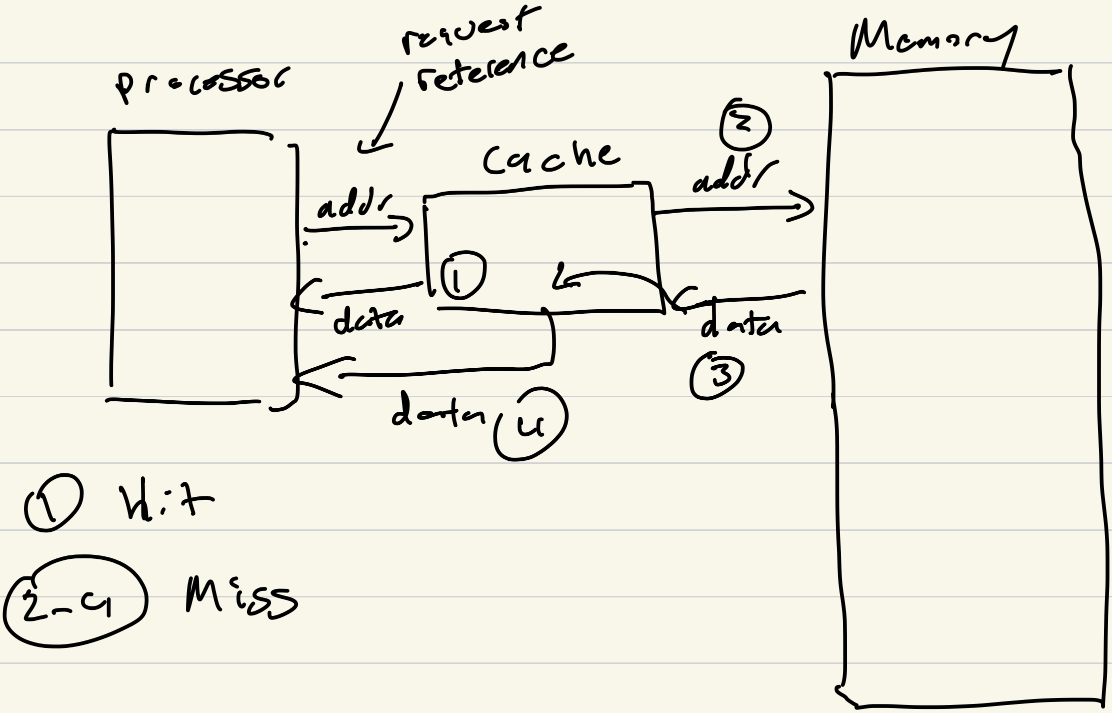
CS 315-02 Cache Simulation



① hit

(2-4) Miss

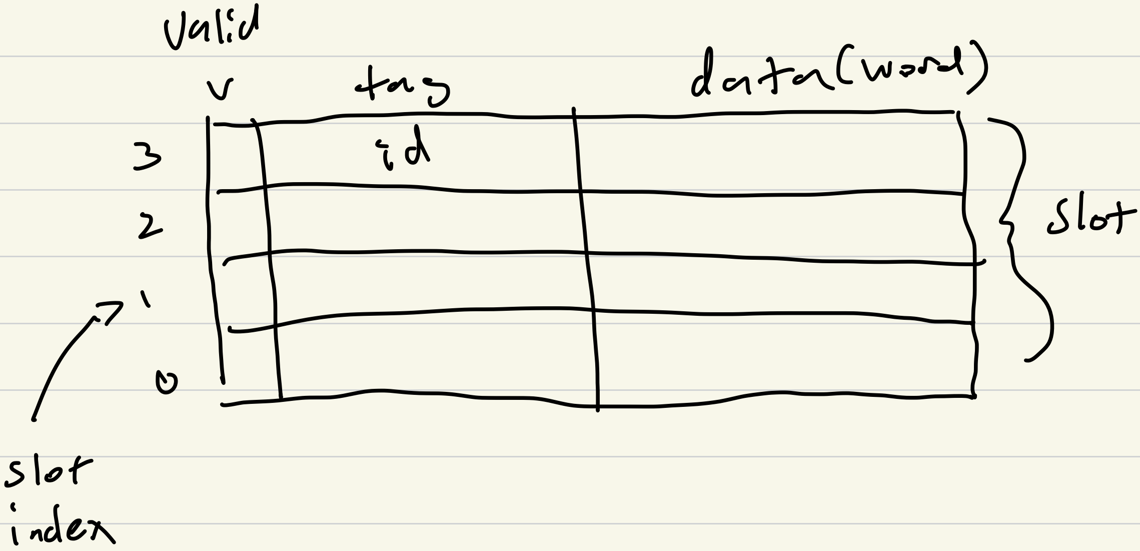
# total number of memory sequests (references)

