A computer system contains a hierarchy of storage devices with different costs, capacities, and access times.

With a memory hierarchy, a faster storage device at one level of the hierarchy acts as a staging area for a slower storage device at the next lower level.

Software that is well-written takes advantage of the hierarchy accessing the faster storage device at a particular level more frequently than the storage at the next level.

As a programmer, understanding the memory hierarchy will result in better performance of applications.
Random Access Memory (RAM)

Features
- Basic storage unit is a cell (one bit per cell); RAM is traditionally packaged as a chip; multiple chips form memory

Static RAM (SRAM)
- Each cell implemented with a six-transistor circuit
- Relatively insensitive to disturbances such as electrical noise, radiation, etc.
- Faster and more expensive than DRAM

Dynamic RAM (DRAM)
- Each bit stored as charge on a capacitor
- Value must be refreshed every 10-100 ms
- Sensitive to disturbances
- Slower and cheaper than SRAM

<table>
<thead>
<tr>
<th></th>
<th>Transistors per bit</th>
<th>Access time</th>
<th>Needs refresh?</th>
<th>Sensitive?</th>
<th>Cost</th>
<th>Power requirements</th>
<th>Power dissipation</th>
<th>Chip density</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>4 or 6</td>
<td>1X</td>
<td>No</td>
<td>No</td>
<td>100x</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>DRAM</td>
<td>1</td>
<td>10X</td>
<td>Yes</td>
<td>Yes</td>
<td>1X</td>
<td>low</td>
<td>low</td>
<td>low</td>
</tr>
</tbody>
</table>

Conventional DRAM Organization
- (d x w) bit DRAM chip is organized as d supercells of size w bits
- Usually w=8, i.e. one byte

Reading 16x8 DRAM Supercell [2,1]

Step 1: Row access strobe (RAS) selects row 2. Row 2 copied from DRAM array to row buffer.
Reading 16x8 DRAM Supercell [2,1] (cont.)
Step 2: Column access strobe (CAS) selects column 1. Supercell [2,1] copied from buffer to data lines to CPU.

- Step 3: Since a read is distractive, the internal row buffer has to be written back into the corresponding row of DRAM array after each read.

Conventional 4Gbit DRAM Organization
- 4Gbit DRAM can be organized as 512M supercells of size 8 bits.
- $512M = 2^9 \times 2^{20} = 2^{14} \times 2^{15} = 16,384 \times 32,768$

Writing DRAM
- CPU provides address and data to be written
- Step 1 write is identical to step 1 read.
- Step 2 write includes updating the appropriate cell in the internal row buffer with data received from CPU
- Step 3 write is identical to step 3 read.

4GB Memory out of Eight 512Mx8 DRAMs
- 64-bit doubleword at main memory address $A$
Enhanced DRAMs

• Enhanced DRAMs have optimizations that improve the speed with which the basic DRAM cells can be accessed.

• Examples:
  – Fast page mode DRAM (FPM DRAM); up to 1996
  – Extended data out DRAM (EDO DRAM); 1996-99
  – Synchronous DRAM (SDRAM)
  – Double Data-Rate Synchronous DRAM (DDR SDRAM)
  – Rambus DRAM (RDRAM)

Prices of Six Generations of DRAMs

• DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter considerably slower.

  • The improvements in access time have been slower but continuous, and cost roughly tracks density improvements, although cost is often affected by other issues, such as availability and demand.

  • The cost per gigabyte (10^9 bytes) is not adjusted for inflation.

Nonvolatile Memory

- Information retained if supply voltage is turned off
- Referred to as read-only memories (ROM), although some may be written to as well as read
- Read-only memory (ROM): programmed during production
- Programmable ROM (PROM): fuse associated with cell that is blown once by zapping with current; can be programmed once
- Erasable PROM (EPROM): cells cleared by shining ultraviolet light, special device used to write 1’s; can be erased and reprogrammed about 1000 times
- Electrically erasable PROM (EEPROM): similar to EPROM but does not require a physically separate programming device, can be re-programmed in place on printed circuit cards; can be reprogrammed about 100,000 times
- Flash Memory
  - Based on EEPROM technology

Solid State Disk - SSD

- SSD package plugs into a standard disk slot on the I/O bus (typically USB or SATA) and behaves like a disk reading and writing logical blocks.
- Consists of one or more flash memory chips and a flash translation layer (hardware/firmware device) that plays the same role as a disk controller.

