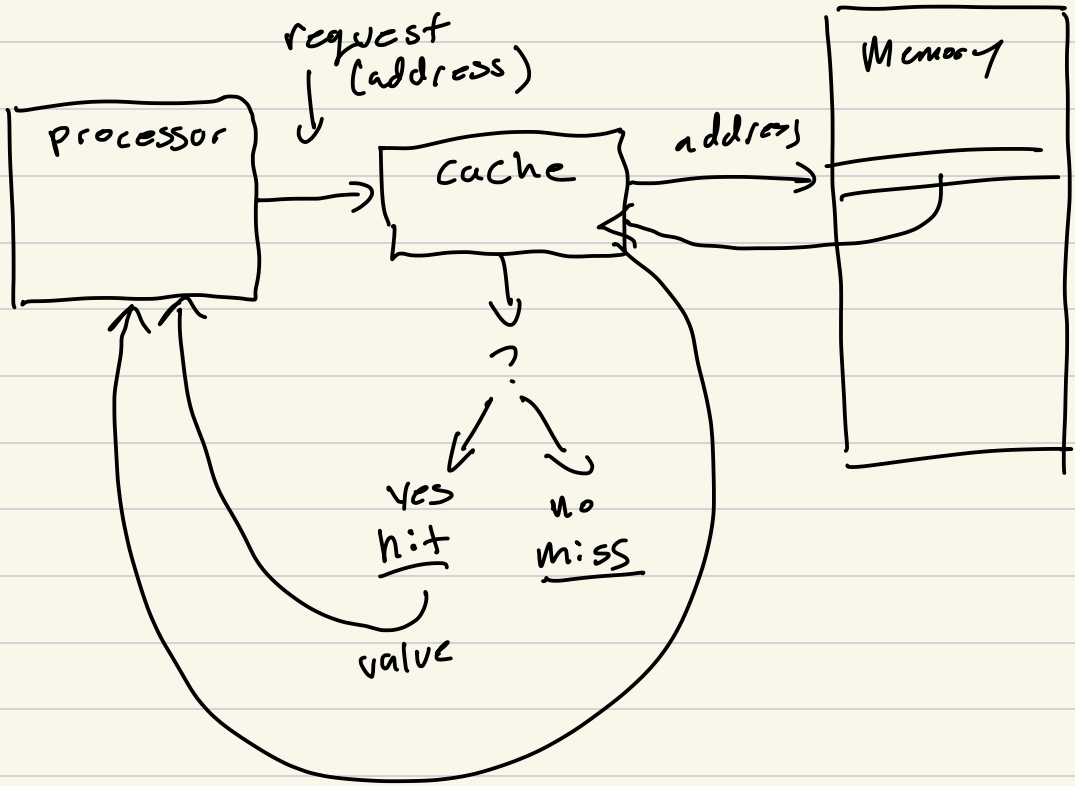
CS315-01 Cache Simulation



memory requests req count

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

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

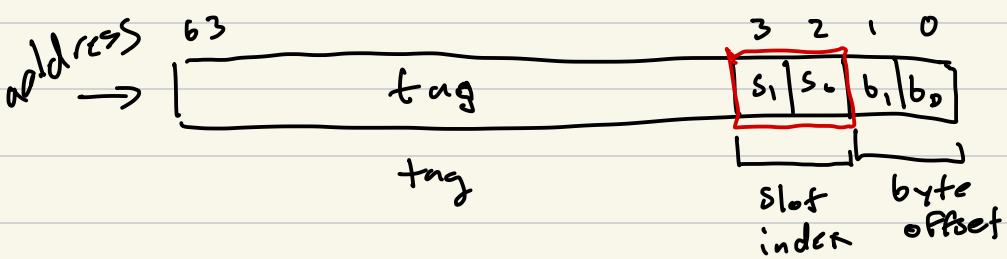
(addr, data)

Direct Mapped

addr assume addr is word aligned

addr_word = addr_byte / 4

slot_index = addr_word % 4 (4 = slots in cache N)



slot_index = (addr >> 2) & 0b11

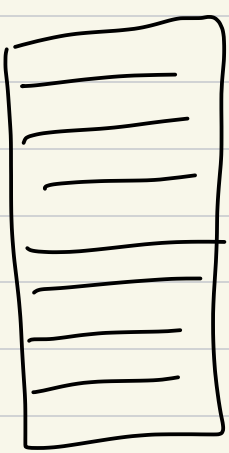
tag = addr >> 4

Direct Mapped Pseudo Code

```
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                    <- load from mem
    slot.data = *((uint32_t *)addr);
    slot.tag = tag
    slot.valid = 1;
}
```