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

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

hit rate = 1 - miss rate

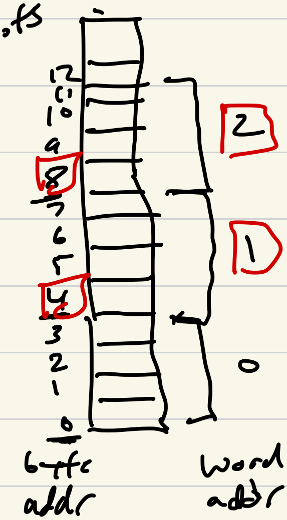
Direct Mapped Cache



addr (byte)   assume addr is word aligned

$addr\_word = addr\_byte / 4$ ← word addr

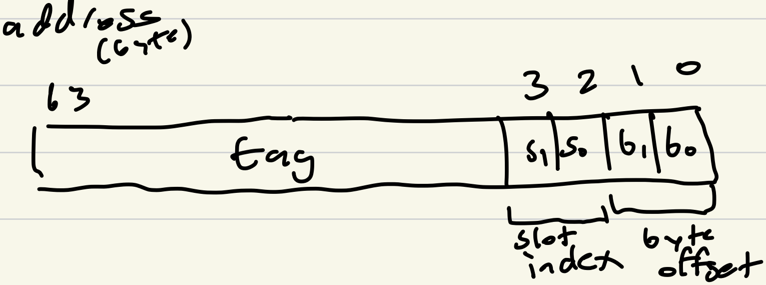
$slot\_index = addr\_word \% 4$ ← # of slots



eg. word addr

$17 \% 4 = 1$

$2002 \% 4 = 2$



$\text{slot\_index} = (\text{addr} >> 2)\ \&\ 0b11$

$\text{tag} \quad = \text{addr} >> 4$

$\underline{\text{addr}} = \overset{\text{tag}}{\overline{128}} >> 4 = \boxed{8}$

$\overset{\text{slot index}}{\overline{(128 >> 2)}} \% 4$

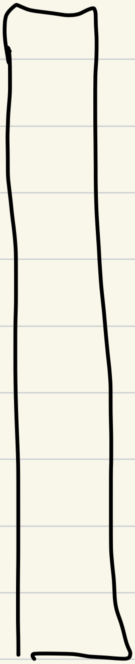
$32 \% 4 = \boxed{0}$

$\underline{\text{addr}} = \overset{\text{tag}}{256} >> 4 = \boxed{16}$

slot inde

$(256 >> 2)$

$64 \% 4 = \boxed{0}$



Direct Mapped Pseudo Code

#slot = 4

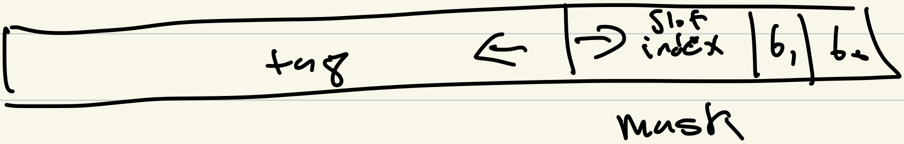
Lookup(addr)

