# Lecture - 27
## Associative Caches

* Problem with direct mapped caches ①

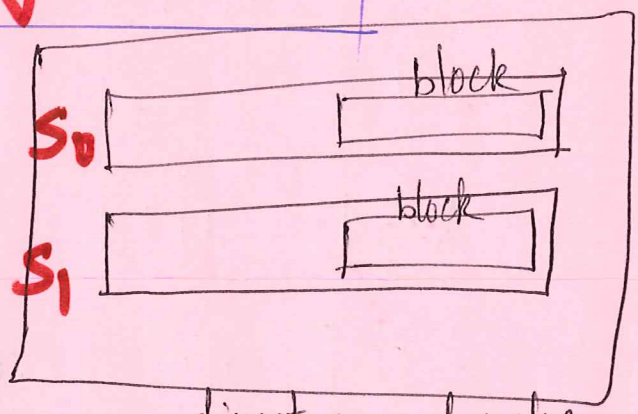
"Conflict misses"

a, b - int arrays of size 4.

```
for (i=0;  i< 8;  i++)
    sum += x[i] * y[i];
```

Cache block size = 8 bytes

Cache size = 16 bytes



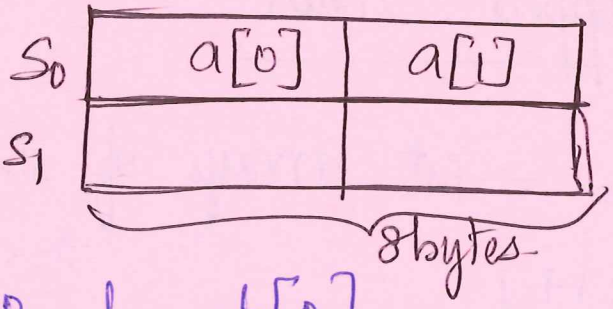

direct mapped cache

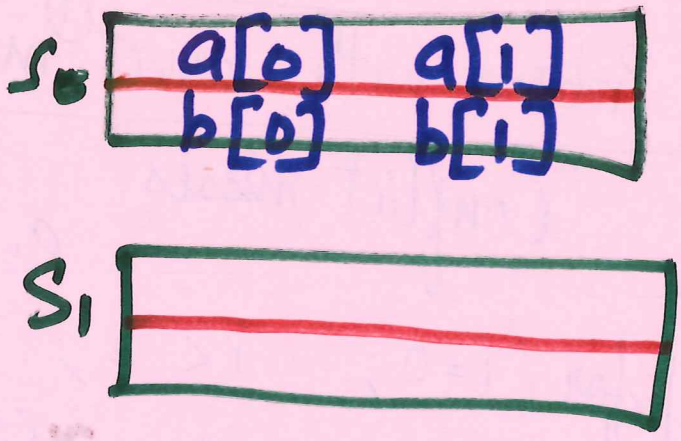
| Elem. | Addr | Set index | |
|---|---|---|---|
| x[0] | 0 | 0 | block 0 |
| x[1] | 4 | 0 | |
| x[2] | 8 | 1 | block 1 |
| x[3] | 12 | 1 | |
| y[0] | 16 | 0 | block 2 |
| y[1] | 20 | 0 | |
| y[2] | 24 | 1 | block 3 |
| y[3] | 28 | 1 | |

x[4]

x[6]

