

CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Instructors:
Randy H. Katz
David A. Patterson

<http://inst.eecs.Berkeley.edu/~cs61c/fa10>

Agenda

- Memory Hierarchy Overview
- Administrivia
- Technology Break
- Direct-Mapped Caches

Agenda

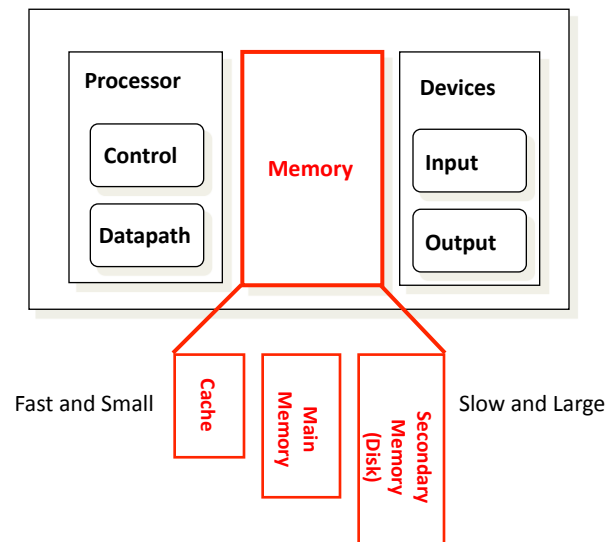
- Memory Hierarchy Overview
- Administrivia
- Technology Break
- Direct-Mapped Caches

9/30/10

Fall 2010 -- Lecture #14

3

Components of a Computer



9/30/10

Fall 2010 -- Lecture #14

4

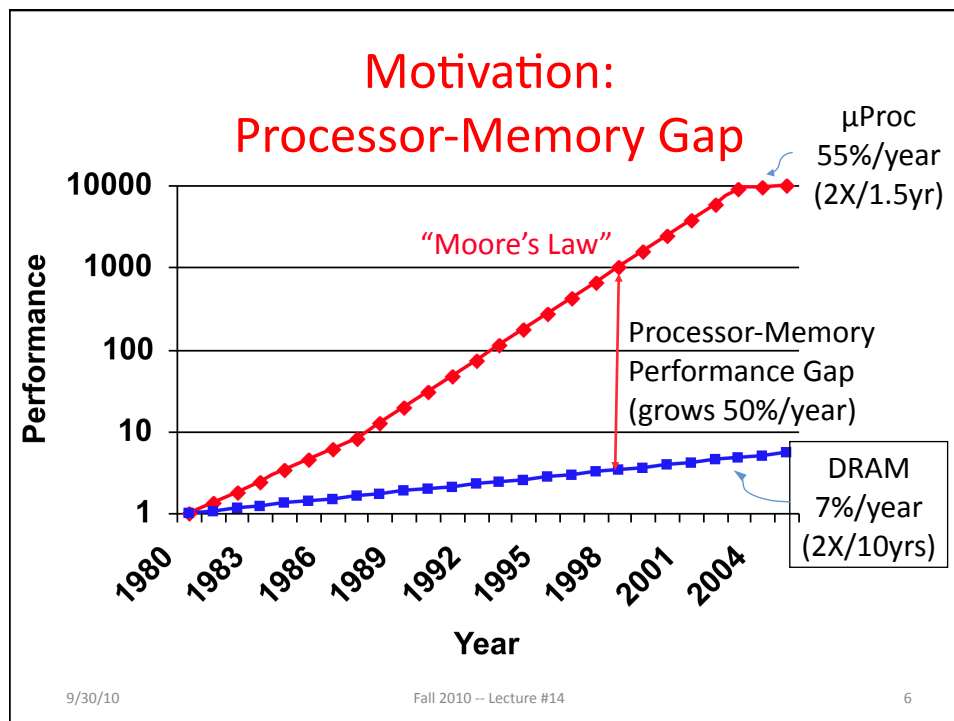
Computer Memories

- Processor
 - Holds data in register file (typically 32-64 x 4 bytes)
 - Registers accessed on <ns timescale
 - Programmer/compiler visible
- Memory (aka “main memory”)
 - More capacity than registers (~Gbytes)
 - Access time ~50-100 ns (100+x slower)
 - Hundreds of clock cycles per memory access
 - Managed automatically by hardware and operating system, though programmer can influence performance (our final project this semester!)
- “Disk” (aka “secondary storage”)
 - HUGE capacity (virtually limitless)
 - VERY slow in comparison: runs >ms (10-100x slower)
 - Emerging role of semiconductor-based solid state disk

