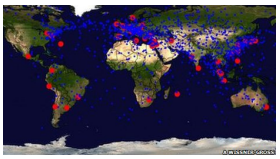
# CS 61C: Great Ideas in Computer Architecture

## *Lecture 12 – Memory Hierarchy/ Direct-Mapped Caches*

Instructors:
Mike Franklin
Dan Garcia
http://inst.eecs.berkeley.edu/~cs61c/fa11



## In the news…



**"Stock trades to exploit speed of light, says researcher"**

http://www.bbc.co.uk/news/science-environment-12827752

(from March 2011)

Financial institutions may soon change what they trade or where they do their trading because of the speed of light

…first solution, published in 2010, considered the various latencies in global fiber-optic links and mapped out where the optimal points for financial transactions to originate - midway between two major financial hubs to maximize the chance of "buying low" in one place and "selling high" in another.

**That means that out-of-the-way places - at high latitudes or mid-ocean island chains - could in time turn into global financial centers.**



## Agenda

- Context – Remember the Great Ideas!
- A Performance Teaser
- Memory Hierarchy Overview
- The Principle of Locality
- Direct-Mapped Caches



## 6 Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy



## 6 Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy



## Defining Performance

- What does it mean to say X is faster than Y?

- 2009 Ferrari 599 GTB
  - 2 passengers, 11.1 secs for quarter mile (call it 10sec)
- 2009 Type D school bus
  - 54 passengers, quarter mile time? (let's guess 1 min)

http://www.youtube.com/watch?v=KwyCoQuhUNA

- *Response Time* or *Latency*: time between start and completion of a task (time to move vehicle ¼ mile)
- *Throughput* or *Bandwidth*: total amount of work in a given time (passenger-miles in 1 hour)

## Cloud Performance: Why Application Latency Matters

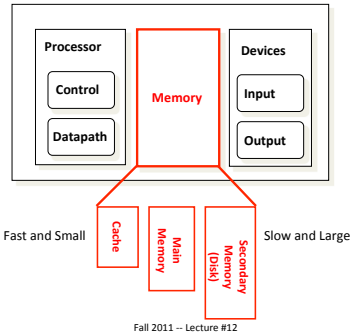
| Server Delay (ms) | Increased time to next click (ms) | Queries/ user | Any clicks/ user | User satisfac- tion | Revenue/ User |
|---|---|---|---|---|---|
| 50 | -- | -- | -- | -- | -- |
| 200 | 500 | -- | -0.3% | -0.4% | -- |
| 500 | 1200 | -- | -1.0% | -0.9% | -1.2% |
| 1000 | 1900 | -0.7% | -1.9% | -1.6% | -2.8% |
| 2000 | 3100 | -1.8% | -4.4% | -3.8% | -4.3% |

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Brutlag and Schurman 2009].

- Key figure of merit: application responsiveness
  - Longer the delay, the fewer the user clicks, the less the user happiness, and the lower the revenue per user



## Components of a Computer



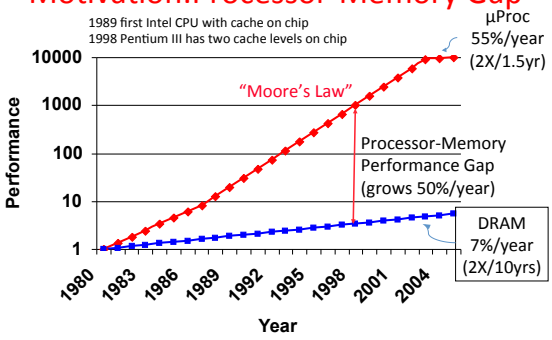



## Storage in a Computer

- Processor
  - holds data in register file: 100's of bytes (e.g., MIPS has 32 x 4 bytes)
  - Registers accessed on sub-nanosecond timescale
- Memory (a.k.a. "main memory")
  - More capacity than registers (~Gbytes)
  - Access time ~50-100 ns
  - Hundreds of clock cycles per memory access!



## Motivation:Processor-Memory Gap





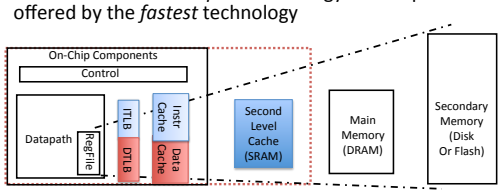

## Memory Hierarchy

- If level closer to Processor, it is:
  - Smaller
  - Faster
  - More expensive
  - Retains a subset of the data from the lower levels (e.g., contains most recently used data)
- Processor accesses data out of highest levels
- Lowest Level (usually disk) contains all available data (does it go beyond the disk?)



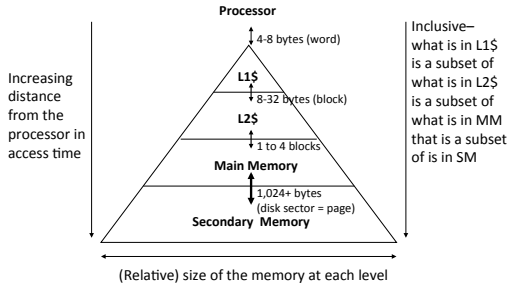
## Typical Memory Hierarchy

- The Trick: present processor with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology




| | | | | | |
|---|---|---|---|---|---|
| **Speed (#cycles):** | ½'s | 1's | 10's | 100's | 10,000's |
| **Size (bytes):** | 100's | 10K's | M's | G's | T's |
| **Cost:** | highest | | | | lowest |