Advantages of SSD over rotating disks:
- No moving parts – semiconductor memory is more rugged
- Much faster random access times
- Use less power

Disadvantages of SSD over rotating disks:
- SSDs wear out with usage
- 10-20 times more expensive than disks

Basic (Single CPU) Computer Structure

- CPU and device controllers connect through common bus providing access to shared memory

Solid State Disk – SSD (cont.)

- Pages: 512B to 4KB, Blocks: 32 to 128 pages
- Data read in units of pages.
- Page can be written only after its block has been erased
- A block wears out after 100,000 repeated writes.
Moving-Head Disk Mechanism

- A sector (usually 512 bytes) is a basic unit of transfer (read/write)

SSD Characteristics

<table>
<thead>
<tr>
<th></th>
<th>Sequential read throughput</th>
<th>Sequential write throughput</th>
<th>Random read throughput</th>
<th>Random write throughput</th>
<th>Random read access</th>
<th>Random write access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250 MB/s</td>
<td>170 MB/s</td>
<td>140 MB/s</td>
<td>14 MB/s</td>
<td>30 us</td>
<td>300 us</td>
</tr>
</tbody>
</table>

Growth in Microprocessor Performance

- Mismatch between CPU performance growth and memory performance growth → "memory wall"
- Importance of cache

Growth in Performance of RAM & CPU
**CPU Trends: “Power Wall”**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8080</td>
<td>386</td>
<td>Pentium</td>
<td>P-4</td>
<td>Core 2</td>
<td>Core i7</td>
<td>—</td>
</tr>
<tr>
<td>Clock rate(MHz)</td>
<td>1</td>
<td>20</td>
<td>600</td>
<td>3300</td>
<td>2000</td>
<td>2500</td>
<td>2500</td>
</tr>
<tr>
<td>Cycle time(ns)</td>
<td>1000</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.4</td>
<td>2500</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Effective cycle time(ns)</td>
<td>1000</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.25</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- At about the same time, besides “memory wall” and “power wall”, processor designers also reached limits in taking advantage of instruction level parallelism – ILP in (sequential) programs.
- Since early 2000’s processors have not been (significantly) getting faster.
- Instead, multi-core processors.

**Principle of Locality of Reference**

- Programs access a small proportion of their address space at any time and they tend to reuse instructions and data they have used recently.
  - temporal locality – recently accessed items are likely to be accessed soon,
  - spatial locality – items near those accessed recently are likely to be accessed soon, i.e. items near one another tend to be referenced close together in time.
- An implication of principle of locality is that we can predict with reasonable accuracy what instructions and data a program will use in near future based on its accesses in the recent past.
- Principle of locality applies more strongly to code accesses than data accesses.

**Taking Advantage of Locality**

- Use memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory (virtual memory)
- Copy recently accessed (and nearby) items from DRAM memory to smaller SRAM memory
  - Cache attached to CPU

**Typical System Organization Without Cache**

- Registers
- ALU
- System bus
- Memory bus
- I/O bus
- Expansion slots for other devices such as network adapters.
Typical Processor Organization with Cache

Cache is fast but because of that it has to be small. Why?

Model of Memory + Cache + CPU System

Basics of Cache Operation

- We will first (and mostly) consider a cache read operation, since it is more important. Why?
- When CPU provides an address requesting a content of a given main memory location:
  - first check the cache for the content; caches include a tag in each cache entry to identify the memory address of a block,
  - if the block present, this is a hit, get the content (fast) and CPU proceeds normally, without (or small) delay,
  - if the block not present, this is a miss, stall CPU and read the block with the required content from main memory,
  - long CPU slowdown: miss penalty $\rightarrow$ time to access main memory and to place a block into the cache,
  - And now (after miss penalty) CPU gets the required content.

