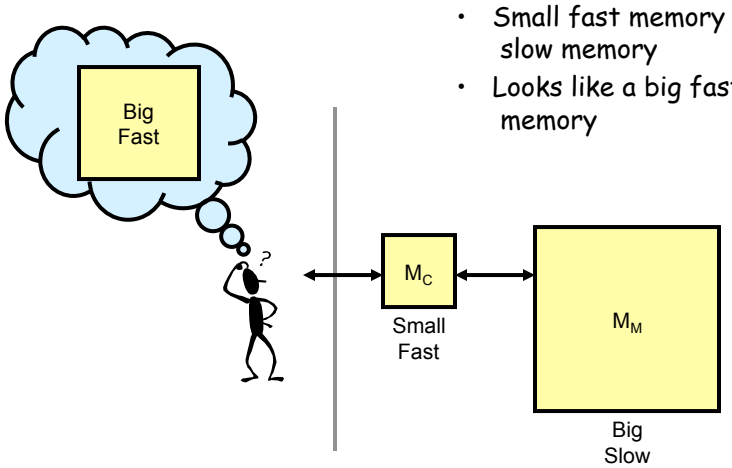
Lecture 15: Cache Memories

- Last Time
  - Exploiting Instruction Level Parallelism
  - Multiple issue processors
  - Out-of-order execution
- Today
  - Take QUIZ 11 over P&H 5.1-3, 5.5, before 11:59pm today
  - Read 5.7-10 for 3/23
  - Homework 6 due Thursday March 25, 2010
  - The Memory Hierarchy
  - Why and how caches work

Cache Memory Theory



- Small fast memory + big slow memory
- Looks like a big fast memory

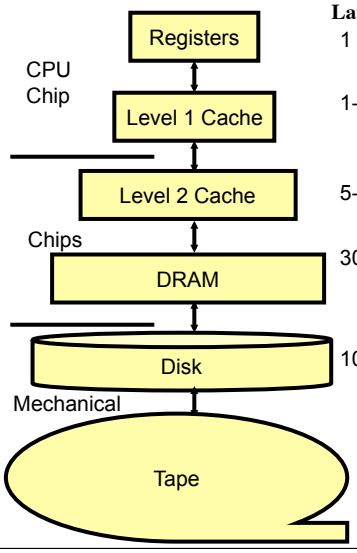
Bookshelf analogy

- Lots of books on shelves
- A few books on my desk
- One book I'm reading at this moment

- Shelves = main memory
- Desk = cache

- Book = block
- Page in book = memory location

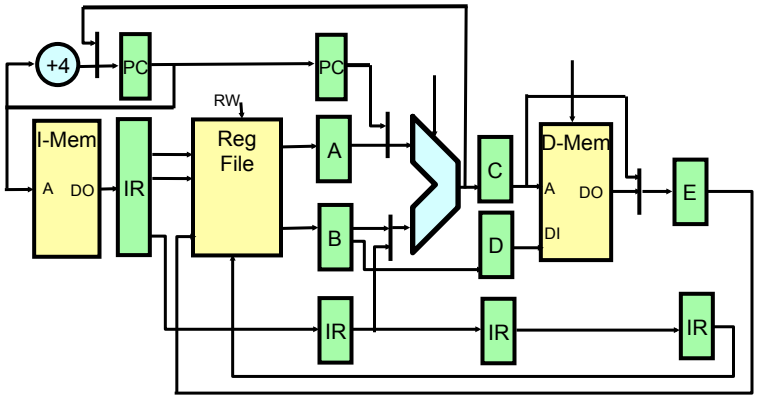
The Memory Hierarchy



	Latency	Bandwidth	
Registers	1 cyc	3-10 words/cycle < 1KB	compiler managed
Level 1 Cache	1-3cy	1-2 words/cycle 32KB -1MB	hardware managed
Level 2 Cache	5-10cy	1 word/cycle 1MB - 4MB	hardware managed
DRAM	30-100cy	0.5 words/cycle 64MB - 4GB	OS managed
Disk	10^6-10^7cy	0.01 words/cycle 4GB+	OS managed

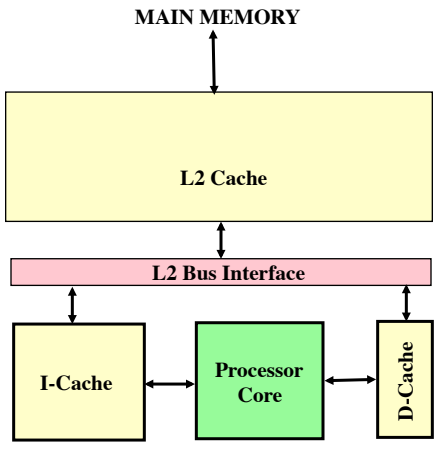
4

Where Does the Memory Hierarchy Fit In?



