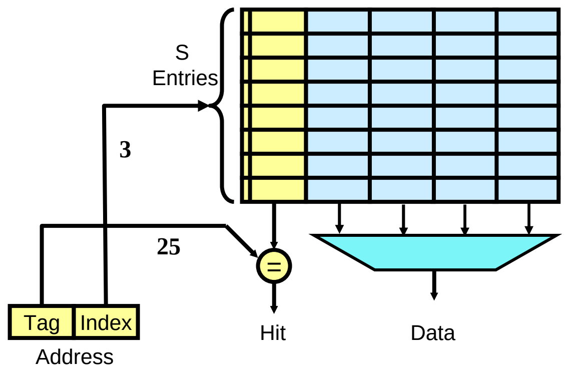
Lecture 17: Cache wrap-up and Memory Intro

- Last Time:
  - Direct-mapped and associative caches
  - Replacement and write policies
- Today
  - *Cache performance optimizations*
  - Introduction to memories and addressing

Cache Review

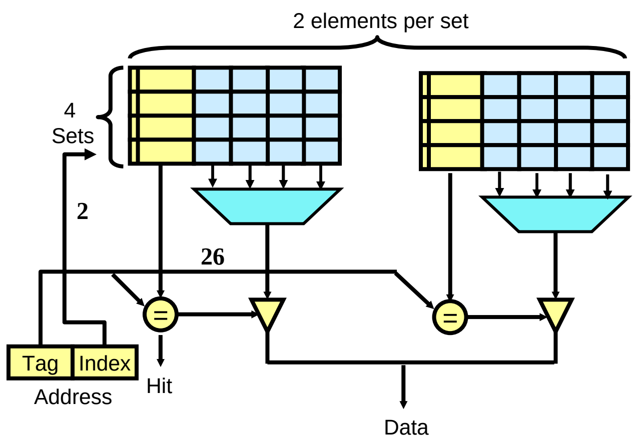
- Locality: Spatial and Temporal
- Terms:
  - block size, associativity

Finding a Block: Direct-Mapped



With cache capacity = 8 blocks

Finding A Block: 2-Way Set-Associative



Three kinds of cache misses

- Compulsory misses
  - First time data is accessed
- Capacity misses
  - Working set larger than cache size
- Conflict misses
  - One set fills up, but room in other sets

How Do We Improve Cache Performance?

$$AMAT = t_{hit} + p_{miss} \bullet penalty_{miss}$$

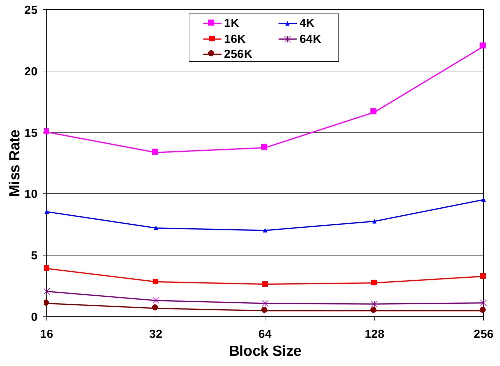
How Do We Improve Cache Performance?

$$AMAT = t_{hit} + p_{miss} \bullet penalty_{miss}$$

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time

Reducing Miss Rate: Increase Block Size

- Fetch more data with each cache miss
  - 16 bytes ⇒ 64, 128, 256 bytes!
  - Works because of Locality (spatial)

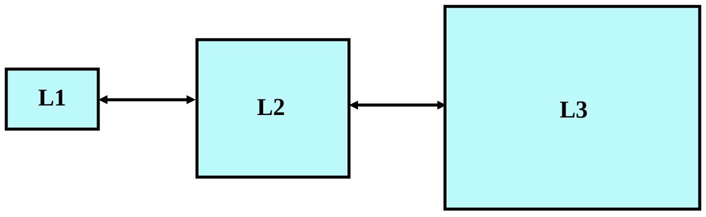


Reducing Miss Rate: Increase Associativity

- Reduce conflict misses
- Rules of thumb
  - 8-way = fully associative
  - Direct mapped size N = 2-way set associative size N/2
- But!
  - Size N associative is larger than Size N direct mapped
  - Associative typically slower that direct mapped (t_{hit} larger)

Reduce Miss Penalty: More Cache Levels

- Average access time =
  $HitTime_{L1} + MissRate_{L1} * MissPenalty_{L1}$
- $MissPenalty_{L1}$ =
  $HitTime_{L2} + MissRate_{L2} * MissPenalty_{L2}$
- etc.
- Size/Associativity of higher level caches?