Tags and Valid Bits

- How do we know what block is stored in a given cache location?
  - store block address as well as the data in a cache entry
  - actually, it may only need the high-order bits of block address called the tag
- What if there is no data in a location?
  - introduce valid bit in each entry
  - valid bit: 1 = present, 0 = not present
  - initially 0
**Cache Example**

- CPU generates 4-byte word read at address 100, i.e., read for contents of bytes at addresses 100-103
- Since cache is empty → cache miss

**Cache after 4B Read at Address 100**

1. Block of 16 byte read from RAM and stored in cache
2. Cache entry chosen according to cache placement algorithm (arbitrarily here).

**Cache after 4B Read at Address 204**

1. Block of 16 byte read from RAM and stored in cache
2. Cache entry chosen according to cache placement algorithm (arbitrarily here).

**Cache & DRAM Memory: Performance**

- \( t_2 \): main memory access time (40-60 nsec)
- \( t_1 \): cache access time (2-5 nsec)
- Hit ratio is a ratio of a number of hits and a total number of memory accesses; Miss ratio = 1 – Hit ratio

\[ \text{Average Access Time} = t_1 + t_2 + (1 - \text{hit ratio}) \times \text{miss penalty} \]

- Because of locality of reference, hit rates are normally well over 90%
Direct Mapped Cache

- In direct mapped caches, a block from memory has only one cache entry where it has to be stored.
- Location of cache entry is determined by block address and number of entries in the cache:
  - location = (block address) mod (number of blocks in cache)
- Block address includes all address bits excluding block offset bits, i.e. rightmost \( n \) bits where:
  - \( 2^n = \) a number of bytes in a block
- Example: 16 bit addresses and block size = 8
  - then address: \( 0x1234 = 0001 \ 0010 \ 0011 \ 0100_2 \) has block address \( 0x0246 = 0 \ 0010 \ 0100 \ 0110 \) since block offset = 3 (\( \boxed{2^3 = 8} \))

Direct Mapped Cache Example

- Assume processor with 10-bit addresses & a direct mapped cache with 8 entries, and 4 bytes blocks;
- 4 byte blocks = \( 2^2 = \) block offset \( \rightarrow \) block offset = 2
- block address = 10-2 = 8 bits
- 8 entries = \( 2^3 = 8 \) \( \rightarrow \) index = 3 bits
- Address format = 5 bit (tag) +3 bits (index) + 2 bits (block offset)

<table>
<thead>
<tr>
<th>V</th>
<th>Tag = 5 bits</th>
<th>Data = 32 bits = 4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Cache Example: Access 1

<table>
<thead>
<tr>
<th>Word address</th>
<th>Binary address</th>
<th>Hit/miss</th>
<th>Cache entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>00010 110 00</td>
<td>Miss</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>00010</td>
<td>Mem[88-91]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Example: Access 2

<table>
<thead>
<tr>
<th>Word address</th>
<th>Binary address</th>
<th>Hit/miss</th>
<th>Cache entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>01011 010 00</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>0011</td>
<td>Mem[104-107]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>00010</td>
<td>Mem[88-91]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Example: Accesses 3, 4

<table>
<thead>
<tr>
<th>Word address</th>
<th>Binary address</th>
<th>Hit/miss</th>
<th>Cache entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>00010 110 00</td>
<td>Hit</td>
<td>110</td>
</tr>
<tr>
<td>104</td>
<td>00011 010 00</td>
<td>Hit</td>
<td>010</td>
</tr>
</tbody>
</table>

Index | V | Tag | Data
000   | N |
001   | N |
010   | Y | 00011 | Mem[104-107] |
011   | N |
100   | N |
101   | N |
110   | Y | 00010 | Mem[88-91] |
111   | N |