Typical Cache Organization

Alpha 21264: 64KB I-Cache
64KB D-Cache
> 1MB L2 Cache

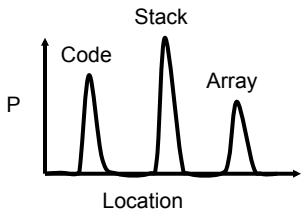


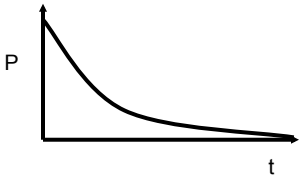
Memory System Overview

- Memory Hierarchies
  - Latency/Bandwidth/Locality
  - Caches
    - Principles - why does it work
    - Cache organization
    - Cache performance
    - Types of misses (the 3 Cs)
  - Main memory organization
    - DRAM vs. SRAM
    - Bank organization
    - Tracking multiple references
  - Trends in memory system design
- Logical Organization
  - Name spaces
  - Protection and sharing
  - Resource management
    - virtual memory, paging, and swapping
  - Segmentation

Why does it work? Program Locality of Reference

- Spatial Locality
  - likely to reference data *near* recent references
- Temporal Locality
  - likely to reference the same data that was referenced recently





Program Behavior

- Locality depends on type of program
- Many programs 'behave' well
  - small loop operating on data on stack
- Some programs don't
  - frequent calls to nearly random subroutines
  - traversal of large, sparse data set
    - essentially random data references with no reuse
- Most programs exhibit some degree of locality

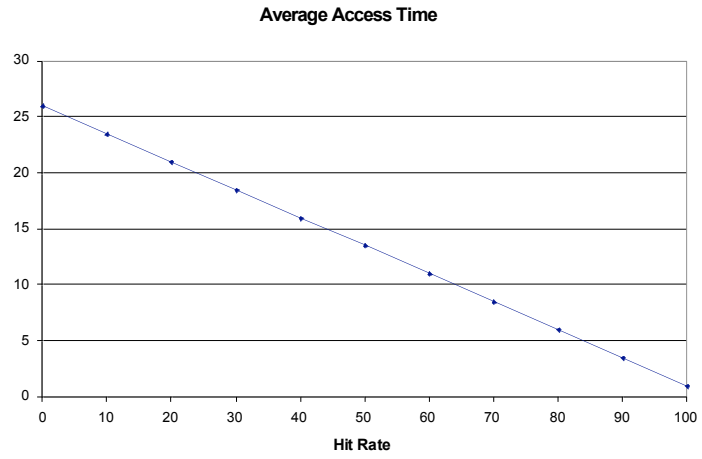
Example

What is the average memory access time?

- 70% of references hit in cache
- Cache hits take one cycle
- Main memory references take 25 cycles

$$AMAT = Latency_{Hit} + P(miss)*Latency_{Miss}$$

Impact of Hit Rate



Two kinds of "fast & small" memory

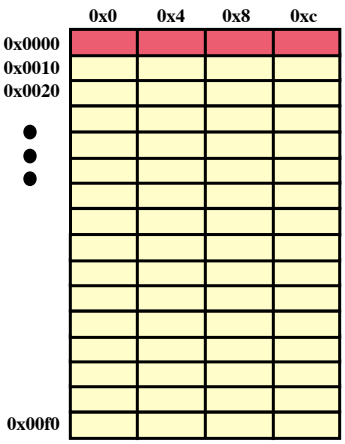
- Programmer manages it manually
  - Sometimes called a "scratchpad" memory
  - CELL processor uses this approach
- Hardware manages it automatically
  - Invisible to programmer
  - Referred to as a "cache"
  - Most CPUs use this approach
  - Easy for programmers; Hard for hardware

How does hardware keep track of what's in the fast memory (cache)?

