

The Memory Hierarchy

These slides based on some provided by the authors of our textbook

Topics

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

Random-Access Memory (RAM)

Key features

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

Static RAM (SRAM)

- Each cell stores bit with a six-transistor circuit.
- Retains value indefinitely, as long as it is kept powered.
- Relatively insensitive to disturbances such as electrical noise.
- Faster and more expensive than DRAM.

Dynamic RAM (DRAM)

- Each cell stores bit with a capacitor and transistor.
- Value must be refreshed every 10-100 ms.
- Sensitive to disturbances.
- Slower and cheaper than SRAM.

CompOrg - Memory Hierarchy

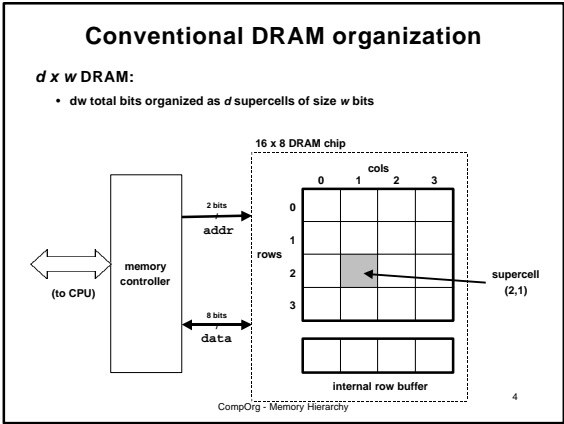
2

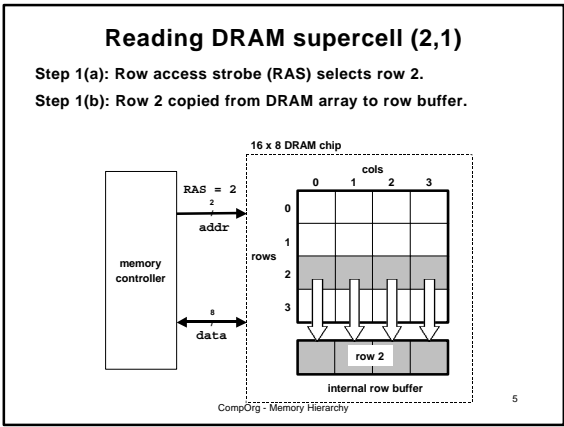
SRAM vs DRAM summary

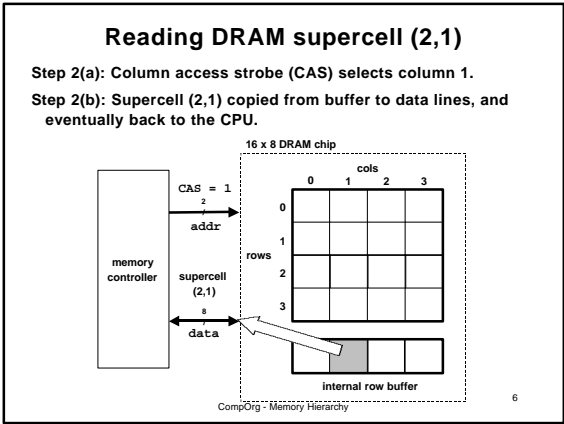
	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

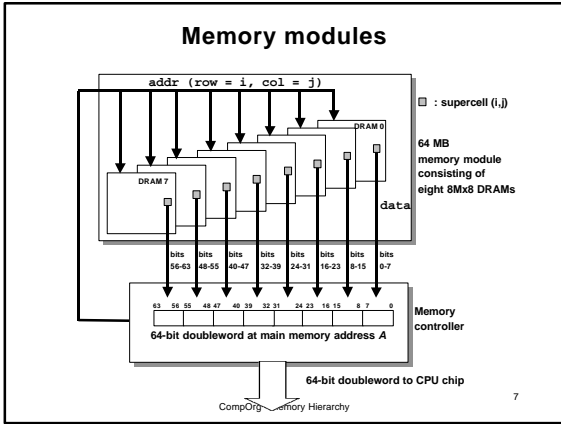
CompOrg - Memory Hierarchy

3









Enhanced DRAMs

All enhanced DRAMs are built around the conventional DRAM core.