Cache Example: Accesses 5, 6, 7

<table>
<thead>
<tr>
<th>Word address</th>
<th>Binary address</th>
<th>Hit/miss</th>
<th>Cache entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>00010 000 00</td>
<td>Miss</td>
<td>000</td>
</tr>
<tr>
<td>12</td>
<td>00000 011 00</td>
<td>Miss</td>
<td>011</td>
</tr>
<tr>
<td>64</td>
<td>00010 000 00</td>
<td>Hit</td>
<td>000</td>
</tr>
</tbody>
</table>

Index | V | Tag | Data
000   | Y | 00010 | Mem[64-67] |
001   | N |
010   | Y | 00011 | Mem[104-107] |
011   | Y | 00000 | Mem[12-15] |
100   | N |
101   | N |
110   | Y | 00010 | Mem[88-91] |
111   | N |

Cache Example: Accesses 8

<table>
<thead>
<tr>
<th>Word address</th>
<th>Binary address</th>
<th>Hit/miss</th>
<th>Cache entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>00010 010 00</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

Index | V | Tag | Data
000   | Y | 00010 | Mem[64-67] |
001   | N |
010   | Y | 00010 | Mem[72-75] |
011   | Y | 00000 | Mem[12-15] |
100   | N |
101   | N |
110   | Y | 00010 | Mem[88-91] |
111   | N |

Direct Mapped Cache

- Direct mapped caches: only one choice for cache entry
- Location of cache entry determined by block address
  - location = (block address) mod (number of blocks in cache)
- Use low-order address bits as a location for cache entry
Direct Mapping Cache: 1 \times 4\text{-byte Blocks}

- Block offset = 2 bits since $2^2 = 4$ bytes
- Here, block = data then block offset = byte offset and word offset =0
- Index = 14 bits since $2^{14} = 16K$ number of cache entries

Cache after Addresses 100 and 204

- Address 100_{10} = 0000000000000000 000000000110100 0 000000001100112
  - 4-byte block [100-103] will be stored in cache entry 25 = 000000001100112
  - Note: bytes with addresses 100-103 have same block address

- Address 204_{10} = 0000000000000000 000000001100111 00
  - block address = 0000000000000000 000000001100112
  - 4-byte block [204-207] will be stored in cache entry 51 = 000000001100112
  - Note: bytes with addresses 204-207 have same block address

Direct Mapping Cache: 4 \times 4\text{-byte Blocks}

- Block offset = 4 bits ($2^4 = 16$ bytes); Index = 12 bits ($2^{12} = 4K$ entries)
- Since 4-byte words (data) \rightarrow block offset = 2 bits word offset + 2 bits byte offset since $4 = 2^2 = 2$byte offset

Cache after Addresses 100 and 204

- Address 100_{10} = 0000000000000000 0000000001110100 0100
  - 16-byte block [96-111] will be stored in cache entry 6 = 0000000001110
  - Note: bytes with addresses 96-111 have same block address

- Address 204_{10} = 0000000000000000 000000001110
  - block address = 0000000000000000 000000001110
  - 16-byte block [192-207] will be stored in cache entry 12 = 000000001110
  - Note: bytes with addresses 192-207 have same block address
Block Size Consideration

- Cache with 4-byte blocks had 16K entries → total of 64KB data
- Cache with 16-byte blocks had 4K entries → total of 64KB data
- Thus, both caches can accommodate the same amount of data

<table>
<thead>
<tr>
<th>Program</th>
<th>Block size in bytes</th>
<th>Instruction miss rate</th>
<th>Data miss rate</th>
<th>Effective combined miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>4</td>
<td>6.1%</td>
<td>2.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.9%</td>
</tr>
<tr>
<td>spice</td>
<td>4</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0.3%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

- Larger blocks reduced miss rate due to spatial locality
- But for a fixed-sized cache, larger blocks → fewer of them → more competition → may increased miss rate
- Larger miss penalty may override benefit of reduced miss rate
- Thus keep in mind, the miss rate is not the only parameter:
  Average access time = hit time + miss rate × Miss penalty

