CS/ECE 5381/7381 Computer Architecture Spring 2023

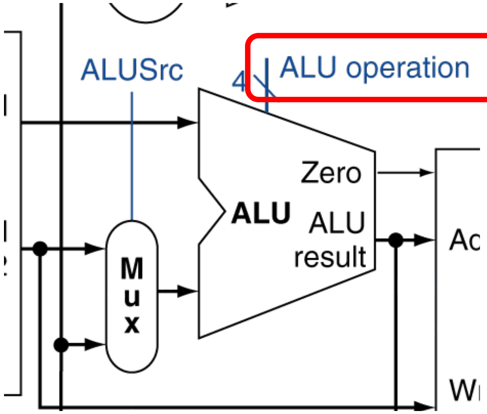
Dr. Manikas

Computer Science

Lecture 14: Mar. 21, 2023

Project 4

- Due next **Tues., Mar. 28** (11:59 pm)
- Cadence Xcelium tool
  – Used to develop and test Verilog code
- Verilog
  – Hardware Description Language
  – Used to construct and simulate computer hardware
- Assignment:
  – Run tool on simple MIPS ALU design



Simple MIPS ALU – only does two functions.

Use Xcelium tool to test this ALU, using the provided test bench

ALU control	Function	Description
0001	OR	Bitwise OR
0111	set-on-less-than	True if A < B, false otherwise

Review of Memory Hierarchy

(Appendix B, Hennessy and Patterson)

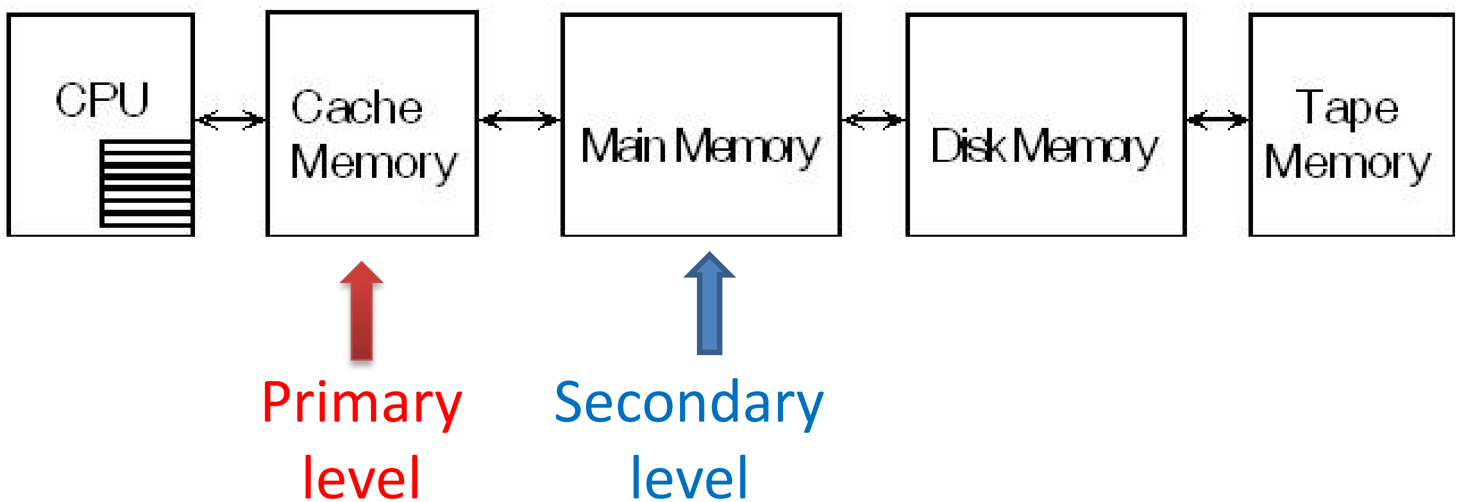
Note: some course slides adopted from publisher-provided material

Outline

- B.1 Introduction
- B.2 Cache Performance
- B.3 Basic Cache Optimizations
- B.4 Virtual Memory

