

# ECE 152 Introduction to Computer Architecture

Caches and Memory Hierarchies  
Copyright 2008 Daniel J. Sorin  
Duke University

Slides are derived from work by  
Amir Roth (Penn) and Alvin Lebeck (Duke)  
Spring 2008

1

## Where We Are in This Course Right Now

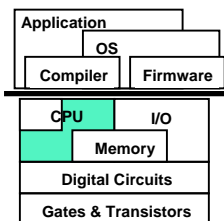
- So far:
  - We know how to design a processor that can fetch, decode, and execute the instructions in an ISA
  - We have assumed that memory storage (for instructions and data) is a magic black box
- Now:
  - We learn why memory storage systems are hierarchical
  - We learn about caches and SRAM technology for caches
- Next:
  - We learn how to implement main memory

© 2008 Daniel J. Sorin from Roth and Lebeck

ECE 152

2

## This Unit: Caches and Memory Hierarchies



- Memory hierarchy
  - Basic concepts
- SRAM technology
  - Transistors and circuits
- Cache organization
  - ABC
  - CAM (content associative memory)
  - Classifying misses
  - Two optimizations
  - Writing into a cache
- Some example calculations

© 2008 Daniel J. Sorin from Roth and Lebeck

ECE 152

3

## Readings

- Patterson and Hennessy
  - Chapter 7
  - Except 7.4 (for now)

© 2008 Daniel J. Sorin from Roth and Lebeck

ECE 152

4

## Storage

- We have already seen some storage implementations
  - **Individual registers**
    - For singleton values: e.g., PC, PSR
    - For  $\mu$ arch/transient values: e.g., in multi-cycle design
  - **Register File**
    - For architectural values: e.g., ISA registers
- What else is there?

## Storage Hierarchy

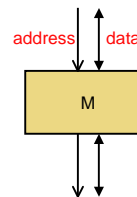
- Registers
  - Few locations: e.g., 32 4-byte words
  - Accessible directly via user-level ISA: multiple specifiers per insn
  - **Volatile** (values disappear when power goes off)
- **Memory**
  - Many (but finite) locations: e.g.,  $2^{32}$  bytes
  - Accessible indirectly via user-level ISA: one specifier per insn
  - Also volatile
- Disk
  - "Infinitely" many locations
  - Not accessible to user-level ISA (only via OS SYSCALL)
  - Non-volatile

## Storage Hierarchy

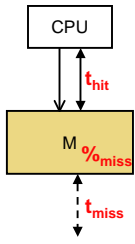
- Only fundamental component is ... **memory**
  - Registers not fundamental (e.g., memory-only and stack ISAs)
  - Only fundamental and desirable property of disks is non-volatility
    - But there is non-volatile memory technology (e.g., Flash)
- Registers vs. memory
  - Direct specification (fast) vs. address calculation (slow)
  - Few addresses (small & fast) vs. many (big & slow)
  - Not everything can be put into registers (e.g., arrays, structs)
- Memory vs. disk
  - Electrical (fast) vs. electro-mechanical (very slow)
  - Disk is so slow (relatively), it is considered I/O
- We will talk just about memory for **instructions** and **data**

## (CMOS) Memory Components

- Interface
  - N-bit **address** bus (on N-bit machine)
  - **Data** bus
    - Typically read/write on same data bus
  - Can have multiple **ports**: address/data bus pairs
  - Can be **synchronous**: read/write on clock edges
  - Can be **asynchronous**: untimed "handshake"
- Performance
  - Access time proportional to  $(\#ports) * \sqrt{(\#bits)}$
  - $\sqrt{(\#bits)}$ ? Proportional to max wire length
    - More about this a little later ...



## Memory Performance Equation



- For memory component M
    - Access**: read or write to M
    - Hit**: desired data found in M
    - Miss**: desired data not found in M
      - Must get from another (slower) component
    - Fill**: action of placing data in M
  - $\%_{miss}$  (miss-rate): #misses / #accesses
  - $t_{hit}$ : time to read data from (write data to) M
  - $t_{miss}$ : time to read data into M from lower level
  - Performance metric
    - $t_{avg}$ : average access time
- $$t_{avg} = t_{hit} + (\%_{miss} * t_{miss})$$

## Memory Hierarchy

$$t_{avg} = t_{hit} + \%_{miss} * t_{miss}$$

- Problem: hard to get low  $t_{hit}$  and  $\%_{miss}$  in one structure
  - Large structures have low  $\%_{miss}$  but high  $t_{hit}$
  - Small structures have low  $t_{hit}$  but high  $\%_{miss}$
- Solution: use a **hierarchy** of memory structures
- A very old (by computer standards) idea:

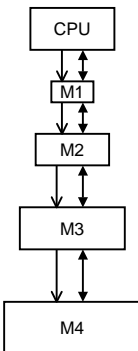
"Ideally, one would desire an infinitely large memory capacity such that any particular word would be immediately available ... We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has a greater capacity than the preceding but which is less quickly accessible."