(addr/4)%4

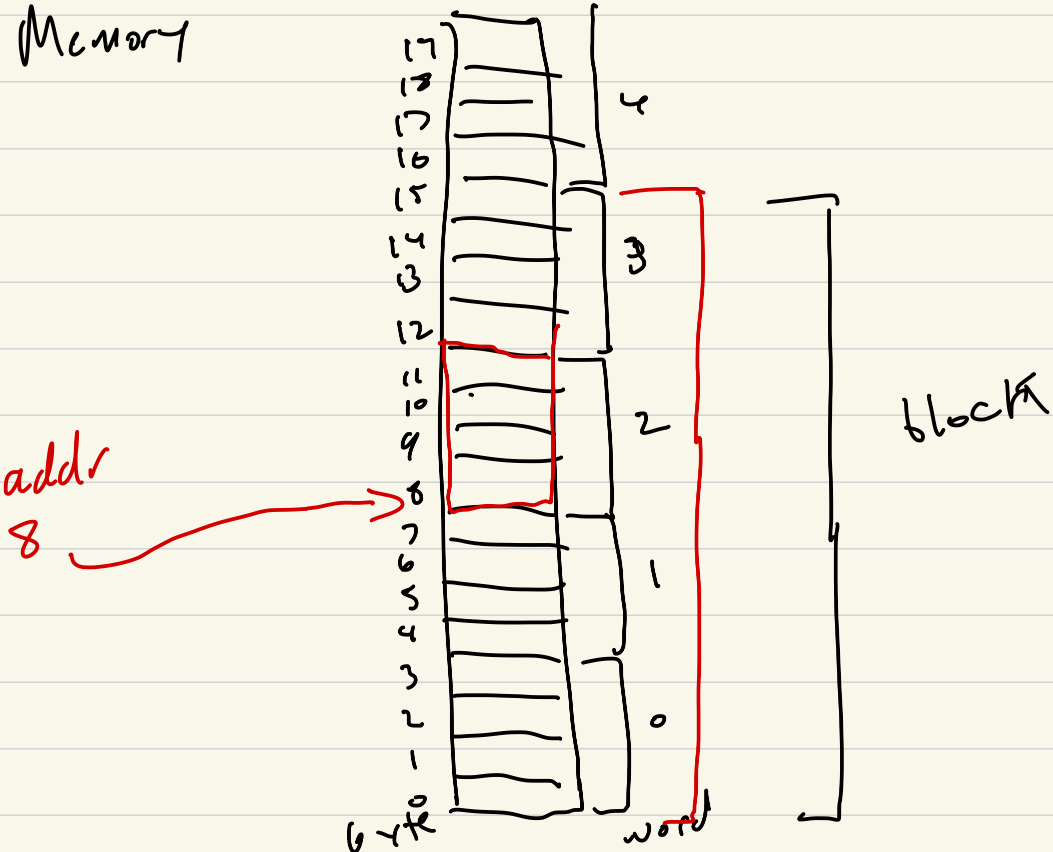
```
tag = addr >>4;
index_mask = 0b11;
slot_index = (addr >>2) & index_mask;
slot = cache[slot_index]
if (slot.valid ==1 && slot.tag == tag) {
    // hit
    return slot.data
} else {
    // miss
    slot.data = *((uint32_t *) addr);
    slot.tag = tag;
    slot.valid = 1;
    return slot.data;
}
```