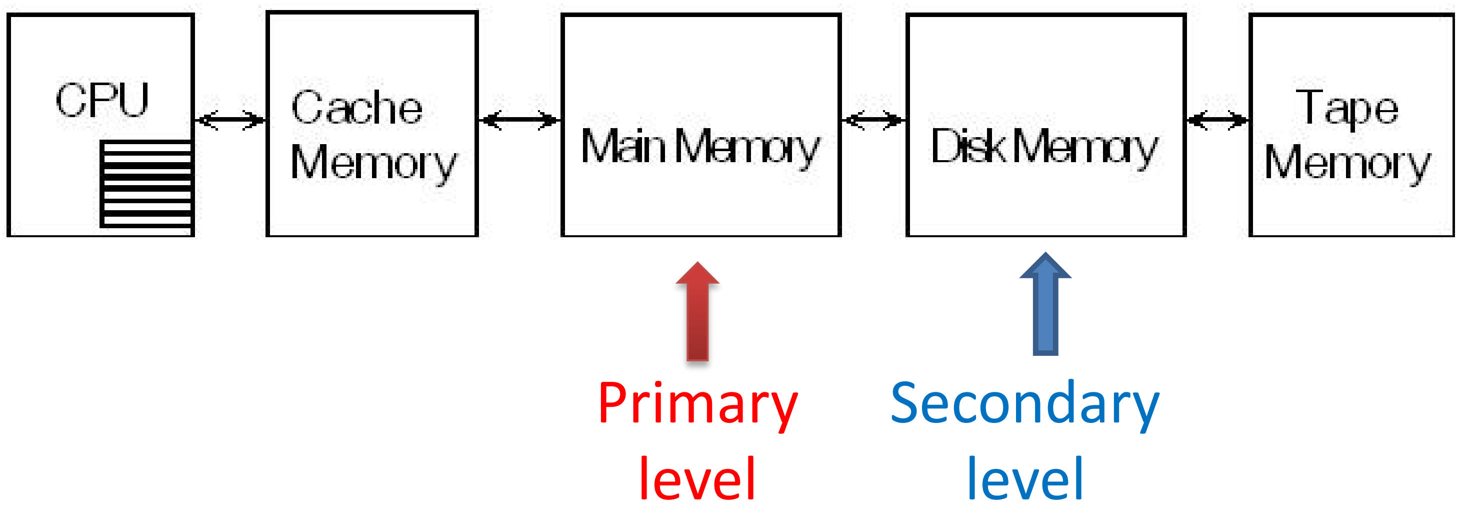
Overview

- Recall memory hierarchy
  - Cache = intermediate memory between CPU and main memory



Overview

- Virtual Memory (VM)
  - Main Memory acts like a “cache” for Disk Memory



Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- E.g., assume that you have Word and Excel open on your laptop
  - These programs and the data files (document, spreadsheet) are *permanently* stored in secondary storage (disk drive)
- However, when the programs are active, **copies** of the program and data are stored and run in main memory
  - since main memory is faster than secondary memory

Virtual Memory

- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page
  - VM translation "miss" is called a page fault

VM Page Sizes

- Page sizes are usually larger than cache block sizes
  - since it takes longer to access secondary memory (disk drive), we want to grab a large chunk of instructions/data for each access.
- Typical page sizes are 1K to 16K bytes
  - recall cache block size may be 8 or 16 bytes

Virtual Memory Addressing

- Addressing VM is somewhat similar to cache addressing
- Recall for fully associative memory example: we had a 16-bit address word
  - Cache block size was 8 B, so 3 bits to address each byte in the block
  - Remaining 13 bits used for *tag* field to identify specific block address in main memory, and determine if copy located in cache

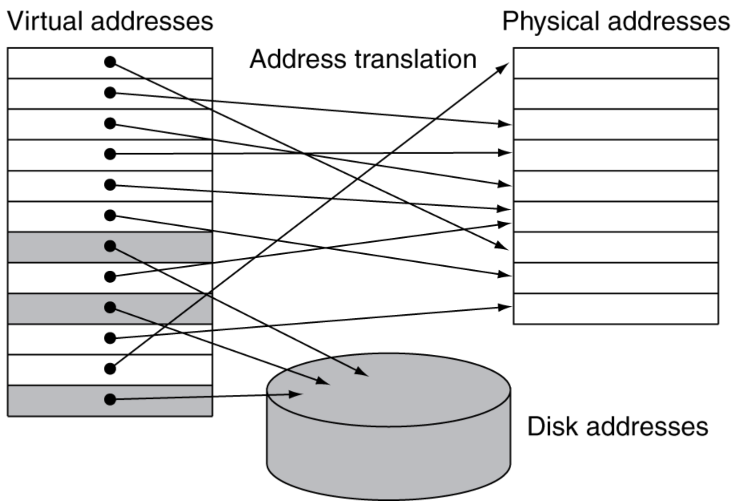
VM Addressing

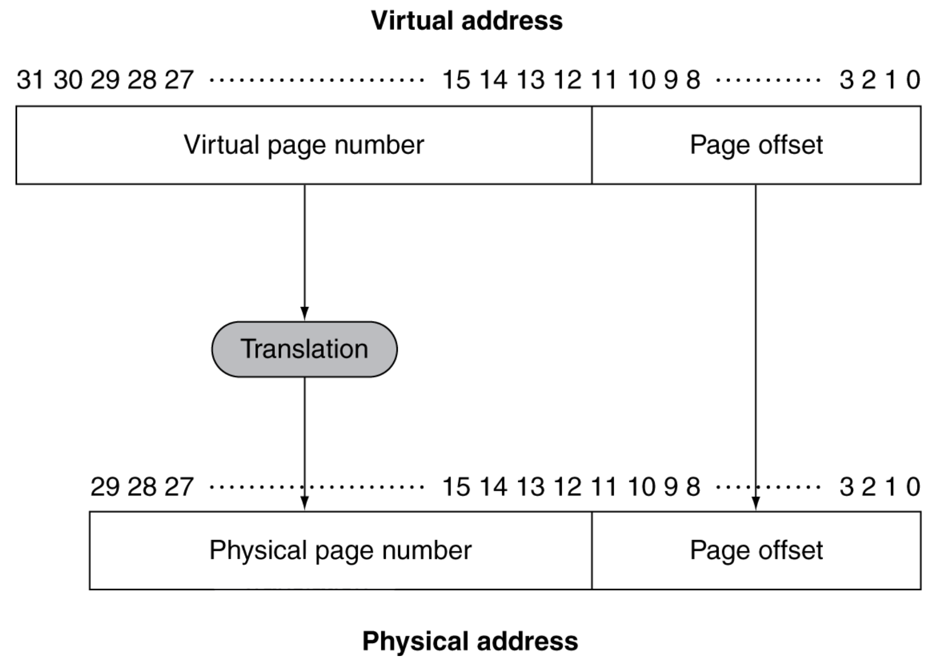
- As an example, assume we have a 32-bit address word, and VM with page size of 4KB
- 4KB = $2^2 2^{10}$ so we need 12 bits to address each byte in the page.
  - This is called **page offset** for VM - similar to byte address field in cache.
- We have 20 bits left over – used for page addressing

