15-213
"The course that gives CMU its Zip!"

The Memory Hierarchy
Feb. 14, 2008

Topics
- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy

class10.ppt

Announcements

Recitation room changes
- C (Nate) Doherty 1211
- G (Pratyusa) Porter A22
- H (Ally) Porter A19

Exam date change
- NOT Thursday, 2/21
- CHANGED TO Tuesday, 2/26: 7:00 p.m. - 8:30 p.m.
- UC McConomy XOR Wean 7500 (expect e-mail)

Calculator policy
- Calculators will not be needed on the exam; hence forbidden.

Collaboration reminder
- Writing code together counts as "sharing code" - forbidden
- Talking through a problem can include pictures (not code)

Opinion Poll

Plan A
- 2/14 Thu Memory Hierarchy (DAE)
- 2/19 Tue Opt II (REB)
- 2/21 Thu No class? [Bill Gates]
- 2/26 Tue Cache Memories (DAE) [Evening Exam]
- 2/28 Thu Linking (DAE) ◊ Back to original schedule

Plan B
- 2/14 Thu Memory Hierarchy (DAE)
- 2/19 Tue Opt II (REB)
- 2/21 Thu Cache Memories (DAE) [Bill Gates]
- 2/26 Tue No class? [Evening Exam]
- 2/28 Thu Linking (DAE) ◊ Back to original schedule

Outline

RAM

ROM

Disks

"Mind the gap!"

Locality

Memory Hierarchy

Caches

Random-Access Memory (RAM)

Key features
- RAM is traditionally packaged as a chip.
- Basic storage unit is normally a cell (one bit per cell).
- Multiple RAM chips form a memory.

Static RAM (SRAM)
- Each cell stores a bit with a four or six-transistor circuit.
- Retains value indefinitely, as long as it is kept powered.
- *Relatively* insensitive to electrical noise (EMI), radiation, etc.
- Faster and more expensive than DRAM.

Dynamic RAM (DRAM)
- Each cell stores bit with a capacitor. One transistor is used for access
- Value must be refreshed every 10-100 ms.
- More sensitive to disturbances (EMI, radiation,...) than SRAM.
- Slower and cheaper than SRAM.

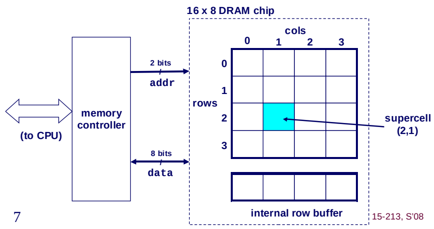
SRAM vs DRAM Summary

	Tran. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

Conventional DRAM Organization

d x w DRAM:

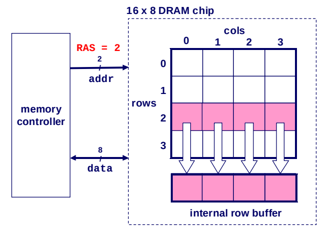
- dw total bits organized as d supercells of size w bits



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

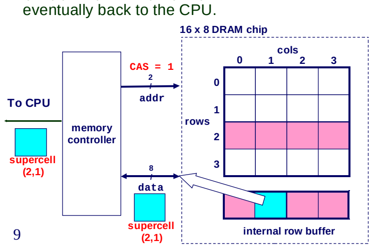
Step 1(b): Row 2 copied from DRAM array to row buffer.



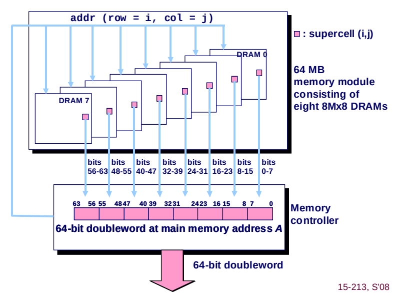
Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Memory Modules



Enhanced DRAMs

DRAM Cores with better interface logic and faster I/O :

- Synchronous DRAM (SDRAM)
  - Uses a conventional clock signal instead of asynchronous control
- Double data-rate synchronous DRAM (DDR SDRAM)
  - Double edge clocking sends two bits per cycle per pin
- RamBus™ DRAM (RDRAM)
  - Uses faster signaling over fewer wires (source directed clocking) with a Transaction oriented interface protocol

Obsolete Technologies :

- Fast page mode DRAM (FPM DRAM)
  - Allowed re-use of row-addresses
- Extended data out DRAM (EDO DRAM)
  - Enhanced FPM DRAM with more closely spaced CAS signals.
- Video RAM (VRAM)
  - Dual ported FPM DRAM with a second, concurrent, serial interface
- Extra functionality DRAMS (CDRAM, GDRAM)
  - Added SRAM (CDRAM) and support for graphics operations (GDRAM)

Nonvolatile Memories

DRAM and SRAM are volatile memories

- Lose information if powered off.