- **Fast page mode DRAM (FPM DRAM)**
 - Access contents of row with [RAS, CAS, CAS, CAS, CAS] instead of [(RAS,CAS), (RAS,CAS), (RAS,CAS), (RAS,CAS)].
- **Extended data out DRAM (EDO DRAM)**
 - Enhanced FPM DRAM with more closely spaced CAS signals.
- **Synchronous DRAM (SDRAM)**
 - Driven with rising clock edge instead of asynchronous control signals.
- **Double data-rate synchronous DRAM (DDR SDRAM)**
 - Enhancement of SDRAM that uses both clock edges as control signals.
- **Video RAM (VRAM)**
 - Like FPM DRAM, but output is produced by shifting row buffer
 - Dual ported (allows concurrent reads and writes)

CompOrg - Memory Hierarchy 8

Nonvolatile memories

DRAM and SRAM are *volatile* memories

- Lose information if powered off.

Nonvolatile memories retain value even if powered off.

- Generic name is read-only memory (ROM).
- Misleading because some ROMs can be read and modified.

Types of ROMs

- Programmable ROM (PROM)
- Erasable programmable ROM (EPROM)
- Electrically erasable PROM (EEPROM)
- Flash memory

Firmware

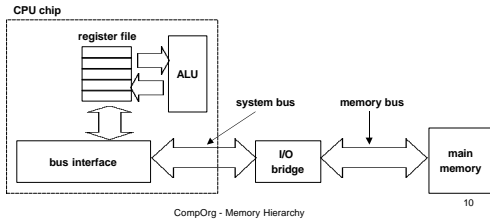
- Program stored in a ROM
 - Boot time code, BIOS (basic input/output system)
 - graphics cards, disk controllers.

CompOrg - Memory Hierarchy 9

Bus structure connecting CPU and memory

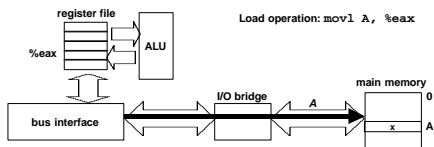
A *bus* is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.



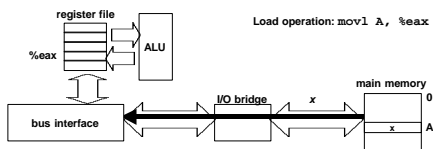
Memory read transaction (1)

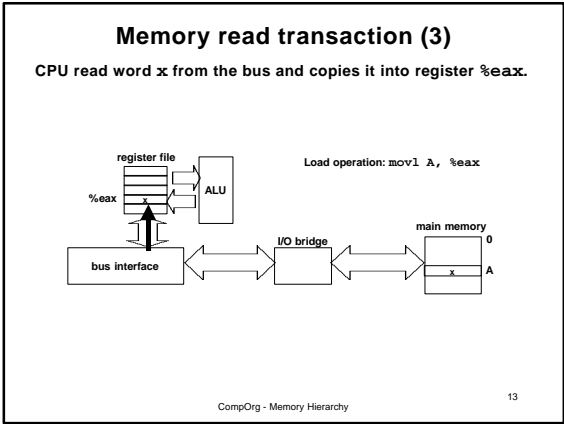
CPU places address *A* on the memory bus.