VM Addressing

- The CPU issues a **virtual address**:
  - contains **virtual page number** and **page offset** fields
- An address translator translates the *virtual page number* to a **physical page number**
- The *physical page number* and *offset fields* combine to form a **physical address**
- The *physical address* can either be main memory (**page hit**) or disk memory (**page miss** or **page fault**)

Address Translation





VM Addressing

- Physical addresses are in the primary memory level, and disk addresses are in the secondary memory level
- Since primary level is small and fast, *virtual* addresses will tend to be larger than *physical* addresses
- For our example above, the *virtual* address is 32 bits (can address $2^2 2^{30}$ = 4 GB of data), but the *physical* address is 29 bits ($2^9 2^{20}$ = 512 MB)

Page Tables for VM Addressing

- VM approach similar to fully associative cache
- However, remember time penalties for searching fully associate cache
- **Page tables** are used to make the page search process more efficient

Page Table Example

Assume that we have 8 KB of virtual memory, where page size is 1 KB. Thus, the VM is divided into 8 pages.

1 KB = 2^{10} B = 1024 B, so we need 10 bits for our offset field for VM addressing

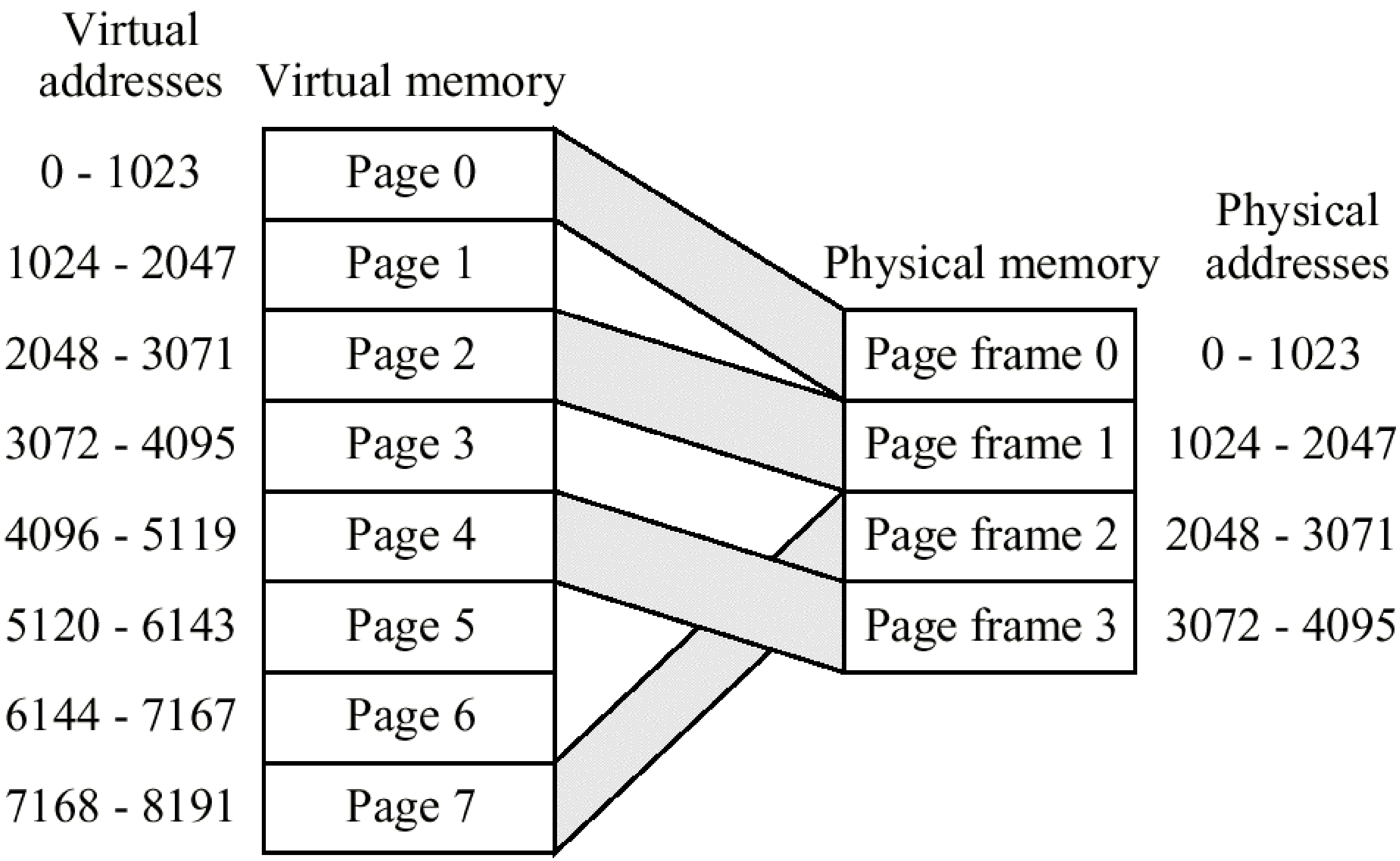
8 KB = $2^3 2^{10}$ B = 8(1024 B) = 8192 B for total VM space (13 bits for total virtual address)

Page Table Example

Now assume that we have 4K of physical memory available (this is our primary memory level). Thus, we can hold 4 pages at a time in our physical memory