# slots

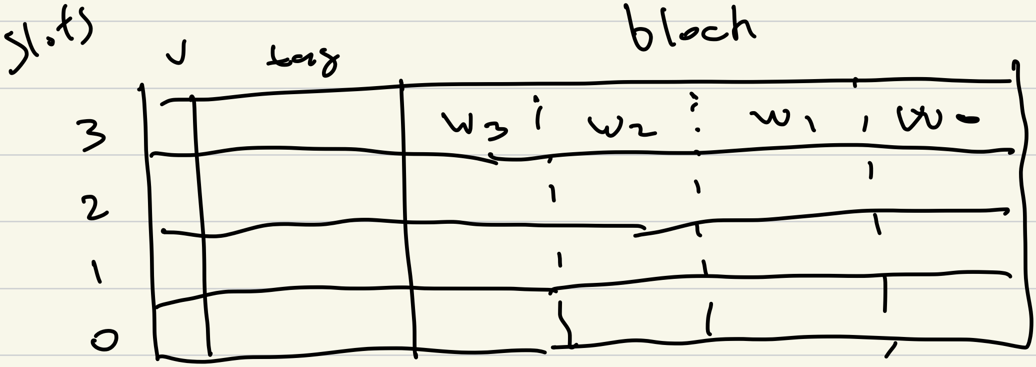
4 slots	(2 bits)
8	(3 bits)
16	(4 bits)
128	(7 bits

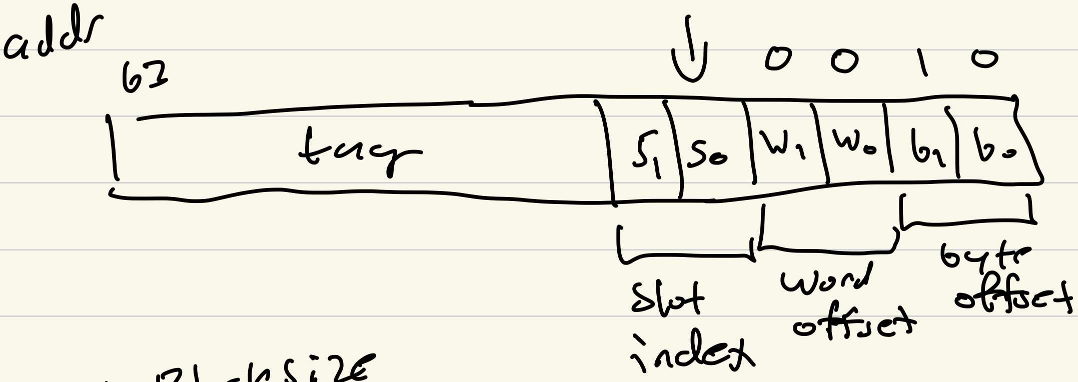


Memory



Block Size (size of 4 words)