Nonvolatile memories retain value even if powered off

- Read-only memory (ROM): programmed during production
- Magnetic RAM (MRAM): stores bit magnetically (in development)
- Ferro-electric RAM (FERAM): uses a ferro-electric dielectric
- Programmable ROM (PROM): can be programmed once
- Eraseable PROM (EPROM): can be bulk erased (UV, X-Ray)
- Electrically eraseable PROM (EEPROM): electronic erase capability
- Flash memory: EEPROMs with partial (sector) erase capability

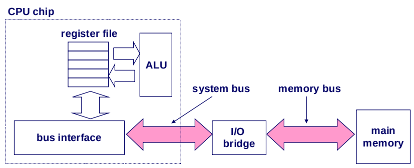
Uses for Nonvolatile Memories

- Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,...)
- Solid state disks (flash cards, memory sticks, etc.)
- Smart cards, embedded systems, appliances
- Disk caches

Traditional Bus Structure Connecting CPU and Memory

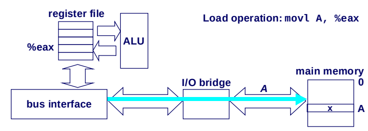
A **bus** is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.



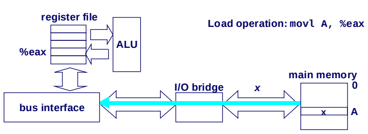
Memory Read Transaction (1)

CPU places address A on the memory bus.



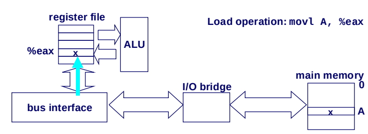
Memory Read Transaction (2)

Main memory reads A from the memory bus, retrieves word x, and places it on the bus.



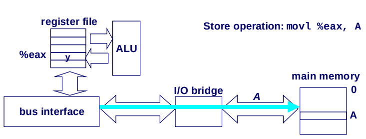
Memory Read Transaction (3)

CPU read word x from the bus and copies it into register %eax.



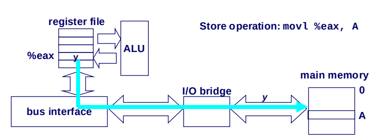
Memory Write Transaction (1)

CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



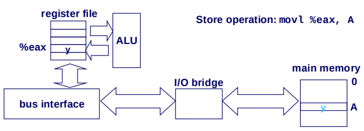
Memory Write Transaction (2)

CPU places data word y on the bus.



Memory Write Transaction (3)

Main memory reads data word y from the bus and stores it at address A.



Memory Subsystem Trends

Observation: A DRAM chip has an access time of about 50ns. Traditional systems may need 3x longer to get the data from memory into a CPU register.

Modern systems integrate the memory controller onto the CPU chip: Latency matters!

DRAM and SRAM densities increase and so does the soft-error rate:

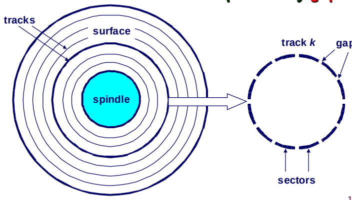
- Traditional error detection & correction (EDC) is a must have (64bit of data plus 8bits of redundancy allow any 1 bit error to be corrected and any 2 bit error is guaranteed to be detected)
- EDC is increasingly needed for SRAMs too
- ChipKill™ capability (can correct all bits supplied by one failing memory chip) will become standard soon

Disk Geometry

Disks consist of platters, each with two surfaces.

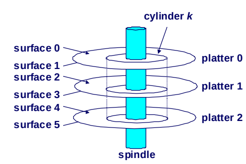
Each surface consists of concentric rings called tracks.

Each track consists of sectors separated by gaps.



Disk Geometry (Muliple-Platter View)

Aligned tracks form a cylinder.



Disk Capacity

Capacity: maximum number of bits that can be stored.

- Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9 Bytes (Lawsuit pending! Claims deceptive advertising).

Capacity is determined by these technology factors:

- Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
- Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
- Areal density (bits/in^2): product of recording and track density.

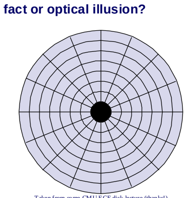
Sector Sizes

Sectors on the outer tracks appear larger

More area per bit might be more reliable (for precious data?)