Reduce Miss Penalty: Transfer Time

- Wider path to memory
  - Transfer more bytes/cycle
  - Reduces total time to transfer block
- Two ways to do this:
  - Wider path to each memory
  - Separate paths to multiple memories ("multiple memory banks")

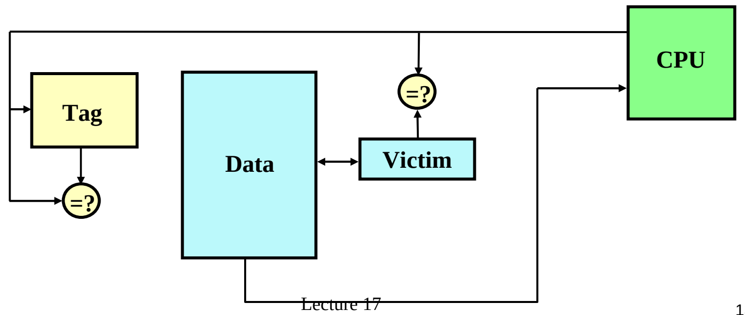
Reducing Hit Time

- Make Caches small and simple
  - Hit Time = 1 cycle is good (3.3ns!)
  - L1 - low associativity, relatively small
- Even L2 caches can be broken into sub-banks
  - Can exploit this for faster hit time in L2

Reducing Miss Rate: Use a "Victim" Cache

- Small cache (< 8 entries)
  - Jouppi 1990
  - Accessed in parallel with main cache
  - Captures conflict misses

Set	1W	2W	3W	4W	
0	X				
1	X	X			
2	X				
3	X	X	X	X	X
4	X				
5	X				
6					
7	X	X			

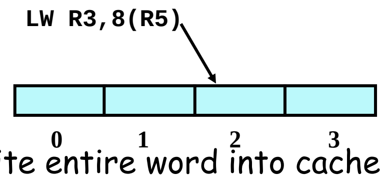


Reducing Miss Rate: Prefetching

- Fetching Data that you will probably need
- Instructions
  - Alpha 21064 on cache miss
    - Fetches requested block intro instruction stream buffer
    - Fetches next sequential block into cache
- Data
  - Automatically fetch data into cache (spatial locality)
  - Issues?
- Compiler controlled prefetching
  - Inserts prefetching instructions to fetch data for later use
  - Registers or cache

Reduce Miss Penalty: Deliver Critical word first

- Only need one word from block immediately

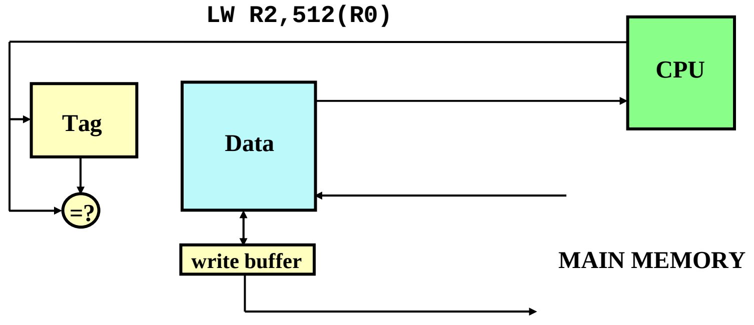


- Don't write entire word into cache first
  - Fetch word 2 first (deliver to CPU)
  - Fetch order: 2 3 0 1

Reduce Miss Penalty: Read Misses First

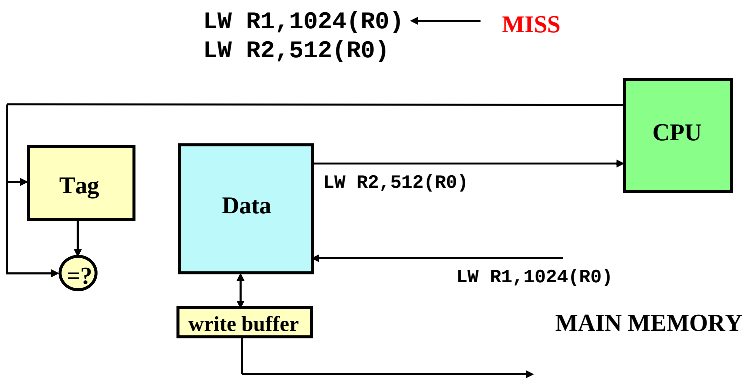
- Let reads pass writes in Write buffer

```
SW 512(R0),R3
LW R1,1024(R0)
LW R2,512(R0)
```