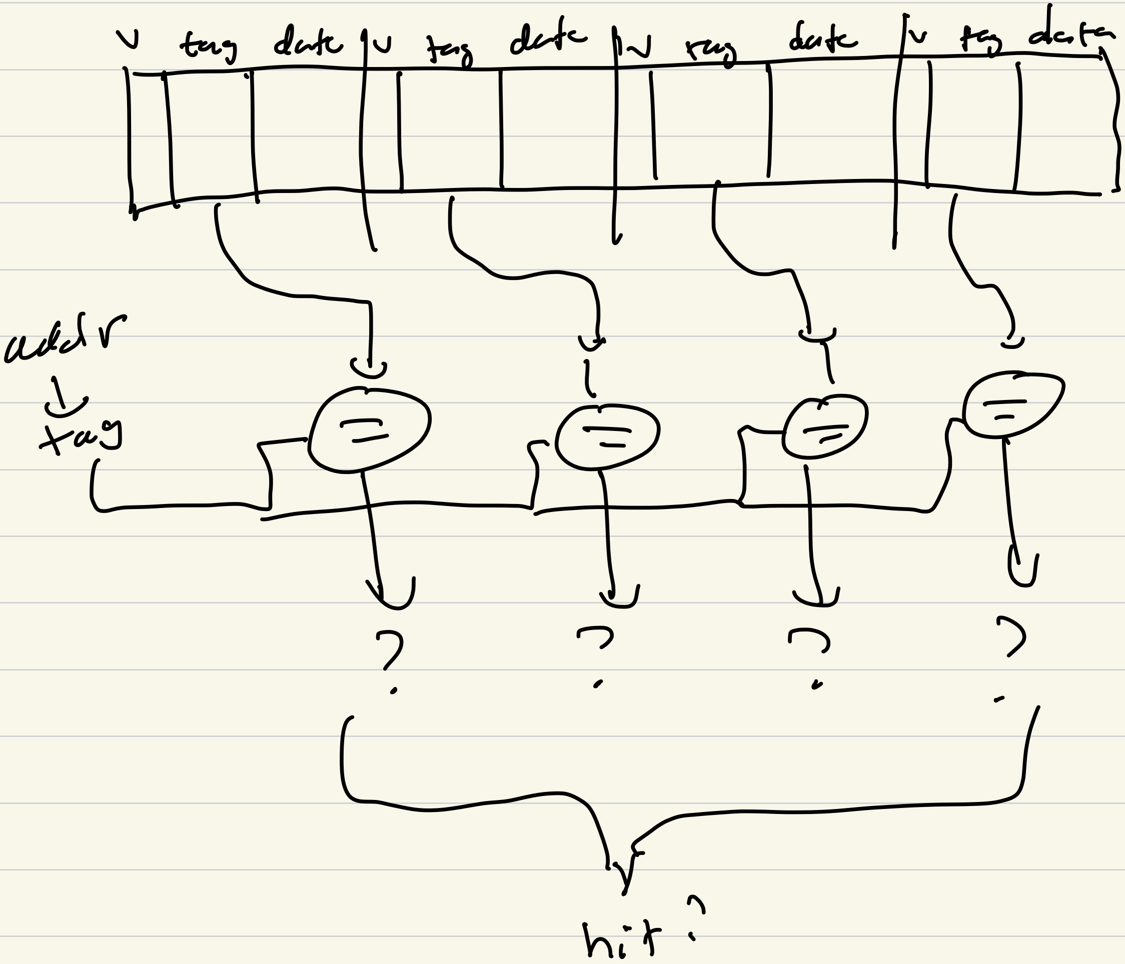


With Blocksize

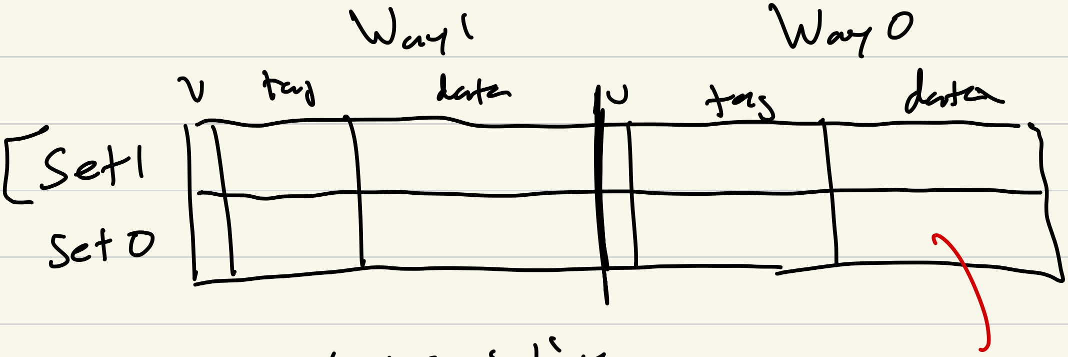
1) on a hit
get the word from slot using the word offset

2) on miss
find block-start-addr from addr
get N words from memory
^ # of words in block (block size)