As number of tracks gets large, this could get wasteful

Mathematical fact or optical illusion?

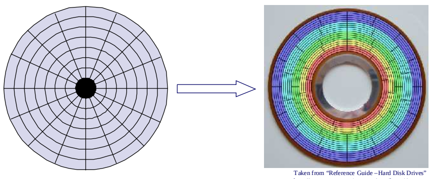


Taken from some CMU ECE disk lecture (thanks!)

Zoned Bit Recording (ZBR)

Modern disks partition tracks into disjoint subsets called recording zones

- Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
- Each zone has a different number of sectors/track



Taken from "Reference Guide – Hard Disk Drives" http://www.storagereview.com/map/lm.cgi/zone

Computing Disk Capacity

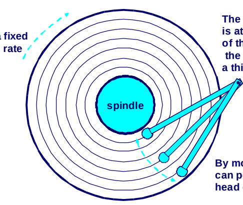
Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:

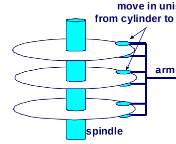
- 512 bytes/sector
- 300 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 300 x 20000 x 2 x 5
= 30,720,000,000
= 30.72 GB

Disk Operation (Single-Platter View)



Disk Operation (Multi-Platter View)



Disk Access Time

Average time to access some target sector approximated by :

- Taccess = Tavg seek + Tavg rotation + Tavg transfer

Seek time (Tavg seek)

- Time to position heads over cylinder containing target sector.
- Typical Tavg seek = 9 ms

Rotational latency (Tavg rotation)

- Time waiting for first bit of target sector to pass under r/w head.
- Tavg rotation = 1/2 x 1/RPMs x 60 sec/1 min

Transfer time (Tavg transfer)

- Time to read the bits in the target sector.
- Tavg transfer = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min.

Disk Access Time Example

Given:

- Rotational rate = 7,200 RPM
- Average seek time = 9 ms.
- Avg # sectors/track = 400.

Derived:

- Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
- Tavg transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
- Taccess = 9 ms + 4 ms + 0.02 ms

Important points:

- Access time dominated by seek time and rotational latency.
- First bit in a sector is the most expensive, the rest are free.
- SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
  - Disk is about 40,000 times slower than SRAM,
  - 2,500 times slower then DRAM.

Logical Disk Blocks

Modern disks present a simpler abstract view of the complex sector geometry:

- The set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)

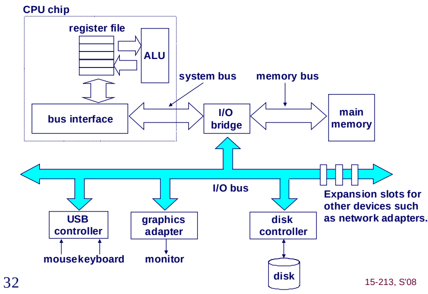
Mapping between logical blocks and actual (physical) sectors

- Maintained by hardware/firmware device called disk controller.
- Converts requests for logical blocks into (surface,track,sector) triples.
- This approach is logical block addressing ("LBA")

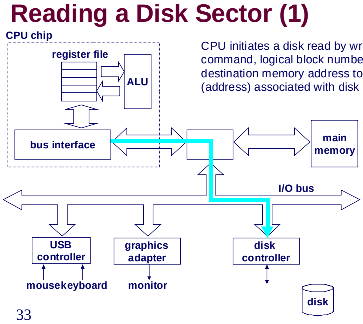
Allows controller to set aside spare cylinders for each zone.

- Accounts for the difference in "formatted capacity" and "maximum capacity".

I/O Bus

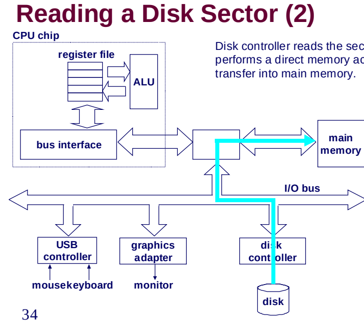


Reading a Disk Sector (1)



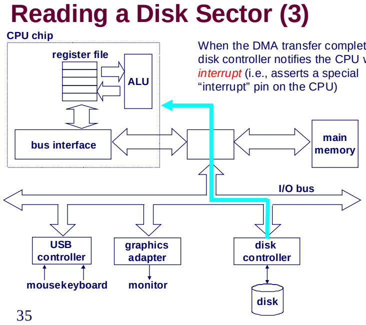
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Reading a Disk Sector (2)



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a Disk Sector (3)