9/30/10

Fall 2010 -- Lecture #14

5



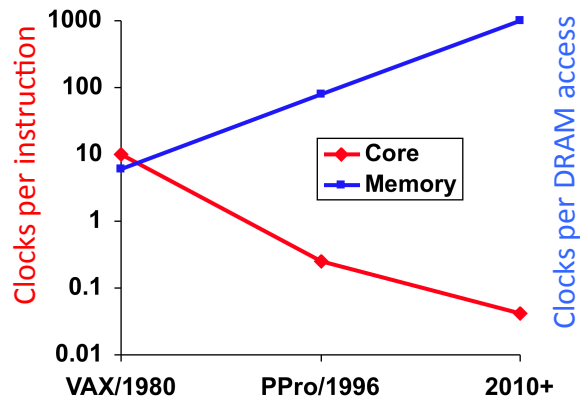
9/30/10

Fall 2010 -- Lecture #14

6

Aka “Memory Wall”

- Processor vs. DRAM speed disparity continues to grow
 - One of those tech trends that just keeps getting worse



9/30/10

Fall 2010 -- Lecture #14

7

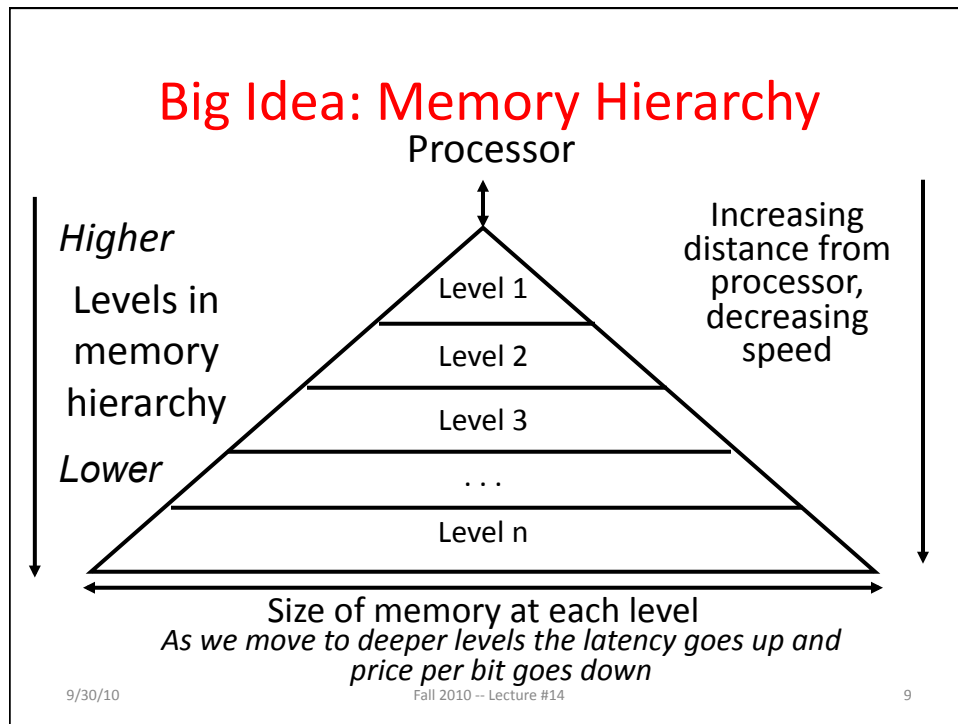
Big Idea: Memory Hierarchy

- Large memories are slow and fast memories are small
- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
 - With hierarchy
 - With parallelism

9/30/10

Fall 2010 -- Lecture #14

8

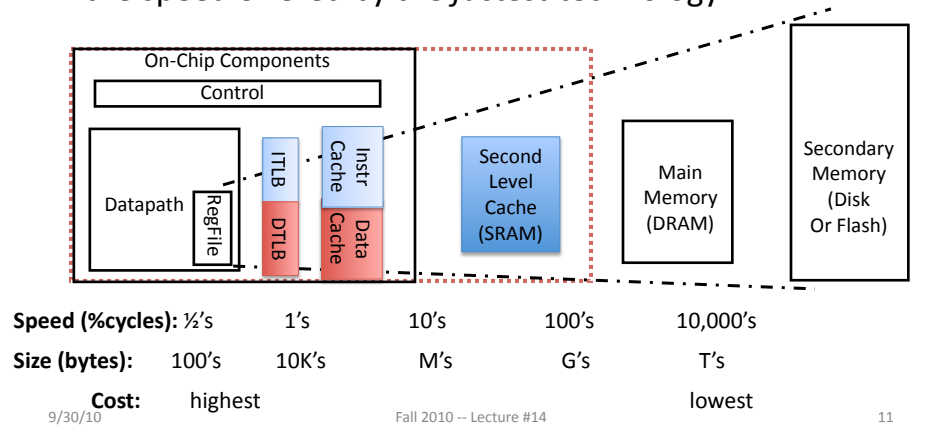


Memory Hierarchy

- 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)
- Lowest Level (usually disk) contains all available data (does it go beyond the disk?)
- Memory Hierarchy presents the processor with the illusion of a very large & fast memory