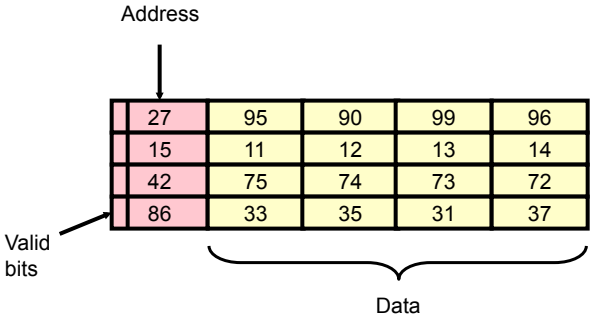
- How does it know what's in the cache 'now'?
- How does it decide what to add to the cache?
- How does it decide what to remove from the cache?
- How does it keep the cache consistent with the off-chip memory?

Cache Definitions

- Cache block (= cache line)
- Miss rate
  - Fraction of references not in cache
- Miss penalty
  - cycles to service a miss
- Index
  - Where to look, i.e., which lines could it be in?
- Tag
  - Is this cache line this address?
- Offset
  - Which word in the line?

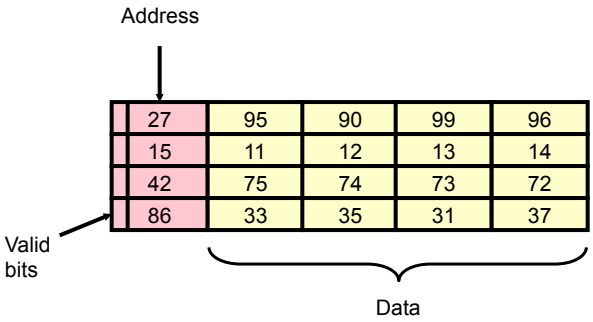


Cache Organization



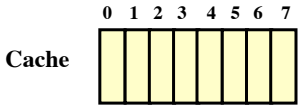
- Where does a block get placed?
- How do we find it?
- Which one do we replace when we bring in a new one?
- What happens on a write?

Cache Organization



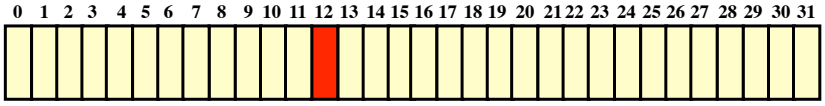
- What is the purpose of a valid bit?

Where Does a Block Go in the Cache?



- Word = 4 bytes
- Block = 1 word

Main Memory

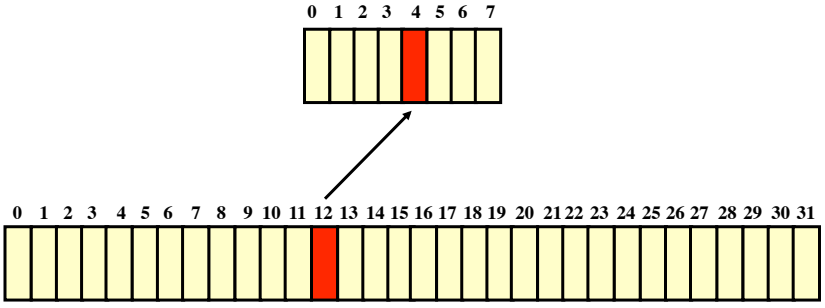


- Where do we put block 12?

Direct Mapped

- **Each block mapped to exactly 1 cache location**

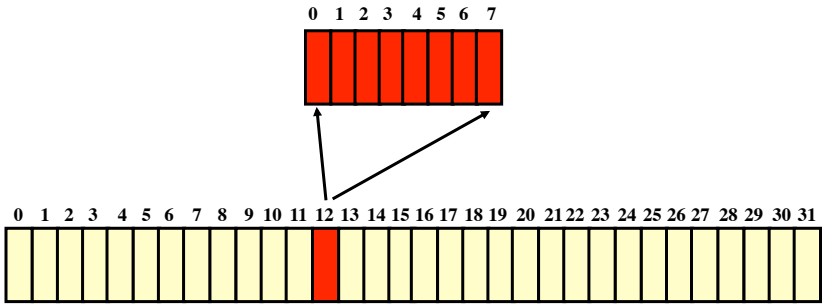
Cache location = (block address) MOD (# blocks in cache)