4 KB = $2^2 2^{10}$ B = 4(1024 B) = 4096 B of physical memory (12 bits for total physical address)

Our mapping between virtual memory and physical memory will be the following:



Page table for this system is the following:

Page #	Present bit	Disk address	Page frame
0	1	01001011100	00
1	0	11101110010	xx
2	1	10110010111	01
3	0	00001001111	xx
4	1	01011100101	11
5	0	10100111001	xx
6	0	00110101100	xx
7	1	01010001011	10

Present bit:
0: Page is not in physical memory
1: Page is in physical memory

Page table notes:

1. There is a page table row for each page in VM. For our example, we have 8 pages in VM, so there are 8 rows in page table
2. Present bit similar to valid bit in cache
3. We have 4 page frames in physical memory, so 2 bits used to identify page frame (location in physical memory)
4. Disk address is location of page data in secondary memory

NOTE: the disk address field is 11 bits. $2^{11} = 2^1 2^{10}$ = 2K pages on hard drive

Our page size is 1KB = 2^{10} B, so 2K pages = $2^{11}2^{10}$ B = 2^{21} B = $2^1 2^{20}$ B = 2 MB hard drive (I told you this was a small example☺)

How do we use the page table to translate a virtual address into a physical address?

Assume we have the following virtual address (recall that this will be 13 bits since we are addressing 8KB):

$1345H = 0001\ 0011\ 0100\ 0101_2$

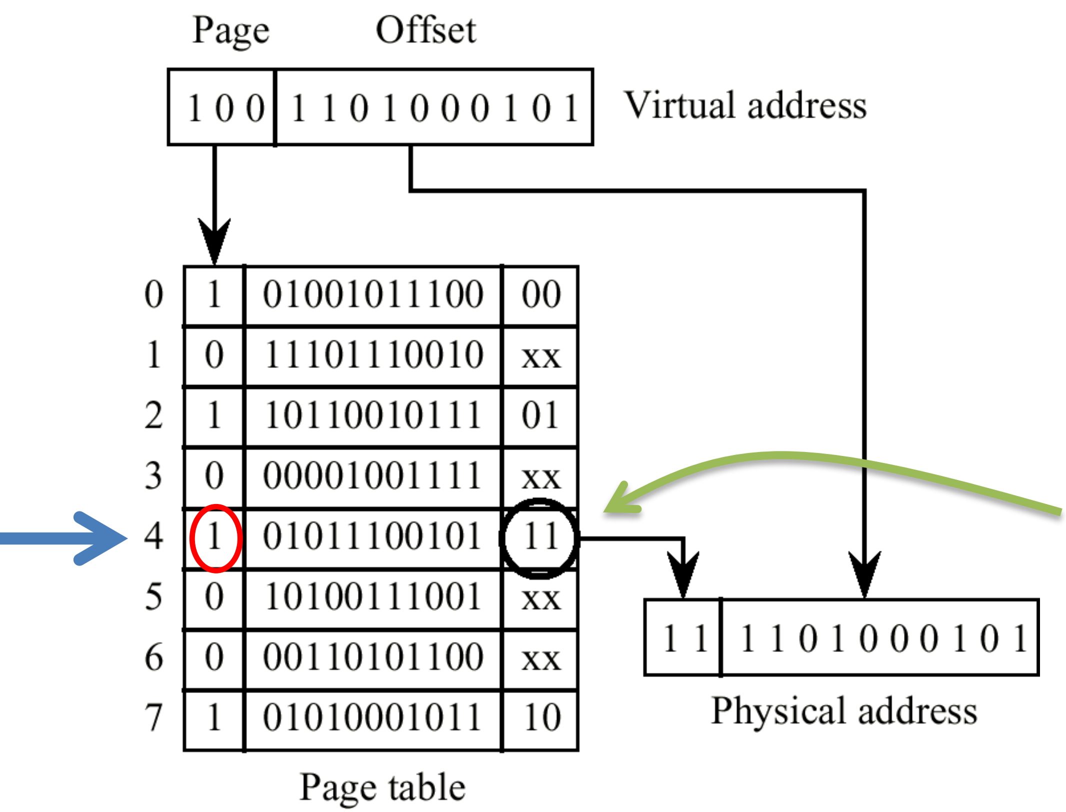
This is 16 bits, but the virtual address is 13 bits, so remove the top 3 bits:

1 0011 0100 0101

Recall that virtual address is split into two fields: virtual page number and page offset

We have 8 pages in VM, so virtual page number field will be first 3 bits of address

Page size is 1K, so offset field is remaining 10 bits of address



Virtual page number is 100 = 4, so look at row 4 in page table

Present bit is 1, so our page is located in physical memory (hit)

Page frame field is 11, which becomes the physical page number field for our physical address.

$11_2 = 3$, so our page is located in page frame 3 in physical memory

Offset field of $11\ 0100\ 0101_2$ is copied from virtual address to physical address

Recall offset field identifies specific byte in page (as cache byte field identifies specific byte in cache block)

$11\ 0100\ 0101_2 = 345H = 837_{10}$, so offset field addresses byte 837 in page.

Also note disk address: 010 1110 0101

So this page is a copy of page 2E5H from the hard drive

Page Faults

- If present bit = 0, then page is NOT in physical memory (primary level)
- This is a miss (page fault) – must find page in secondary level (disk memory) and copy to physical memory
- Recall that a fully associative scheme is used, so must find a page frame to copy to. Use LRU (Least Recently Used) similar to cache for page replacement policy.