Reduce Miss Penalty: Lockup Free Cache

- Let cache continue to function while miss is being serviced

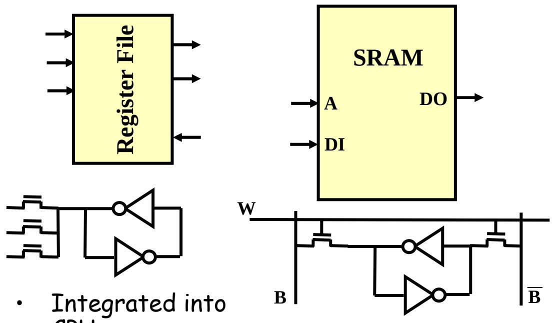


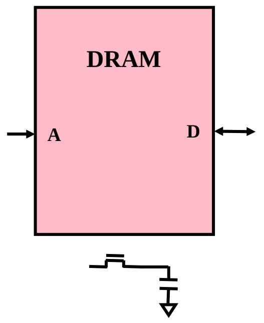
Memory systems

Memory Systems

- Memory Technology
  - SRAM, DRAM
- Higher order memory functions
  - Relocation, protection
- Virtual memory

Typical Memory Technologies





- Integrated into CPU
- Fast, many ports

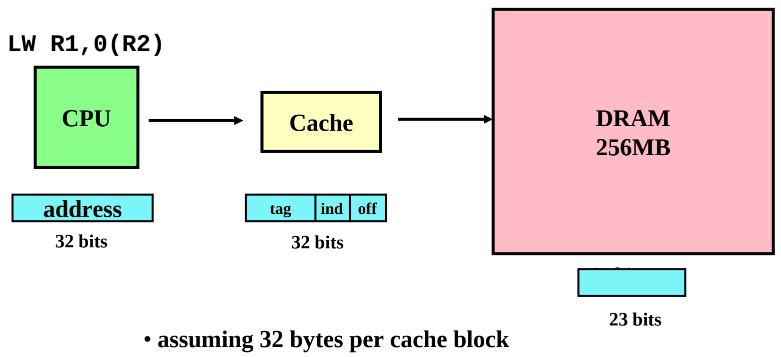
- Caches
  - On chip L1
  - On/off chip L2
  - Off chip L3
- More bits, fewer ports

- Main Memory
- Very dense
- Slower to access, one port

SRAM vs. DRAM

- SRAM
  - 4MBit capacity
  - Low latency for access
  - Storage cells are self-restoring
  - 4 times cost per bit (vs DRAM)
- DRAM
  - 512Mbit capacity
  - High latency for access
  - Reads destroy data
    - Must write data back
    - Refresh periodically
  - Lower cost per bit

Physical Memory Addressing

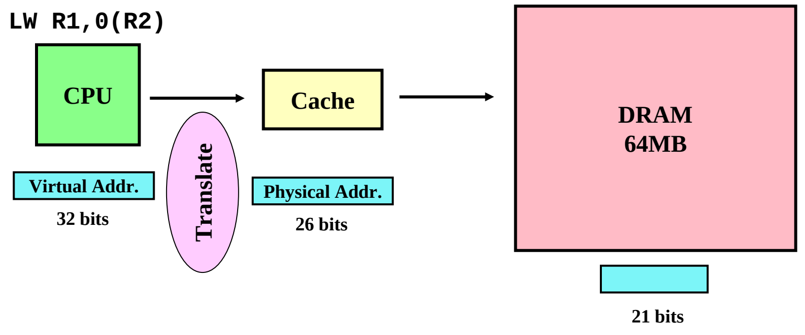


• assuming 32 bytes per cache block

What if?

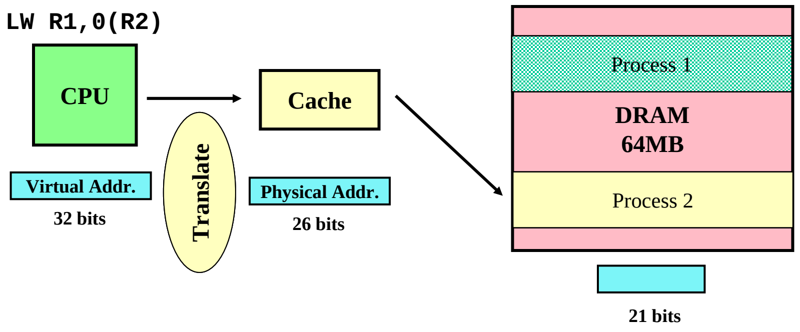
- A program is loaded into different places in memory each time it runs?
  - Relocation
- A program wants to use more memory than physically exists?
  - Page to disk
- We want to switch between multiple programs that use different data?
  - Protection

A Load to Virtual Memory



- Translate from virtual space to physical space
  - VA $\Rightarrow$ PA
  - May need to go to disk

A Load to Virtual Memory



- Both programs can use the same set of addresses!
  - Change translation tables to point same VA to different PA for different programs

Summary

- Cache performance optimizations
- Virtual memory provides
  - Illusion of private memory system for each process
  - Protection
  - Relocation in memory system
  - Demand paging
- But – page tables can be large
  - Motivates: paging page tables, multi-level tables, inverted page tables
- Next time
  - Virtual memory in detail