Main Memory Supporting Caches

- DRAM memory has a width of its read/write operations determined by a width of its bus data lines.
- Numbers used in examples that follow for reading from main memory:
  - 2nsec for address transfer from CPU to memory controller (mostly propagation delay),
  - 50nsec per DRAM access,
  - 2nsec per data transfer from memory controller to cache (mostly propagation delay)

4-Byte Main Memory Bus

- For 16-byte block, and 4-byte-wide DRAM bus:
  Miss penalty = 2 + 4 × 50 + 4 × 2 = 210 nsec,
  Bandwidth = 16 bytes/210 = 0.08 bytes/nsec.
- Although caches are interested in low-latency memory, it is generally easier to improve memory bandwidth with new memory organization than it is to reduce memory latency.

16-Byte (Wider) Main Memory Bus

- For 16-byte block, and 16-byte-wide DRAM bus:
  Miss penalty = 2 + 50 + 2 = 54 nsec,
  Bandwidth = 16 bytes/54 = 0.30 bytes/nsec.
- CPU accesses a word at a time, so a need for a multiplexer
**Wider Main Memory Bus & Level-2 Cache**

- But the multiplexer is on the critical timing path.
- Level-2 cache helps since the multiplexing is now between level-1 and level-2 caches and not on critical timing path.

**Interleaved Memory Organization**

- This is four-way interleaved memory with 4 byte memory bus — saves wires in bus
- Miss penalty = 2 + 50 + 4 x 2 = 60nsec,
- Throughput = 16 bytes / 60 = 0.27 bytes/nsec

**Multilevel Caches**

- Level-1 (primary) cache attached to CPU
  - small, but very fast
- Level-2 (secondary) cache services misses from level-1 cache
  - larger, slower, but still faster than main memory
- Main memory services level-2 cache misses
- Some high-end systems include level-3 cache
- Level-1 cache: focus on minimal hit time
- Level-2 cache:
  - focus on low miss rate to avoid main memory access
  - hit time has less overall impact
Associative Caches

- In addition to direct mapped, two more cache organizations
  - Fully associative caches:
    - allow a given block to go in any cache entry
    - requires all entries to be searched at once
    - tag comparator per entry (expensive)
  - n-way set associative caches:
    - each set contains n entries (blocks)
    - block number determines which set in the cache
      set number = (block number) mod (number of sets)
    - search only all entries in a given set at once
    - n comparators (less expensive)

Illustration of Cache Organizations

4-Way Set Associate Cache with 4-bB Blocks

- There are 256 sets
- Indexing sets

Set Associative Cache: Example

- Consider 2-way set associative cache with 4 sets, and 8-byte blocks. Assume 16-bit address.

  a. provide address format.
  - 8 byte blocks → 3-bit block offset; 4 sets → 2-bit index
  - 4-byte data (words) → 2-bit byte offset
  - 3-bit block offset = 2-bit byte offset + 1-bit word offset
  - 16-bit address = 11-bit tag + 2-bit index + 1-bit word offset + 2-bit byte offset

  b. Provide cache layout

  c. For address sequence: 8, 0, 52, 20, 56, 16, 24, 116, 20, 8, 16 indicate hit/miss and content of the cache.
Reducing a number of comparisons to reduce cost.

Associativity | Location method | Tag comparisons
---|---|---
Direct mapped | Index entry | 1
n-way set associative | Index a set, then search entries within the set | n
Fully associative | Search all entries | Number of entries

Comparing Finding a Block

- Refers to the situation when on a miss there is no room for a new block and one of existing blocks in the cache has to be removed from caches.
- Direct mapped cache: no choice.
- Set or fully associative caches:
  - choose among entries in the set.
- Least-recently used (LRU) replacement policy:
  - choose the one unused for the longest time,
  - simple for 2-way, manageable for up to 16-way, probably too inefficient beyond that;
- Random replacement policy:
  - gives approximately the same performance as LRU for high associativity.

Replacement Policy