Memory Design Hierarchy

(Chapter 2, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- 2.1 Introduction
- 2.2 Memory Technology and Optimizations
- 2.3 Ten Advanced Optimizations of Cache Performance
- 2.4 Virtual Memory and Machines

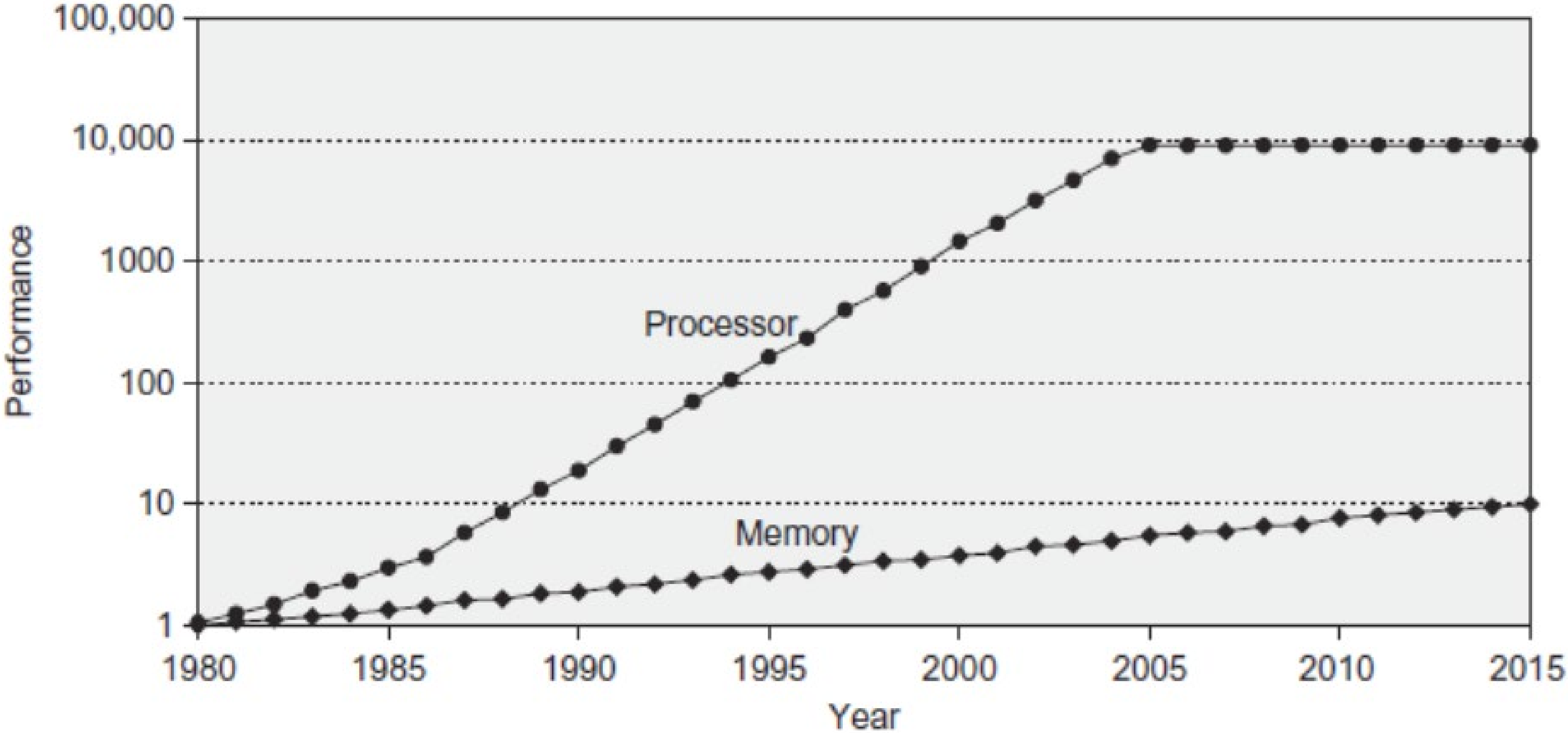
Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory

Introduction

- **Solution**: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!

Memory Hierarchy Design

- DRAM bandwidth is only 8% of this
  - 34.1 GB/s
  - Requires:
    - Multi-port, pipelined caches
    - Two levels of cache per core
    - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache =>* one block per set
  - *Fully associative* => one set
- Writing to cache: two strategies
  - *Write-through:* Immediately update lower levels of hierarchy
  - *Write-back:* Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Six Basic Cache Optimizations

1. Larger block size
   - Reduces compulsory misses
   - Increases capacity and conflict misses, increases miss penalty
2. Larger total cache capacity to reduce miss rate
   - Increases hit time, increases power consumption
3. Higher associativity
   - Reduces conflict misses
   - Increases hit time, increases power consumption

Six Basic Cache Optimizations

4. Higher number of cache levels
   - Reduces overall memory access time
5. Giving priority to read misses over writes
   - Reduces miss penalty
6. Avoiding address translation in cache indexing
   - Reduces hit time