Fully Associative Cache



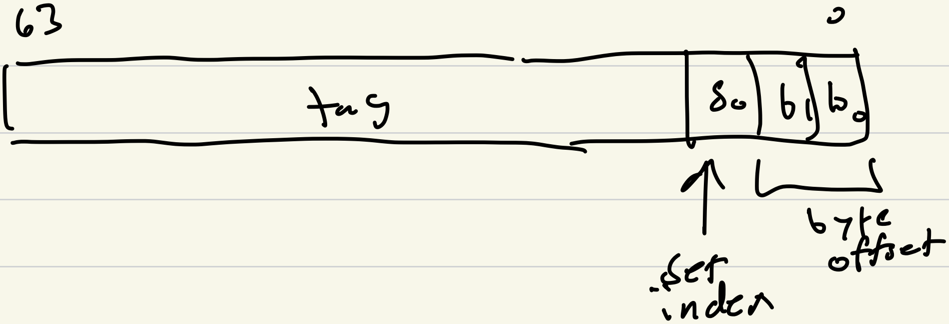
Set Associative Cache



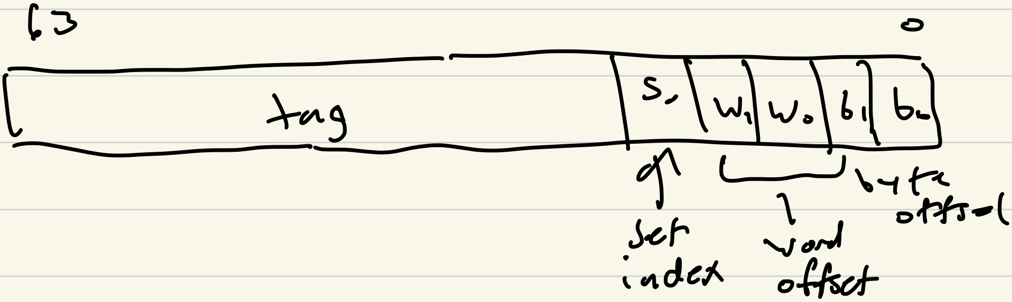
n-way set associative

4-way SA

addr



Block size

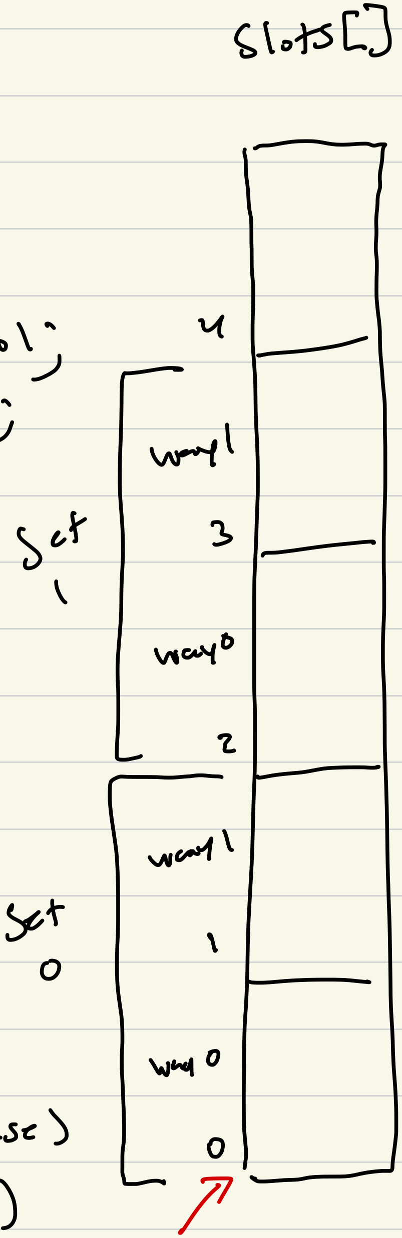


Set Associative Pseudo Code

lookup(addr)   2-way

```
num_refs += 1;
num_ways = 2;
    tag = addr >> 3;
set_index = (addr >> 2) & 0b1;
set_base = set_index * num_ways;
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;
```


slots[]
4
way1
3
set 1
way0
2
way1
1
set 0
way0
0