Fully Associative

- **Each block mapped to any cache location**

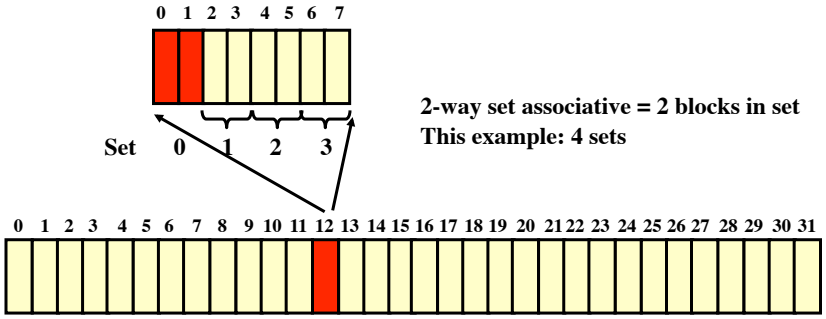
Cache location = any



Set Associative

- **Each block mapped to subset of cache locations**

Set selection = (block address) MOD (# sets in cache)

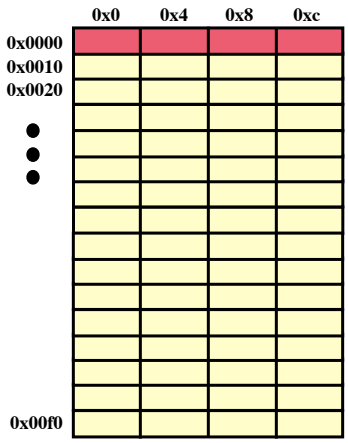


2-way set associative = 2 blocks in set
This example: 4 sets

More Cache Definitions

Miss type

- **Compulsory**: first reference
- **Capacity**: between two accesses to a single location, the program accesses more than a cache full of data
- **Conflict**: two accesses to same location, but an intervening access caused it to be replaced, and the program has **not** accessed a cache full of data

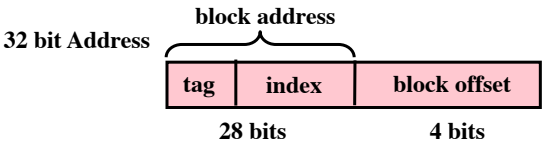


Taking advantage of Spatial Locality

- Instead of each block in cache being just 1 word, what if we made it 4 words?
- When we get our 1 word instruction or 1 word of data from memory to put in the cache, get the next 3 as well, because they are likely to be used soon!
- Need to add a way to choose which of the 4 words in the block we want when we go to cache… called block offset.

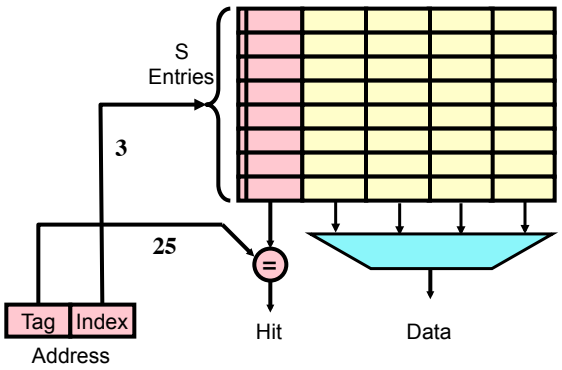
How do we use memory address to find block in the cache?

How Do We Find a Block in The Cache?

- Our Example:
  - Main memory address space = 32 bits (= 4GBytes)
  - Block size = 4 words = 16 bytes
  - Cache capacity = 8 blocks = 128 bytes
- 

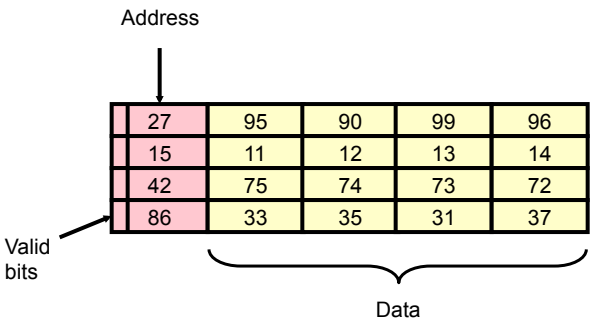
- index ⇒ which set
- tag ⇒ which data/instruction in block
- block offset ⇒ which word in block
- # tag/index bits determine the associativity
- tag/index bits can come from anywhere in block address

Finding a Block: Direct-Mapped



With cache capacity = 8 blocks

Cache Organization



- Where does a block get placed? Answered!
- How do we find it? Answered for Direct Map!
- Which one do we replace when we bring in a new one?
- What happens on a write?

Do something fun

- Next Time
  - Homework 6 is due March 25, 2010
  - Reading: P&H 5.7-10

Have a wonderful and relaxing Spring Break!