---

8 slots

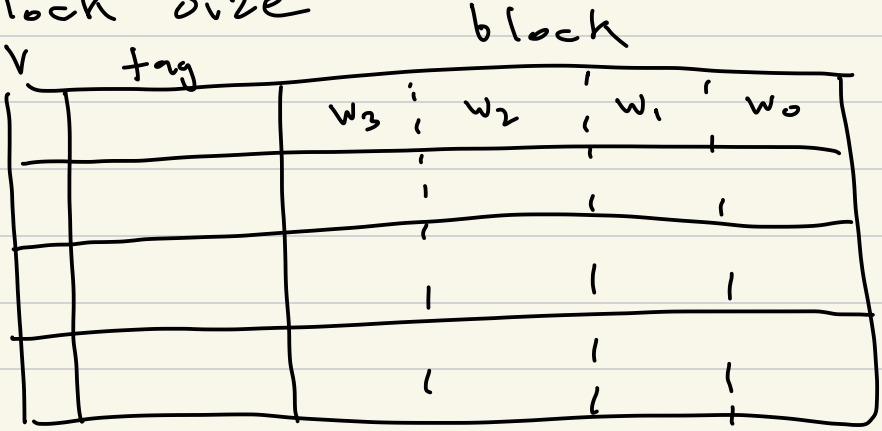


addr_word = addr_byte / 4

slot = addr_word % 8

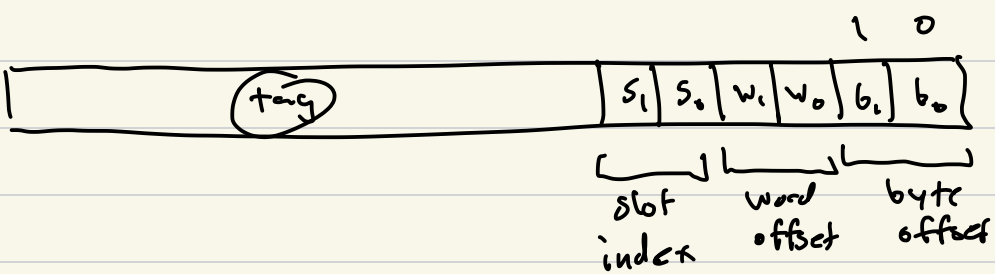
```
tag = addr >> 5;
index_mask = 0b11
slot = (addr >> 2) & index_mask;
```

Block Size



addr

addr_word = addr / 4



slots array

addr

tag

slot

hit

data = slot.block[ ];

block size

block_index = addr_word % 4;

miss

addr

base

block

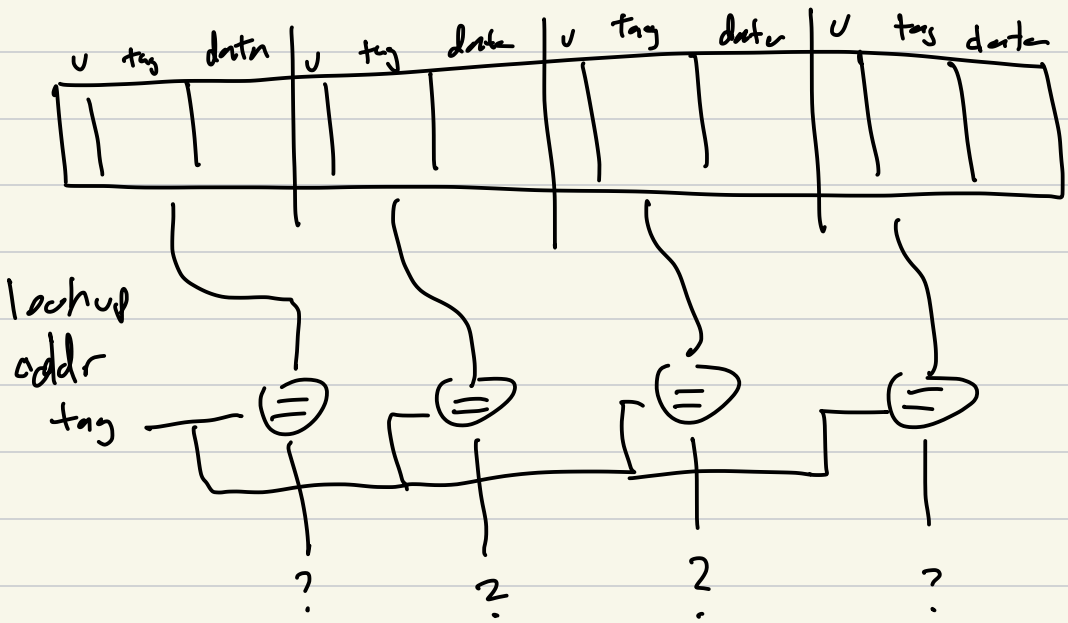
block_base = addr_word − block_index

memory

read entire block into slot

return word/data

Fully Associative Cache

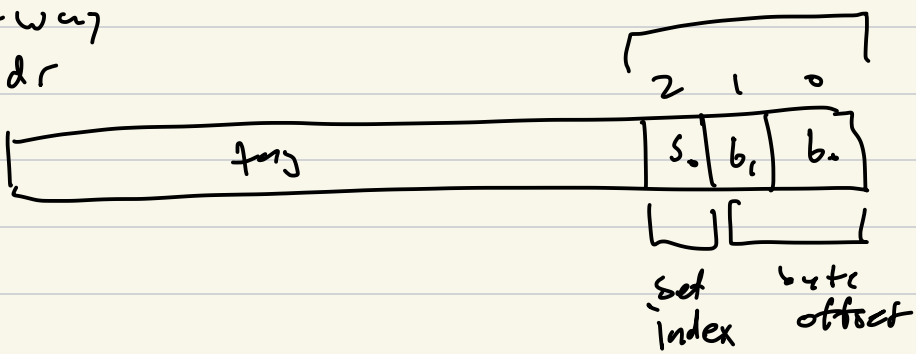


Set Associative Cache

n-way set associative cache

2-way

addr



SA Pseudo code Lookup

```
num_refs += 1;
num_ways = 2;
addr_tag = addr >> 3;
set_index = (addr >> 2) & 0b1;
set_base = set_index * 2   // ways
for(i=0; i<2; 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
```