Burks, Goldstone, and Von Neumann

"Preliminary discussion of the logical design of an electronic computing instrument"

IAS memo 1946

## Abstract Memory Hierarchy



- Hierarchy of memory components
  - Upper components (closer to CPU)
    - Fast ↔ Small ↔ Expensive
  - Lower components (further from CPU)
    - Slow ↔ Big ↔ Cheap
- Connected by buses
  - Which we will ignore for now
- Make average access time close to M1's
  - How?
  - Most frequently accessed data in M1
  - M1 + next most frequently accessed in M2, etc.
  - Automatically** move data up&down hierarchy

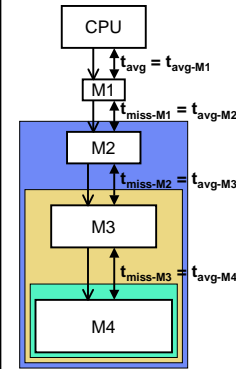
## Why Hierarchy Works I

- 10/90 rule (of thumb)**
  - For Instruction Memory:
    - 10% of static insns account for 90% of executed insns
    - Inner loops
  - For Data Memory:
    - 10% of variables account for 90% of accesses
    - Frequently used globals, inner loop stack variables

## Why Hierarchy Works II

- **Temporal locality**
  - Recently executed insns likely to be executed again soon
    - Inner loops (next iteration)
  - Recently referenced data likely to be referenced again soon
    - Data in inner loops, hot global data
  - Hierarchy can be **"reactive"**: move things up when accessed
- **Spatial locality**
  - Insns near recently executed insns likely to be executed soon
    - Sequential execution
  - Data near recently referenced data likely to be referenced soon
    - Elements in an array, fields in a struct, variables in frame
  - Hierarchy can be **"proactive"**: move things up speculatively

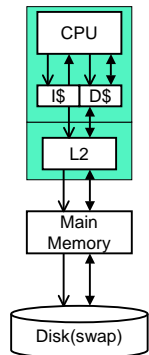
## Abstract Hierarchy Performance



How do we compute  $t_{avg}$  ?

$$\begin{aligned}
 &= t_{avg-M1} \\
 &= t_{hit-M1} + (\%_{miss-M1} * t_{miss-M1}) \\
 &= t_{hit-M1} + (\%_{miss-M1} * t_{avg-M2}) \\
 &= t_{hit-M1} + (\%_{miss-M1} * (t_{hit-M2} + (\%_{miss-M2} * t_{miss-M2}))) \\
 &= t_{hit-M1} + (\%_{miss-M1} * (t_{hit-M2} + (\%_{miss-M2} * t_{avg-M3}))) \\
 &= \dots
 \end{aligned}$$

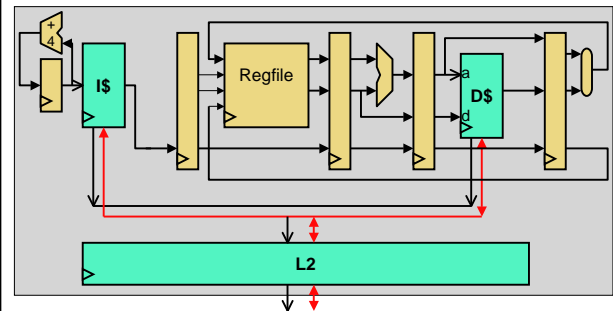
## Concrete Memory Hierarchy



- 1st level: **L1 I\$, L1 D\$** (L1 insn/data caches)
- 2nd level: **L2 cache**
  - Often on-chip, certainly on-package (with CPU)
  - Made of SRAM (same circuit type as CPU)
  - Managed in hardware
  - This unit of ECE 152
- 3rd level: **main memory**
  - Made of DRAM
  - Managed in software
  - Next unit of ECE 152
- 4th level: **disk (swap space)**
  - Made of magnetic iron oxide discs
  - Managed in software
  - Course unit after main memory

Note: some processors have off-chip L3\$ between L2\$ and memory

## Concrete Memory Hierarchy



- Chips today are 80+% cache by area → important!