1. Read a[0] .

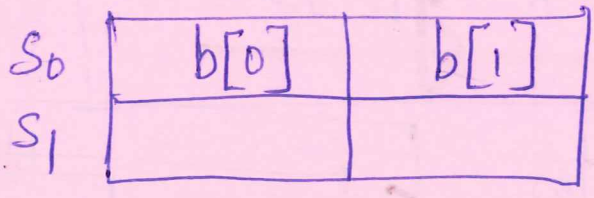


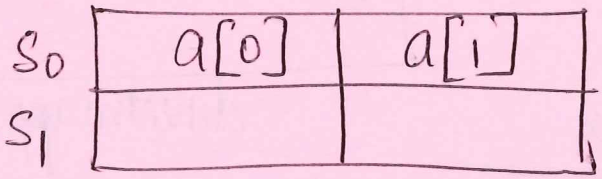




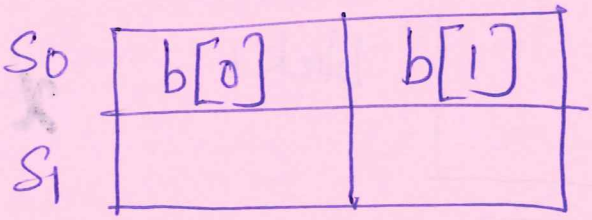
2. Read b[0]




3. Read a[1]




4. Read b[1]




$\Rightarrow$ Trashing!

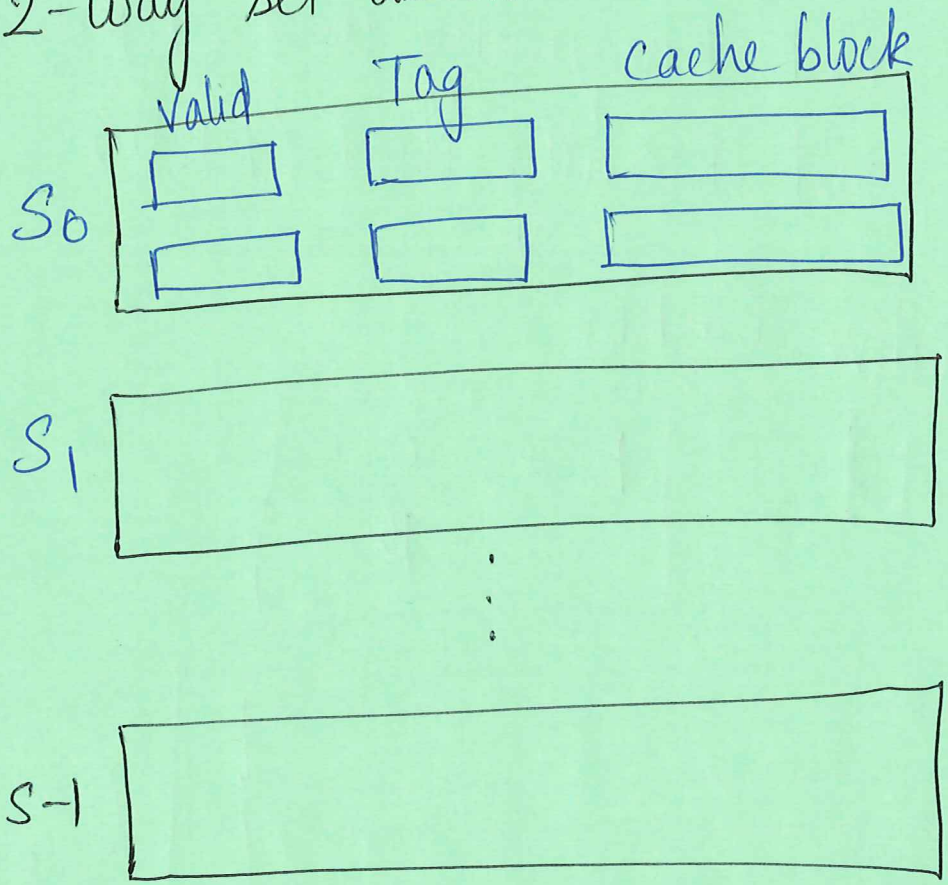
# * Set Associative caches

Each set holds more than one cache line.

$$1 < E < \frac{C}{B}$$

E-way set associative cache.

eg. 2-way set associative cache.