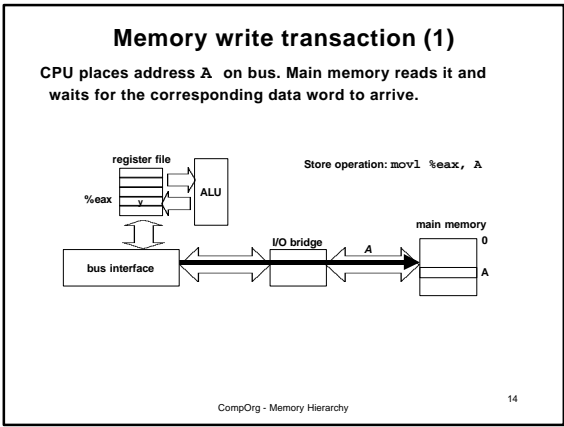


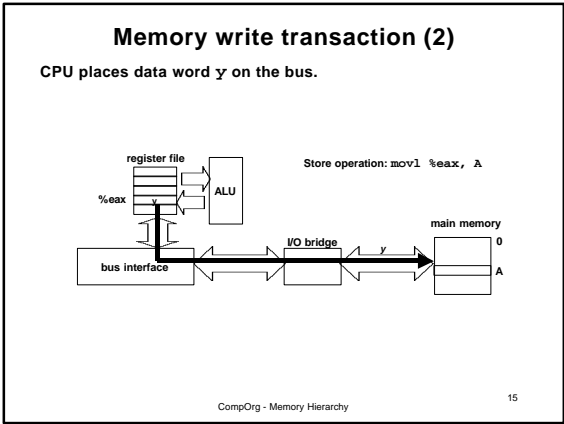
Memory read transaction (2)

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



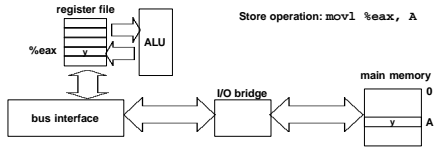






Memory write transaction (3)

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

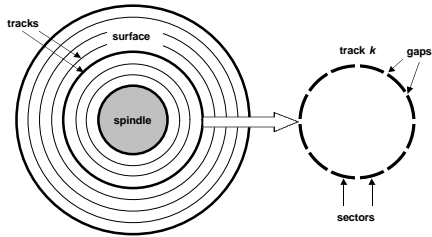


CompOrg - Memory Hierarchy

16

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.

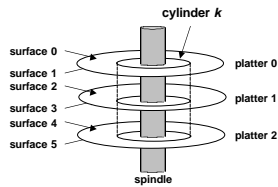


CompOrg - Memory Hierarchy

17

Disk geometry (multiple-platter view)

Aligned tracks form a *cylinder*.



CompOrg - Memory Hierarchy

18

Disk capacity

Capacity: maximum number of bits that can be stored.

- Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^6 .

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/in²):** product of recording and track density.

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

CompOrg - Memory Hierarchy

19

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

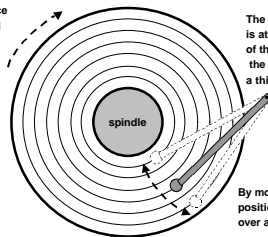
Capacity = 512 x 300 x 20000 x 2 x 5
= 30,720,000,000
= 30.72 GB

CompOrg - Memory Hierarchy

20

Disk operation (single-platter view)

The disk surface spins at a fixed rotational rate



The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

CompOrg - Memory Hierarchy

21

Disk operation (multi-platter view)

read/write heads
move in unison
from cylinder to cylinder

arm

spindle

CompOrg - Memory Hierarchy 22

Disk access time

Average time to access some target sector approximated by :

- $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$

Seek time

- Time to position heads over cylinder containing target sector.
- Typical $T_{\text{avg seek}} = 9 \text{ ms}$

Rotational latency

- Time waiting for first bit of target sector to pass under r/w head.
- $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$

Transfer time

- Time to read the bits in the target sector.
- $T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg \# sectors/track}) \times 60 \text{ secs}/1 \text{ min}$.

CompOrg - Memory Hierarchy 23

Disk access time example

Given:

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

Derived:

- $T_{\text{avg rotation}} = 1/2 \times (60 \text{ secs}/7200 \text{ RPM}) \times 1000 \text{ ms}/\text{sec} = 4 \text{ ms}$.
- $T_{\text{avg transfer}} = 60/7200 \text{ RPM} \times 1/400 \text{ secs}/\text{track} \times 1000 \text{ ms}/\text{sec} = 0.02 \text{ ms}$
- $T_{\text{access}} = 9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms}$

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 4ns/doubleword, DRAM about 60 ns
 - Disk is about 40,000 times slower than SRAM,
 - 2,500 times slower than DRAM.

CompOrg - Memory Hierarchy 24

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.

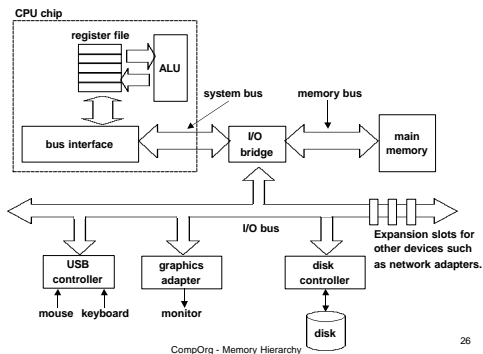
Allows controller to set aside spare cylinders for each zone.