Typical Memory Hierarchy

- **Principle of locality** presents programmer with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



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)

Memory Hierarchy Technologies

- Caches use SRAM for speed and technology compatibility
 - Fast (typical access times of 0.5 to 2.5 ns)
 - Low density (6 transistor cells), higher power, expensive (\$2000 to \$5000 per GB in 2008)
 - Static: content will last “forever” (as long as power is left on)
- Main memory uses DRAM for size (density)
 - Slower (typical access times of 50 to 70 ns)
 - High density (1 transistor cells), lower power, cheaper (\$20 to \$75 per GB in 2008)
 - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
 - Consumes 1% to 2% of the active cycles of the DRAM
 - Addresses divided into 2 halves (row and column)
 - RAS or Row Access Strobe triggering the row decoder
 - CAS or Column Access Strobe triggering the column selector

9/30/10

Fall 2010 -- Lecture #14

13

Big Idea: 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

9/30/10

Fall 2010 -- Lecture #14

14

Agenda

- Memory Hierarchy Overview
- **Administrivia**
- Technology Break
- Caches

9/30/10

Fall 2010 -- Lecture #14

15

Midterm!

- Lab #3a: Cache Blocking
- HW #3: Posted, Due SUNDAY@23:59:59
- Exam Review on Monday, 100 GPB 6-8 PM
- NO lecture next Wednesday, 6 October
- Exam, 6-9 PM, 1 Pimentel
 - Closed book, notes
 - No calculator
 - One 8.5" x 11" crib sheet

9/30/10

Fall 2010 -- Lecture #14

16

Agenda

- Memory Hierarchy Overview
- Administrivia
- **Technology Break**
- Direct-Mapped Caches

9/30/10

Fall 2010 -- Lecture #14

17

Agenda

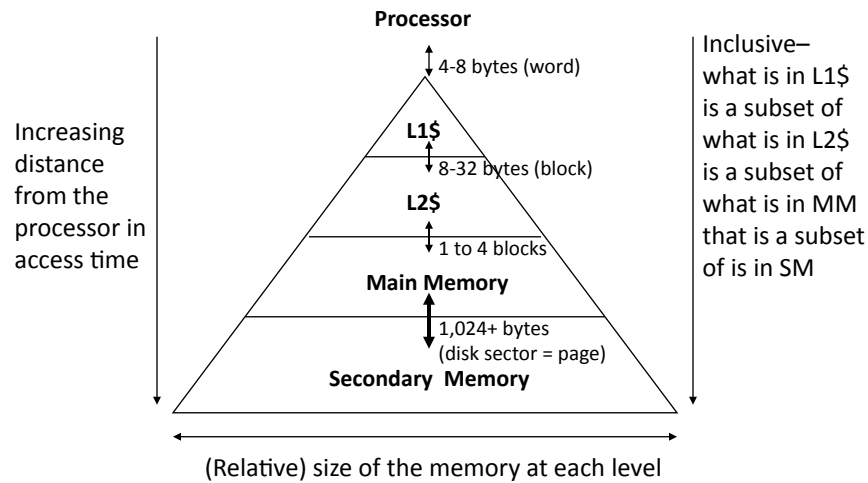
- Memory Hierarchy Overview
- Administrivia
- Technology Break
- **Direct-Mapped Caches**

9/30/10

Fall 2010 -- Lecture #14

18

Characteristics of the Memory Hierarchy



How is the Hierarchy Managed?

- registers \leftrightarrow memory
 - By compiler (or assembly level programmer)
- cache \leftrightarrow main memory
 - By the cache controller hardware
- main memory \leftrightarrow 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

- How best to organize the memory blocks of the cache?
- 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
- How do we know which blocks of main memory currently have a copy in cache?
- How do we find these copies quickly?