## Characteristics of the Memory Hierarchy






## Cache ("$") Concept

- Mismatch between processor and memory speeds leads us to add a new level: a memory *cache*
- Implemented with same IC processing technology as the CPU, integrated on-chip: faster but more expensive than DRAM memory
- *Cache is a copy of a subset of main memory*
- Modern processors have separate caches for instructions and data, as well as several levels of caches implemented in different sizes and technologies (e.g., processor vs. SRAM)



## Why Does the Hierarchy Work?

- *Principle of Locality*: Programs access small portion of address space at any instant of time



## Library Analogy

- Writing a report on a specific topic.
  - E.g., works of J.D. Salinger
- While at library, check out books and keep them on desk.
- If need more, check them out and bring to desk.
  - But don't return earlier books since might need them
  - Limited space on desk; Which books to keep?
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries



## Two Types of Locality

- *Temporal Locality* (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon
  - ⇒ Keep most recently accessed data items closer to the processor
- *Spatial Locality* (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
  - ⇒ Move blocks consisting of contiguous words closer to the processor



## Why Does the Hierarchy Work?

- *Principle of Locality*: Programs access small portion of address space at any instant of time
- What program structures lead to temporal and spatial locality in code?
- In data?

## How is the Hierarchy Managed?

- registers ↔ memory
  - By compiler (or assembly level programmer)
- cache ↔ main memory
  - By the cache controller hardware
- main memory ↔ disks (secondary storage)
  - By the operating system (virtual memory)
  - Virtual to physical address mapping assisted by the hardware (TLB)
  - By the programmer (files)



## Cache Design Questions

1. How best to organize the memory blocks of the cache?
2. To which block of the cache does a given main memory address map?
   - Since the cache is a subset of memory, multiple memory addresses can map to the same cache location
3. How do we know which blocks of main memory currently have a copy in cache?
4. How do we find these copies quickly?



## Cache Basics: Direct Mapped

- Direct mapped
  - Each memory *block* is mapped to exactly one block in the cache
    - Many lower level blocks share a given cache block
  - Address mapping:
    - (*block* address) modulo (# of *blocks* in the cache)
    - *Tag* associated with each cache *block* containing the address information (the upper portion of the address) required to identify the *block* (to answer **Q3**)

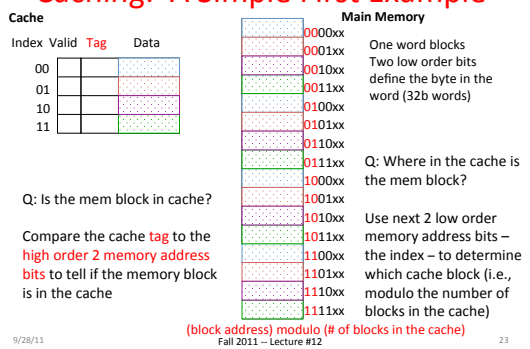


## Caching Terminology

- When reading memory, 3 things can happen:
  - cache **hit**:
    cache block is valid and contains proper address, so read desired word
  - cache **miss**: (Case 1 – cache block is empty/invalid)
    nothing in cache in appropriate block, so fetch from memory
  - cache **miss**: (Case 2- block **replacement**)
    wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

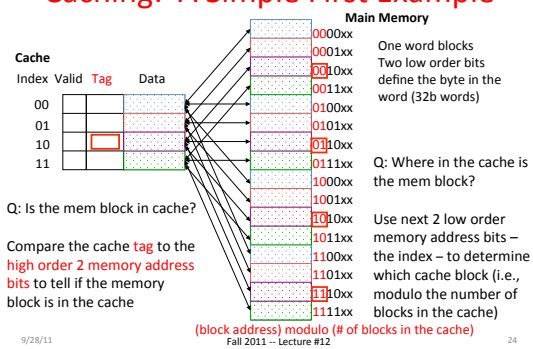


## Caching: A Simple First Example






## Caching: A Simple First Example



## TIO: Mapping the Memory Address

- Lowest bits of address (*Offset*) determine *which byte within a block* it refers to.
- Full address format:


Memory Address

- n-bit Offset means a block is how many bytes?
- n-bit Index means cache has how many blocks?



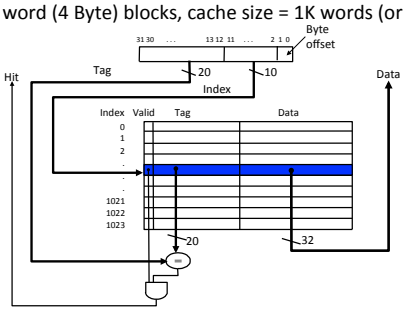
## TIO: Breakdown- Summary

- All fields are read as unsigned integers.
- Index
  - specifies the cache index (which "row"/block of the cache we should look in)
  - I bits <=> $2^I$ blocks in cache
- Offset
  - once we've found correct block, specifies which byte within the block we want (which "column" in the cache)
  - O bits <=> $2^O$ bytes per block
- Tag
  - the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to a given location



## Direct Mapped Cache Example

One word (4 Byte) blocks, cache size = 1K words (or 4KB)




*What kind of locality are we taking advantage of?*



## Summary

- Wanted: effect of a large, cheap, fast memory
- Approach: Memory Hierarchy
  - Successively lower levels contain "most used" data from next higher level
  - Exploits *temporal & spatial locality*
  - Do the common case fast, worry less about the exceptions (RISC design principle)
- Direct Mapped Caches as the first "programmer-invisible" layer of the memory hierarchy