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

CompOrg - Memory Hierarchy

25

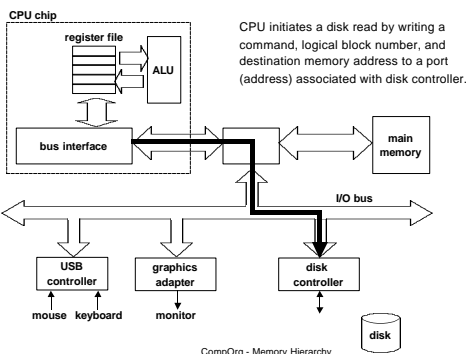
Bus structure connecting I/O and CPU



CompOrg - Memory Hierarchy

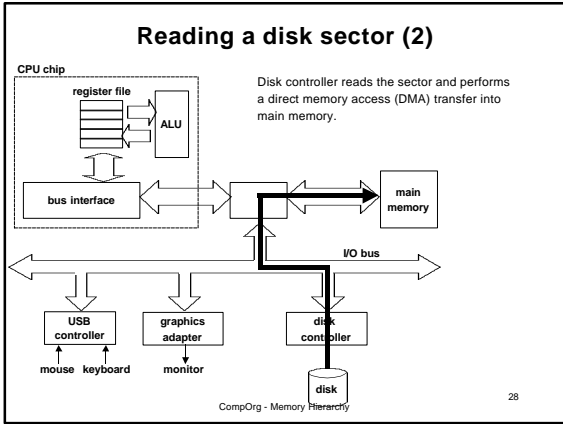
26

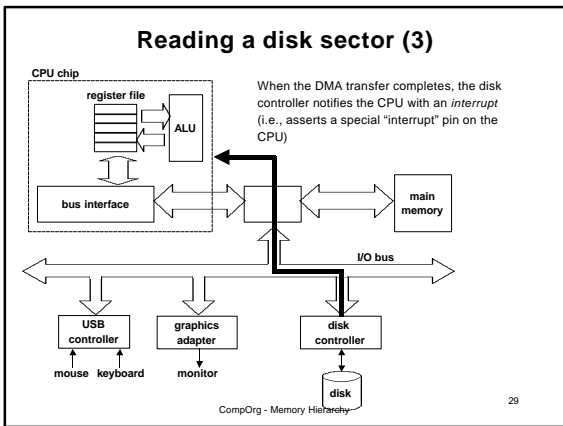
Reading a disk sector (1)



CompOrg - Memory Hierarchy

27





Storage trends

	metric	1980	1985	1990	1995	2000	2000:1980
SRAM	\$/MB	19,200	2,900	320	256	100	190
	access (ns)	300	150	35	15	2	100
DRAM	\$/MB	8,000	880	100	30	1	8,000
	access (ns)	375	200	100	70	60	6
	typical size(MB)	0.064	0.256	4	16	64	1,000
Disk	\$/MB	500	100	8	0.30	0.05	10,000
	access (ms)	87	75	28	10	8	11
	typical size(MB)	1	10	160	1,000	9,000	9,000

(Culled from back issues of Byte and PC Magazine) ³⁰

30

CPU clock rates

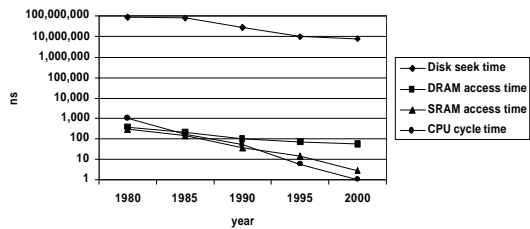
	1980	1985	1990	1995	2000	2000:1980
processor	8080	286	386	Pent	P-III	
clock rate(MHz)	1	6	20	150	750	750
cycle time(ns)	1,000	166	50	6	1.6	750

CompOrg - Memory Hierarchy

31

The CPU-Memory Gap

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



CompOrg - Memory Hierarchy

32

Locality of Reference

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 (spatial)
- **Instructions**
 - Reference instructions in sequence (spatial)
 - Cycle through loop repeatedly (temporal)

CompOrg - Memory Hierarchy

33

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.