Outline

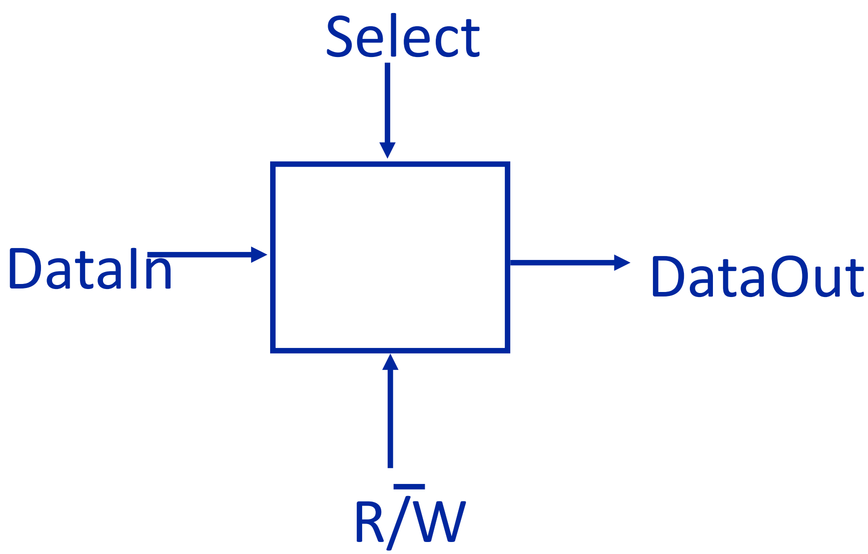
- 2.1 Introduction
- 2.2 Memory Technology and Optimizations
- 2.3 Ten Advanced Optimizations of Cache Performance
- 2.4 Virtual Memory and Machines

RAM (Random-Access Memory)

- Cache and main memory are RAM
- Types of memory access
  - Random-access: all cells can be accessed in equal time
  - Sequential: cells must be accessed in sequence (think of a tape)
  - Disk-access: time varies dependant on location of R/W head

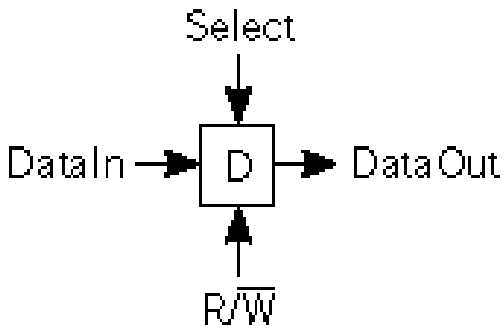
Memory Cells - a conceptual view

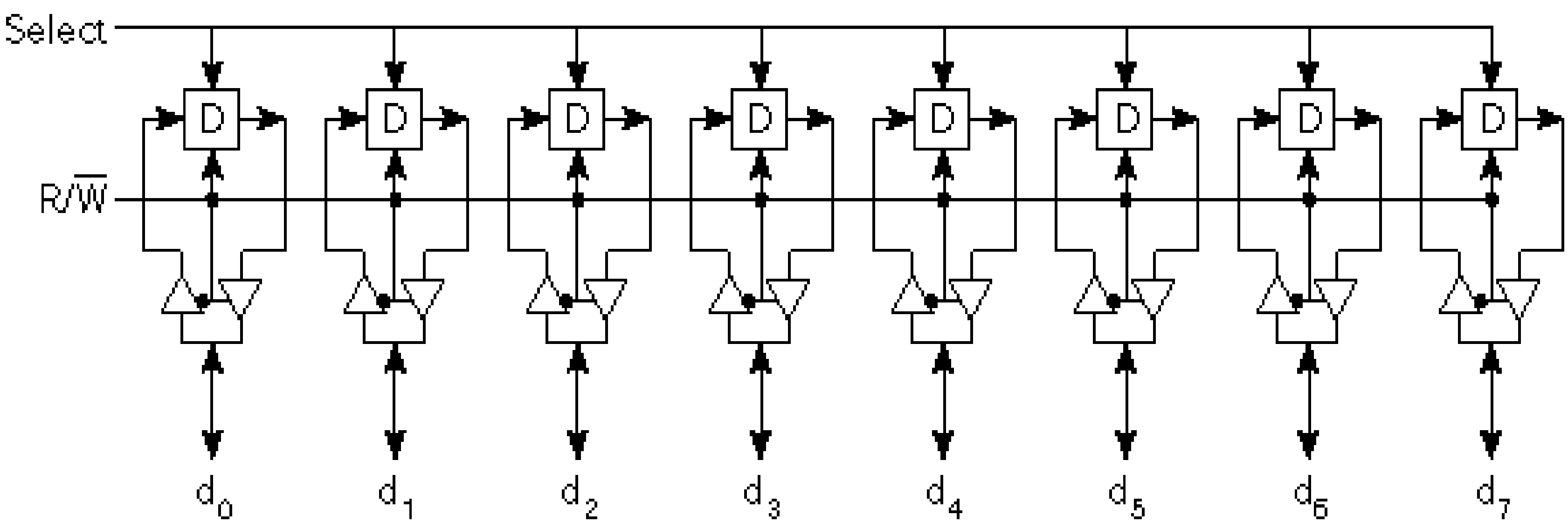
Regardless of the technology, all RAM memory cells must provide these four functions: Select, DataIn, DataOut, and R/W.