1. Set selection – same as direct-mapped cache.

2. Line matching and word selection. (X)

3. Line replacement on Misses.

Replacement policies

1. Random
2. Least Frequently used (LFU)
3. Least Recently Used (LRU).

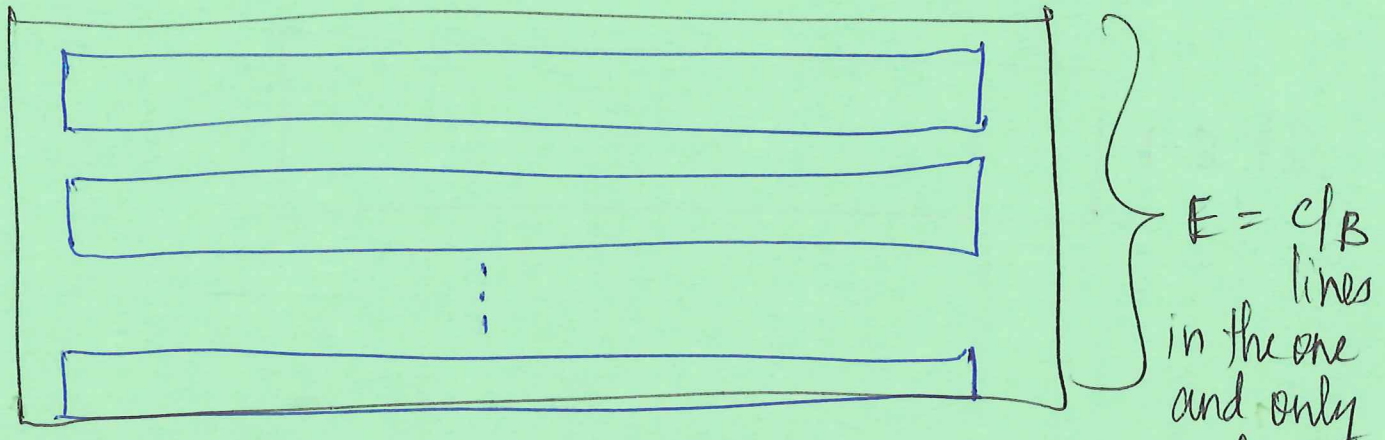
---

* Fully Associative Caches

Only 1 Set!

$$E = \frac{C}{B}$$

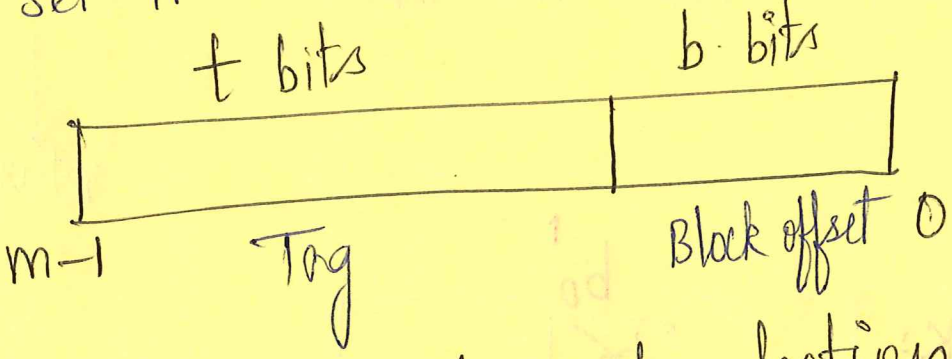
So



## 1. Set Selection

trivial ∵ only one set.

∴ no set index bits in the address




## 2. Line matching and word selection.

III^r to set associative cache.

Fully associative caches → appropriate for small caches

∵ tag matching in a large cache takes a long time

# HW problem 6.30

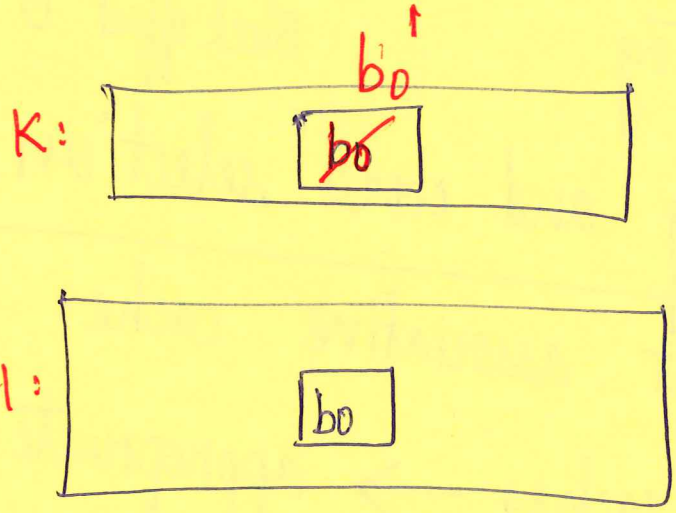
HW: Why the middle bits are used for set-index? Why not the higher order bits?