Does this function have good locality?

```
int sumarrayrows(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
}
```

CompOrg - Memory Hierarchy

34

Locality example

Does this function have good locality?

```
int sumarraycols(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
}
```

CompOrg - Memory Hierarchy

35

Locality example

Permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good locality)

```
int sumarray3d(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
}
```

CompOrg - Memory Hierarchy

36

Memory hierarchies

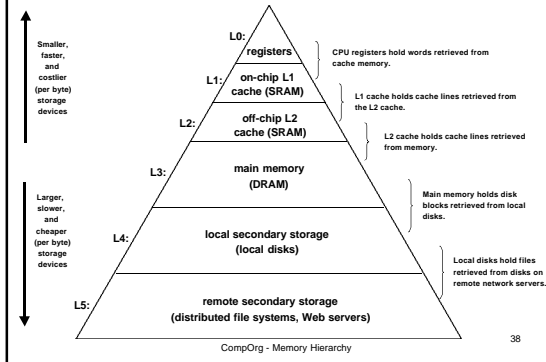
Some fundamental and enduring properties of hardware and software:

- Fast storage technologies cost more per byte and have less capacity.
- 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.

Suggest an approach for organizing memory and storage systems known as a “memory hierarchy”.

An example memory hierarchy



Caches

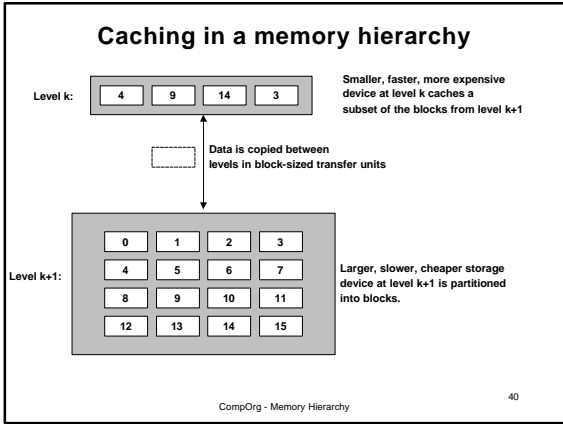
Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

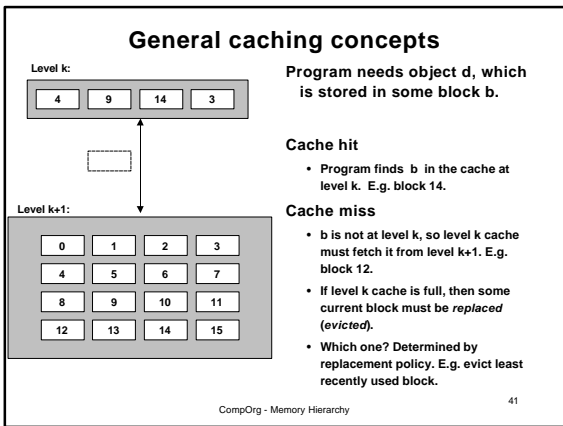
Fundamental idea of a memory hierarchy:

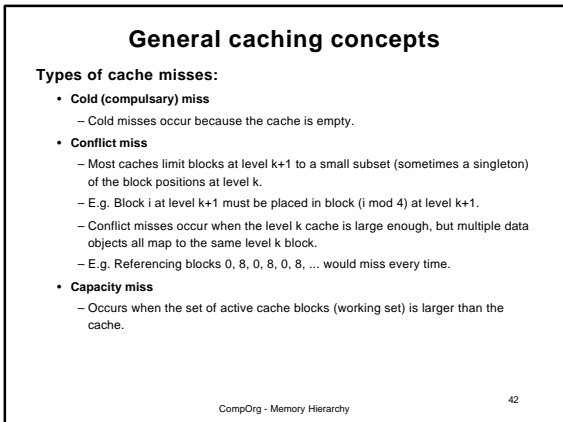
- For each k , the faster, smaller device at level k serves as a cache for the larger, slower device at level $k+1$.

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.







Examples of caching in the hierarchy

Cache Type	What Cached	Where Cached	Latency (cycles)	Managed By
Registers	4-byte word	CPU registers	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	32-byte block	On-Chip L1	1	Hardware
L2 cache	32-byte 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

CompOrg - Memory Hierarchy

43