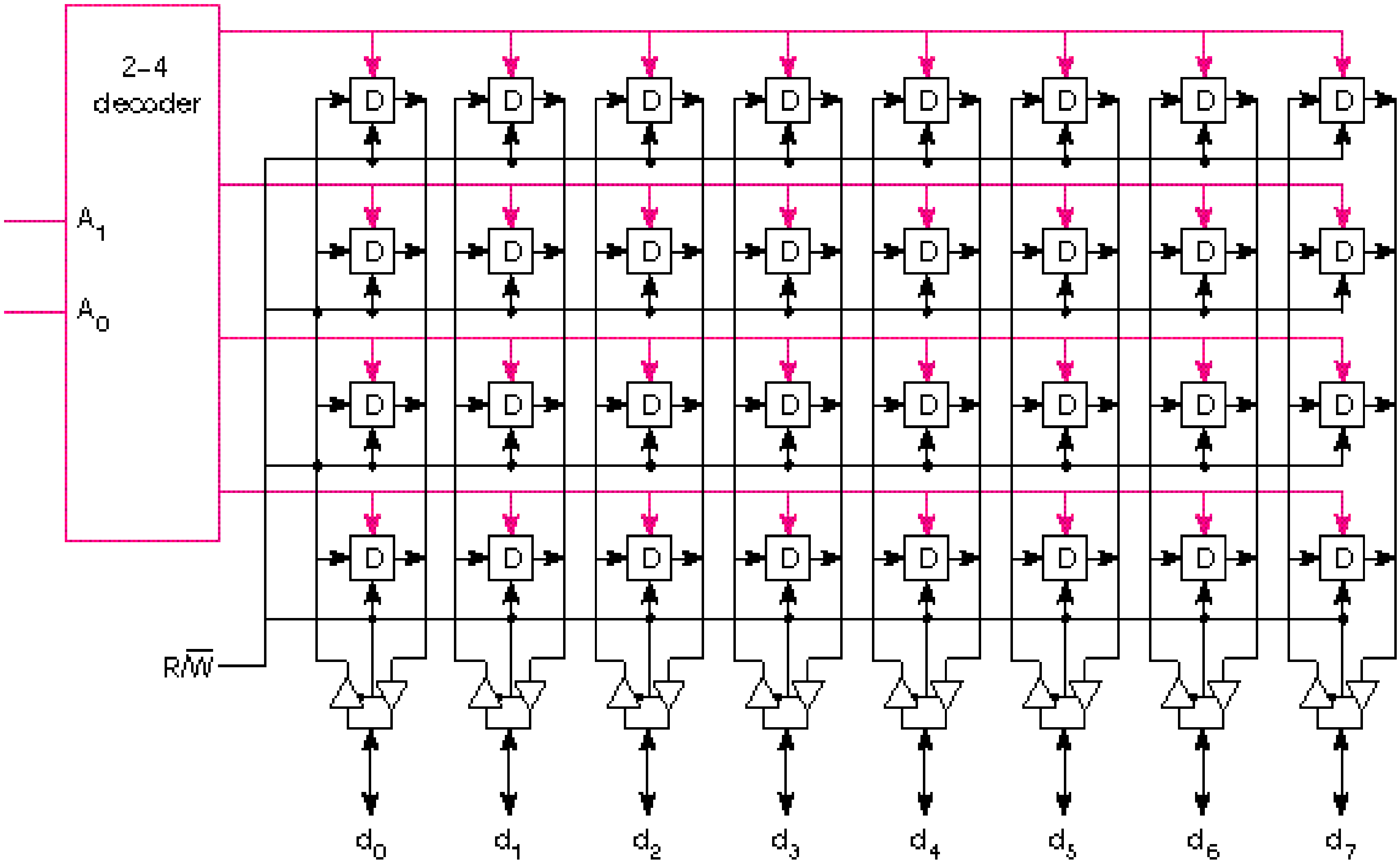
An 8-bit register as a 1D RAM array

The entire register is selected with one select line, and uses one





A 4x8 2D Memory Cell Array

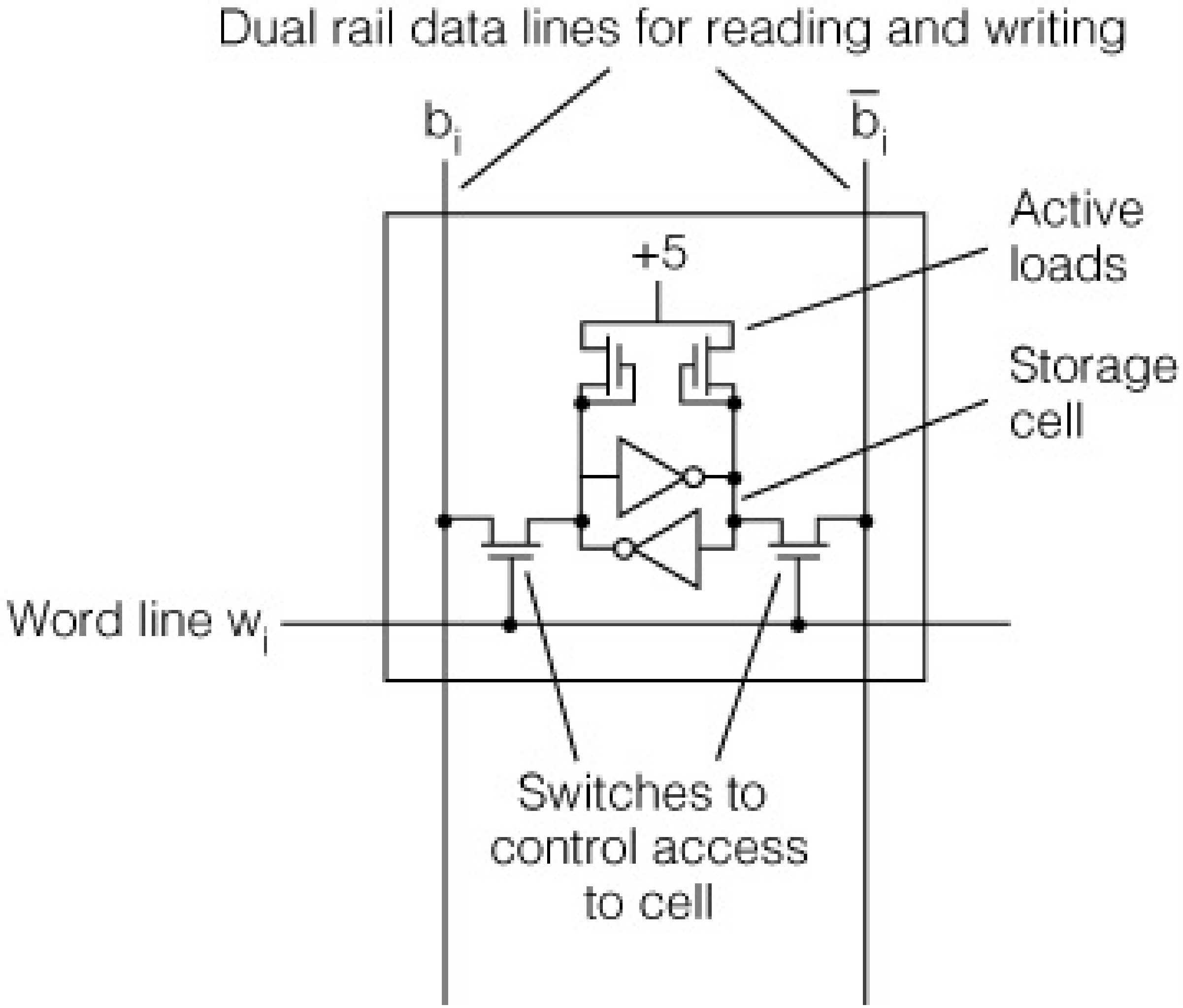


Memory Technology

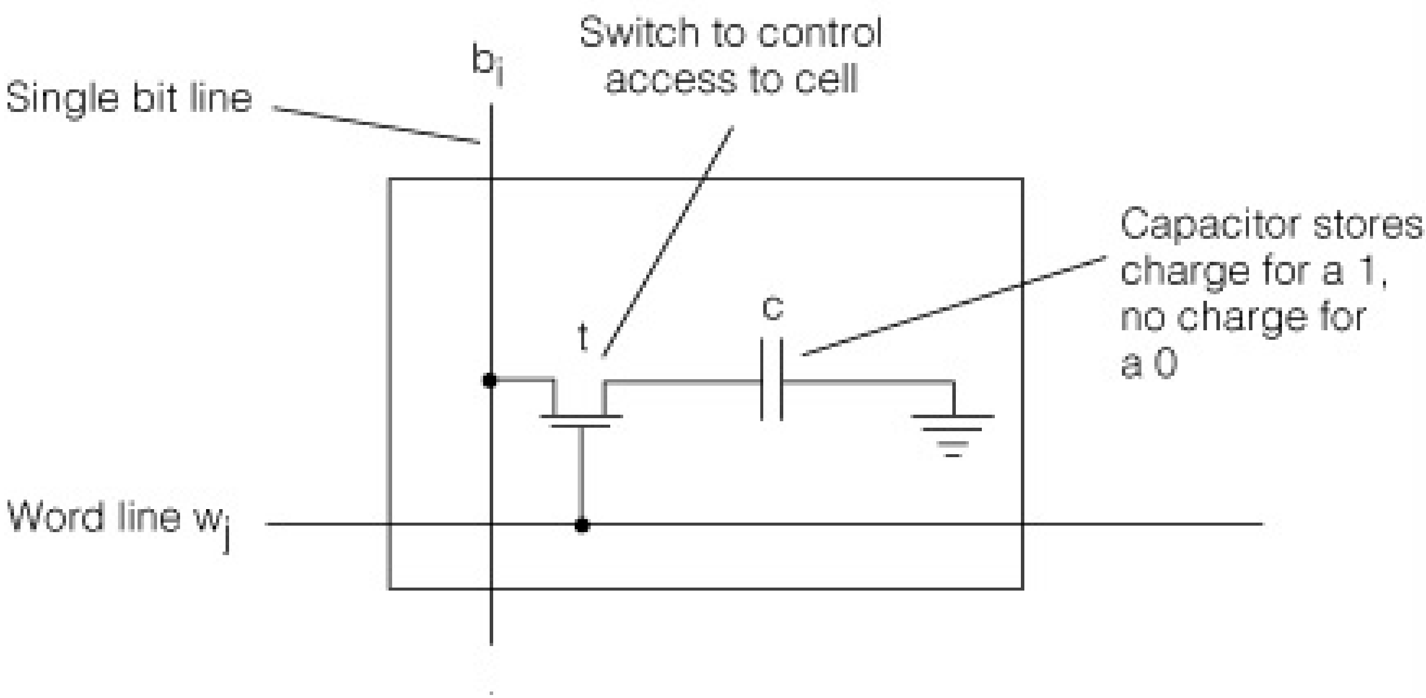
- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory

- DRAM used for main memory, SRAM used for cache

SRAM (Static RAM) cell



DRAM (Dynamic RAM) cell



Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Memory Cell Applications

- Main Memory is *DRAM*
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
- Cache uses *SRAM*
  - No refresh (6 transistors/bit vs. 1 transistor)
    Size: DRAM/SRAM *4-8*,
    Cost/Cycle time: SRAM/DRAM *8-16*