# Writes

Reads - straightforward.

cache miss? → read from the next lower level → store block in some cache line → return w.

↑ read w




## Writes

### Write through

* more bus traffic
* simple (no dirty bit)
* writes block to level K+1 immediately

### Write-back.

* Less bus traffic
* dirty bit needed.
* writes block to level K+1 only when the block

(7)

# Writes

## write-allocate
loads block in cache and writes in cache

↑

write-back caches

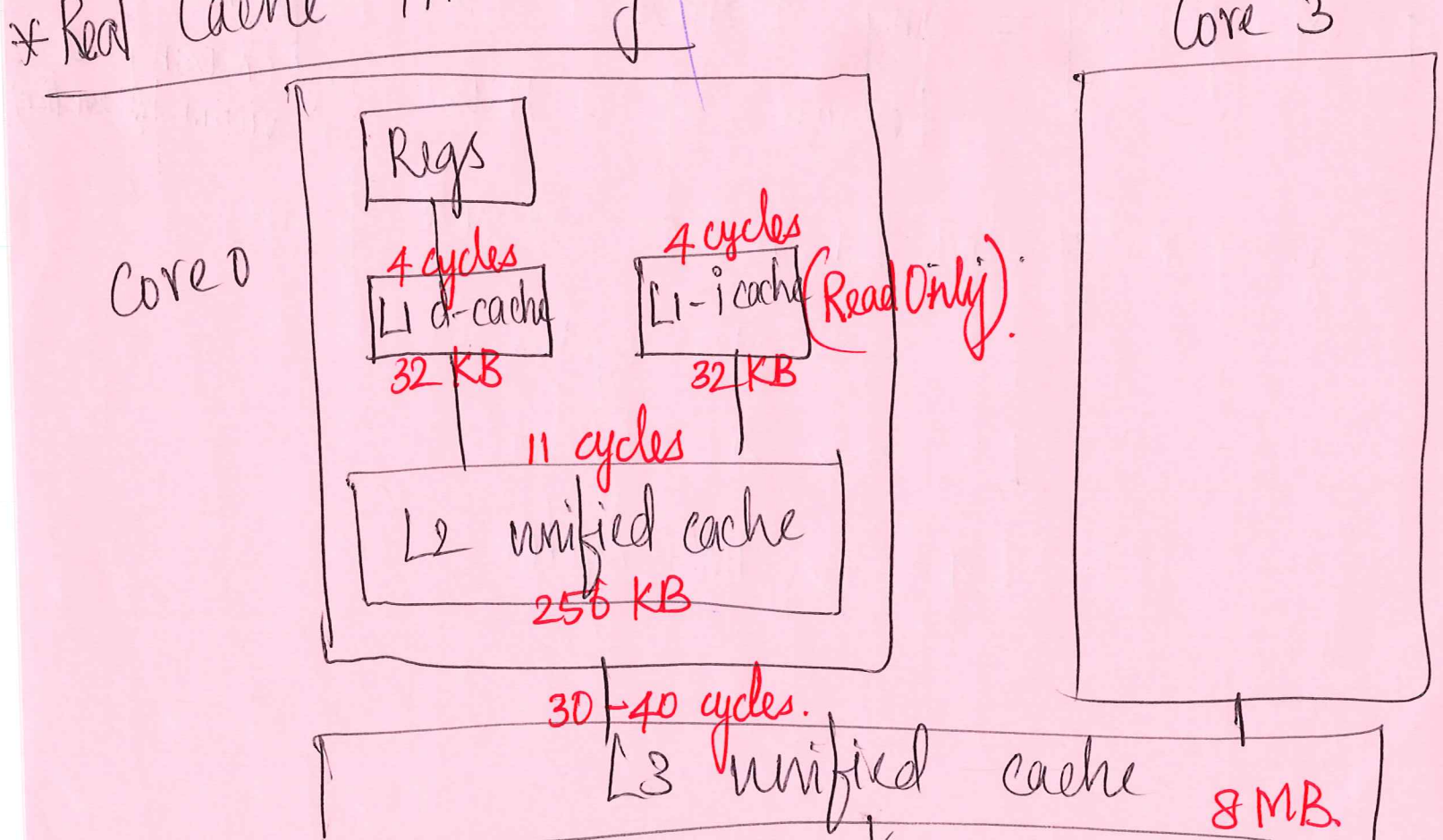
## no-write allocate
writes directly in main memory if block not present in cache.