9/30/10

Fall 2010 -- Lecture #14

21

Cache Basics: Direct Mapped

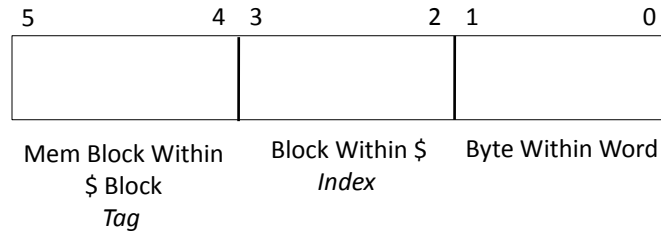
- Direct mapped
 - Each memory *block* is mapped to exactly one block in the cache
 - Many lower level blocks share a given cache block
 - Address mapping:
 - $(\text{block address}) \bmod (\# \text{ of blocks in the cache})$
 - *Tag* associated with each cache *block* containing the address information (the upper portion of the address) required to identify the *block* (to answer Q1)

9/30/10

Fall 2010 -- Lecture #14

22

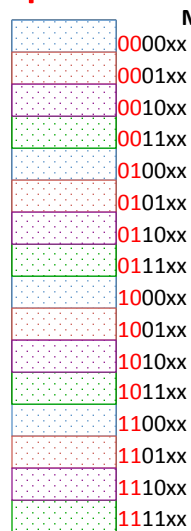
Mapping the Memory Address



- Byte addresses
- 16 Words, 4 Bytes/Word, 64 Byte Memory
 - Byte Memory: Mem[0:63]<7:0>
 - Memory Blocks: Mem[0:15]<31:0>
- Cache
 - Word-sized blocks
 - Four blocks in cache
 - (4 words in memory map to each block in cache)

Caching: A Simple First Example

Cache			
Index	Valid	Tag	Data
00			
01			
10			
11			



Main Memory
 One word blocks
 Two low order bits define the byte in the word (32b words)

Q: Is the mem block in cache?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

Q: Where in the cache is the mem block?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

Caching: A Simple First Example

Cache

Index	Valid	Tag	Data
00			
01			
10			
11			

Q: Is the mem block in cache?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

Main Memory

One word blocks
Two low order bits define the byte in the word (32b words)

Q: Where in the cache is the mem block?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

9/30/10 Fall 2010 -- Lecture #14 25

Direct Mapped Cache

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0

1

2

3

4

3

4

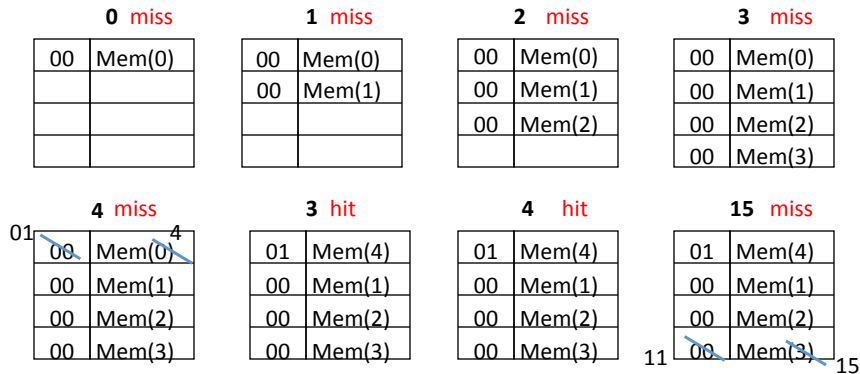
15

9/30/10 Fall 2010 -- Lecture #14 26

Direct Mapped Cache

- Consider the main memory word reference string

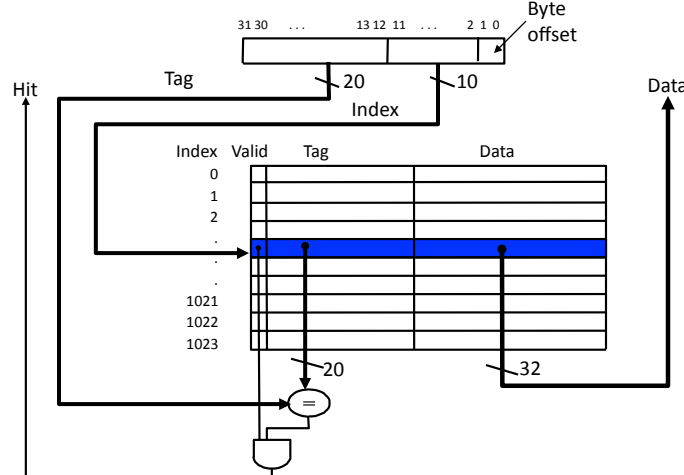
Start with an empty cache - all blocks initially marked as not valid



- 8 requests, 6 misses

Direct Mapped Cache Example

- One word 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