Practical Caches

RECAP

- Fully associative: great for temporal locality, but costly
- Direct Mapped: bad for TL but cheap
  - Compromise: set associative
- LRU: best cache replacement algorithm for temporal locality
  - Expensive too
  - Cyclic, random... alternatives

Cache Control Bits

- Cache INITIALISATION?
  - need VALID bit for each entry to indicate meaningful data
- Dirty bit: ↦ cache value written BACK to memory
  - write through ≠ write back

Exploiting Spatial Locality (!) Complete (!)

- Wider cache line stores

Multiple line Wood size (!) Complete (!)

Effect of Line Size

- Larger line ⟶ fewer misses (:) more CHR

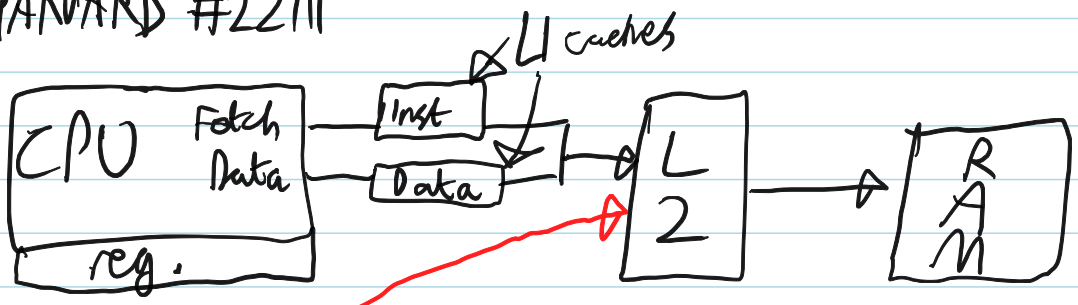
(!) Data unused
(!) Larger RAM accesses take longer
(!) Displaces other useful data
] — if line becomes too big [less temporal but more spatial

- too small ⟶ less spatial but more temporal

Separate I&D caches (!) Complete (!)

- Fetch per instruction
- Data fetch every 3 instructions

#HARVARD #22111



MULTIPLE level caches

- Bigger caches → Higher hitrates BUT slower speeds!
  - Solution: decrease number of cycles per
- L2 cache bigger ☹ slower than L1
  - BUT L2 still faster than RAM
- If missrate in L2 and L1 were only 2% each:
  - memory accessrate = 0.04% 👍
- inclusive cache: can't have data in level (x) if data isn't at lower level (x-1)
- L2 usually shared between I and D caches

8MB cache → 2^{23} bytes

32 bit/4byte words, each line of entry = 8 words

☹ line = 32 bytes = 2^5 bytes

∴ entries ≈ $2^{23-5} = 2^{18}$ entries (≈256k entries)

- Looking at slide:

Tag	Index	Word ID	Align
9	18	3	2

size and format of address

- $2 \times ((1) + \text{number of cycles})$

  (1) you need to determine if INITIALLY the cache has a miss or hit

h_i = hit rate at level i — t_i = technology based access time at level i

$m_i = 100\% - h_i$ — T_i = perceived (what we calculate)

$$T_i = t_i \cdot h_i + m_i \cdot (t_i + T_{i+1})$$

$$= t_i + m_i T_{(i+1)}$$ (You do the math)