When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

Storage Trends

SRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	19,200	2,900	320	256	100	75	*256*
access (ns)	300	150	35	15	12	10	*30*

DRAM

metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	8,000	880	100	30	1	0.20	*40,000*
access (ns)	375	200	100	70	60	50	*8*
typical size(MB)	0.064	0.256	4	16	64	1,000	*15,000*

Disk

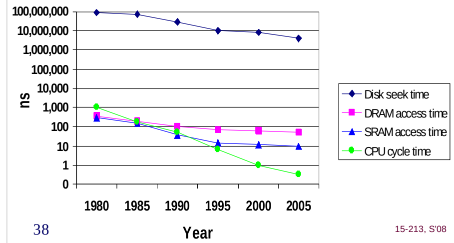
metric	1980	1985	1990	1995	2000	2005	*2005:1980*
$/MB	500	100	8	0.30	0.05	0.001	*10,000*
access (ms)	87	75	28	10	8	*4*	*22*
typical size(MB)	1	10	160	1,000	9,000	400,000	*400,000*

CPU Clock Rates

	1980	1985	1990	1995	2000	2005	*2005:1980*
processor	8080	286	386	Pentium	P-III	P-4	
clock rate(MHz)	1	6	20	150	750	3,000	3,000
cycle time(ns)	1,000	166	50	6	1.3	0.3	3,333

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Locality

Principle of Locality:
- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
- Temporal locality: Recently referenced items are likely to be referenced in the near future.
- Spatial locality: Items with nearby addresses tend to be referenced close together in time.

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Locality Example:
- Data
  - Reference array elements in succession (stride-1 reference pattern): Spatial locality
  - Reference `sum` each iteration: Temporal locality
- Instructions
  - Reference instructions in sequence: Spatial locality
  - Cycle through loop repeatedly: Temporal locality

Locality Example

Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

Question: Does this function have good locality?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```


Locality Example

Question: Does this function have good locality?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```


Locality Example

Question: Can you permute the loops so that the function scans the 3-d array `a[]` with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

Memory Hierarchies

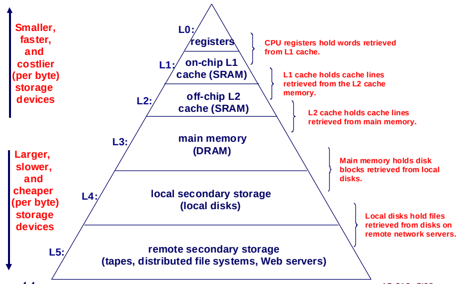
Some fundamental and enduring properties of hardware and software:

- Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
- The gap between CPU and main memory speed is widening.
- Well-written programs tend to exhibit good locality.

These fundamental properties complement each other beautifully.

They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

An Example Memory Hierarchy



Caches

Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

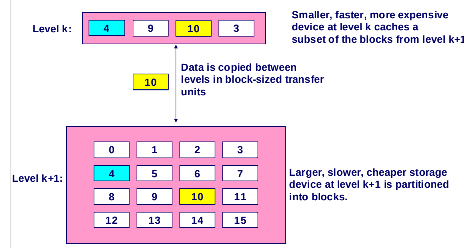
Fundamental idea of a memory hierarchy:

- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.

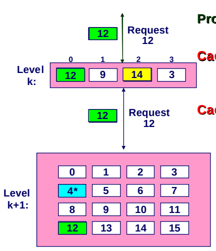
Why do memory hierarchies work?

- Programs tend to access the data at level k more often than they access the data at level k+1.
- Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

Caching in a Memory Hierarchy



General Caching Concepts



Program needs object d, which is stored in some block b.

Cache hit

- Program finds b in the cache at level k. E.g., block 14.

Cache miss

- b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12.
- If level k cache is full, then some current block must be replaced (evicted). Which one is the "victim"?
  - Placement policy: where can the new block go? E.g., b mod 4
  - Replacement policy: which block should be evicted? E.g., LRU

General Caching Concepts

Types of cache misses:

- Cold (compulsory) miss
  - Cold misses occur because the cache is empty.
- Conflict miss
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
  - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k+1.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
  - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- Capacity miss
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-byte words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware+ OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Summary

- **The memory hierarchy is fundamental consequence of maintaining the *random access memory* abstraction and practical limits on cost and power consumption.**
- **Caching works!**
- **Programming for good *temporal* and *spatial* locality is critical for high performance.**
- **Trend: the speed gap between CPU, memory and mass storage continues to widen, thus leading towards deeper hierarchies.**
  - **Consequence: maintaining locality becomes even more important.**