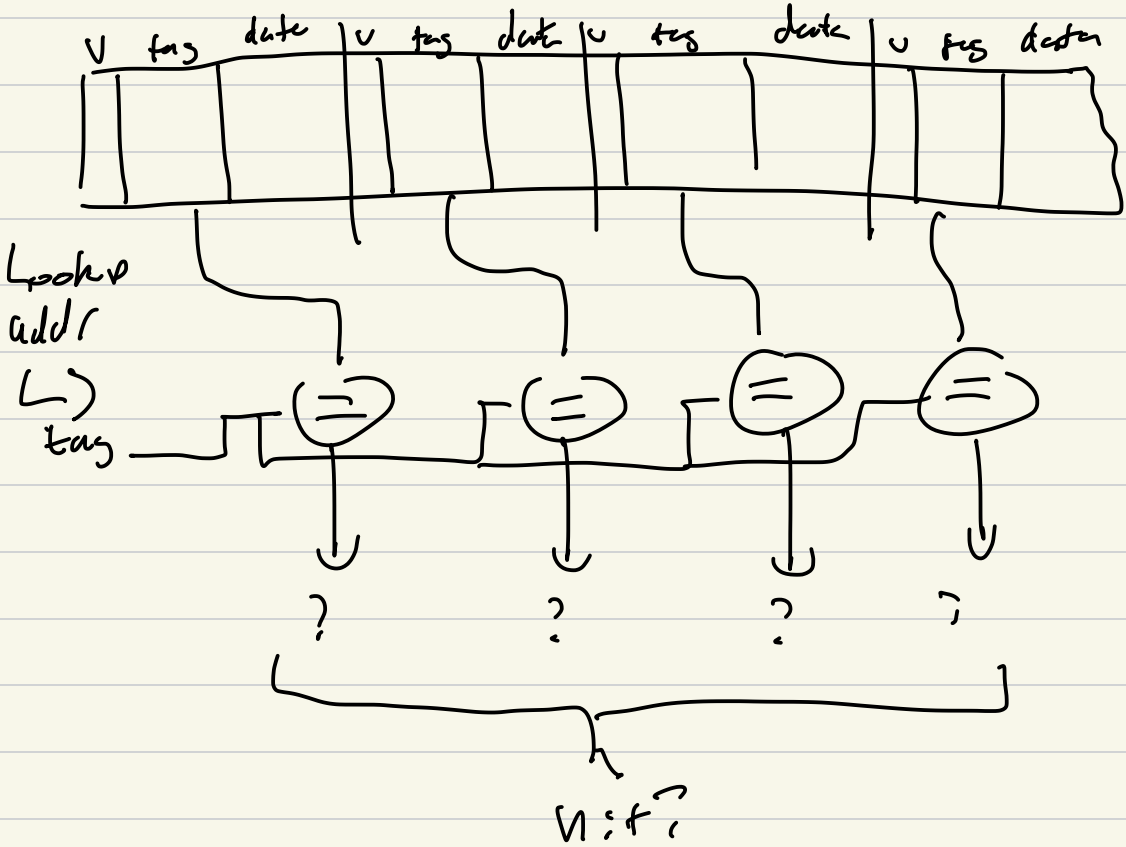
get word from slot using word offset

2) on miss

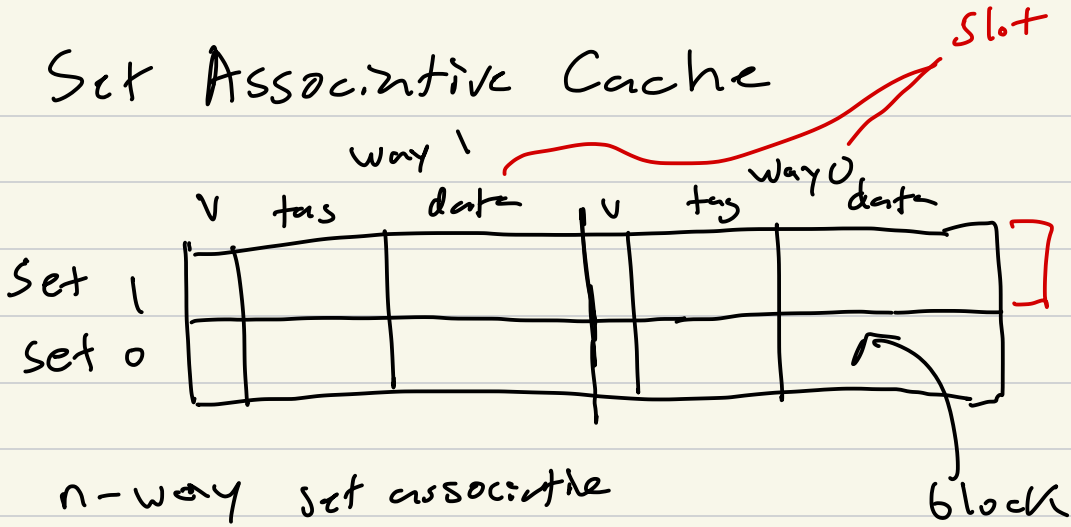
find block_start_addr from addr

get N words from memory

Fully Associative Cache



Set Associative Cache

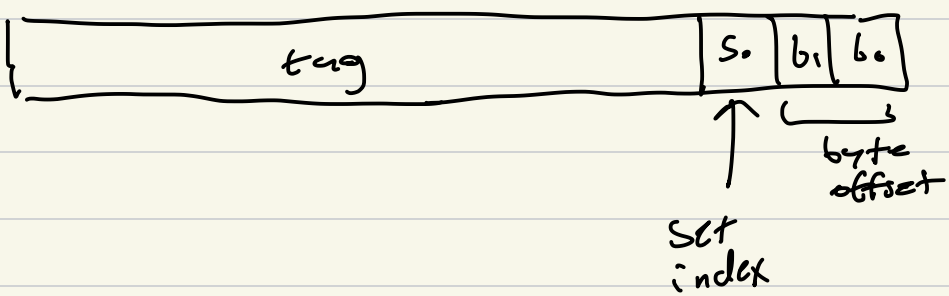


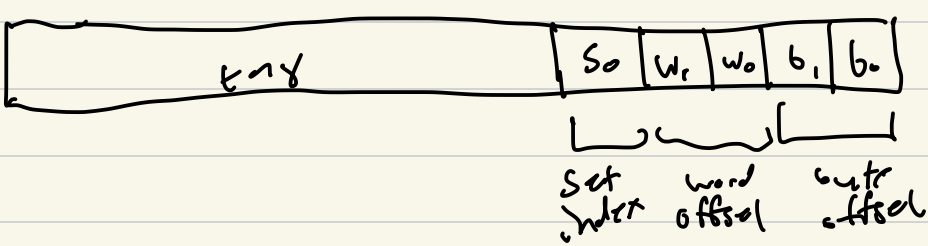
n-way set associative

4-way

addr

2-way





Set Associative Pseudo Code

Lookup(addr) 2-way

```
num_refs += 1
num_ways = 2
    tag = addr >> 3
set_index = (addr >> 2) & 0b1;
set_base = set_index * 2
for(i=0; i<num_ways; i++) {
    slot = cache[set_base + i]
    if (slot.valid &&
        slot.tag == tag)
      // hit
      slot.timestamp = num_refs;
      return slot.data
    }
}
// miss
slot = find_lru_in_set(cache, set_base)
slot.data = *(uint32_t *) addr);
 slot.tag = tag
slot.timestamp = num_refs;
return slot.data;
```