↑

write through cache

---

## * Real Cache Hierarchy



# Performance impact of cache parameters



1. miss rate = $\frac{\#\text{misses}}{\#\text{references}}$

2. hit rate = 1 − miss rate.

3. hit time = time to deliver a word in the cache to the CPU

$= t_{\text{set selection}} + t_{\text{line identification}} + t_{\text{word identification}}$.

4. Miss penalty = any additional time required because of a miss. (eg: choosing a victim for eviction)

eg. penalty for L1 misses served from:

1. L2 → 10 cycles
2. L3 → 40 cycles
3. L4 → 100 cycles.

## Impact of cache size

large cache ⇒ high hit rate.

⇒ high hit time

∴ Faster caches are usually smaller.

## Impact of block size

larger blocks ⇒ increase hit rate (for programs with good spatial locality).

⇒ decrease hit rate (for programs with good temporal locality)

∵ For a given cache size, larger blocks ⇒ fewer cache lines.

larger blocks ⇒ negative impact on "miss penalty"

∵ larger blocks ⇒ Larger transfer times.

# Impact of Associativity (E)



larger E ⇒ increase hit rate
∵ conflict misses are reduced.

## Choice of E:

hit time vs miss penalty.

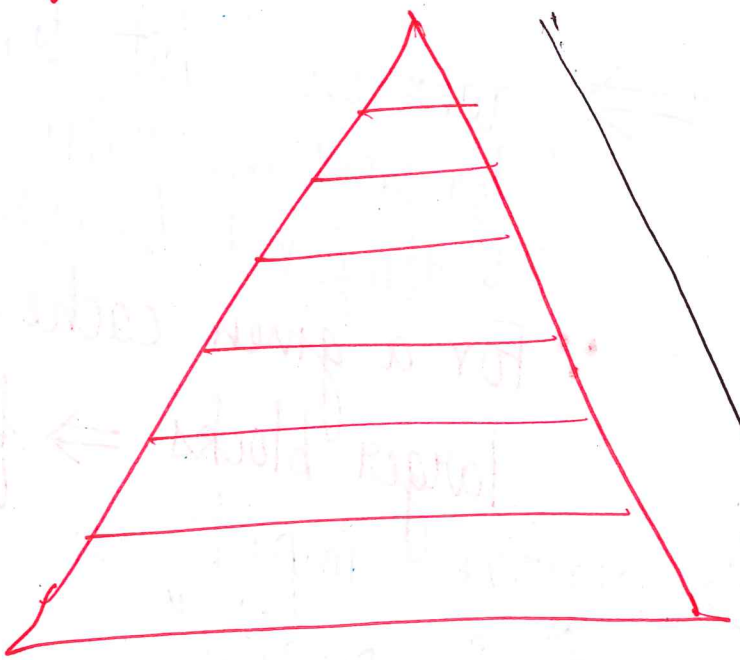
Intel i7

$L_1$, $L_2$ → E = 8

$L_3$ → E = 16

↓ miss penalty increases
~~hit time~~

# Impact of Write Strategy



use of write-back caches increases.
(∵ transfer time increases).