Chapter 7
Memories: Review

- **SRAM:**
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)

- **DRAM:**
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)
## Time and Cost: 1997

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Access Time</th>
<th>Cost Band</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>5–20 ns</td>
<td>$100–$250/Mbyte</td>
</tr>
<tr>
<td>DRAM</td>
<td>60–120 ns</td>
<td>$5–$10/Mbyte</td>
</tr>
<tr>
<td>Mag Disk</td>
<td>10–20 ms</td>
<td>$0.10–$0.20/Mbyte</td>
</tr>
</tbody>
</table>

*Relative costs change slowly.*
Cache

- A faster, smaller memory placed closer to the CPU than a larger one.

- Can be several levels.

- If the data is available in (fast) cache, use it. Else resort to the next (slower) level.

- Try to keep the most recently used data in the cache, so it can be obtained quickly.
Memory Hierarchy: Theoretical

- CPU
- Level 1
- Level 2
- . . .
- Level n

Levels in the memory hierarchy

Increasing distance from the CPU in access time

Size of the memory at each level
Memory Hierarchy: Practical

May be more levels.
Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  
  temporal locality: it will tend to be referenced again soon
  spatial locality: nearby items will tend to be referenced soon.

*Why does code have locality?*

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
Cache

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level
Direct-Mapped Cache

Address (showing bit positions)
31 30 ⋯ 13 12 11 ⋯ 2 1 0

Hit
Tag
Index

Data

Index
Valid Tag

0
1
2
⋯

⋯

⋯

1021
1022
1023

20
32

=
Memory Access

- Break up the address into tag, index, and byte offset.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
</tr>
</thead>
</table>

- If the entry indicated by index is valid and the tags match, the data is in the cache.

- Otherwise, fetch the word from memory and set the cache line to be valid, contain the tag from the address, and the data from the memory.

- If the fetch was less than a word, choose the appropriate portion starting at the byte offset.

- Writes are placed in both cache and memory.
## Cache Example

<table>
<thead>
<tr>
<th>Ind.</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hits vs. Misses

- Read hits
  - this is what we want!

- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart

- Write hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)

- Write misses:
  - read the entire block into the cache, then write the word
Memory Stores

- Write-through is easier to implement.

- Write-back saves when a value can be changed in cache several times then written to memory once.

- A write-back buffer (a queue) holds data pending its write to memory.
Multi-Word Cache

- Each line of the cache contains several words. An extra field in the address selects the word within the line.

- Information is moved between cache and memory in units of one line.

- Uses locality to advantage.

- When storing a word, write-through must write a line.
Multi-Word Cache

Address (showing bit positions)

31...16  15...4  32  1  0

Index

16

12

2 Byte offset

Hit

Tag

16 bits

V

Tag

128 bits

Data

Block offset

Mux

4K entries

16

32

32

32

32

CSc 314 • T W Bennet • Mississippi College
## Multi-Word Cache Example

<table>
<thead>
<tr>
<th>Ind.</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performace

- Increasing the block size tends to decrease miss rate:

- Use split caches because there is more spatial locality in code:

<table>
<thead>
<tr>
<th>Program</th>
<th>Block size in words</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>1</td>
<td>6.1%</td>
<td>2.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.9%</td>
</tr>
<tr>
<td>spice</td>
<td>1</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.3%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
Memory Organization

a. One-word-wide memory organization

b. Wide memory organization

c. Interleaved memory organization
Fully Associative Cache

31–2

1,0

Hit

Index

Search

V

Tag (30 bits)

Connects to all tags

Data

Data
Associative Cache

- Data may be placed anywhere.
- Each line records what it contains.
- Access must search the lines.
- Search is complicated.
- Compromise: Set-Associative cache.
Set-Associative Cache
Set-Associative Cache

- Each memory location must be cached on a specific line in one of the caches.

- Less to search; Simplified hardware.

- Retains most of the benefit.
### Set-Associative Cache Example

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache

- Example:
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%

- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache
Memory Hierarchy

CPU

Cache

Main Memory

SRAM

DRAM

Virtual Memory

Disk (Swap File)
Virtual Memory Points of View

- Program: Total memory appears larger than it is. *This is the historical view.*

- Cache: Real memory is just a cache for disk.
Virtual Memory

- Machine appears to have more memory that it really does.
- Extra data is stored on disk.
- Data is moved to memory on demand.
Address Space

- *Address space* is a set of legal addresses in some memory.

- A *virtual* address space differs from that of the actual memory. This address space is an abstraction presented to programs.

- Different programs may run in different virtual address spaces on the same machine.

- Different virtual address spaces may share portions of the real address space.
Virtual Address Mapping

Virtual Address Space

Memory

Program

Real Addresses

Virtual Addresses

Disk
Terminology

- **Page**: The unit of storage moved between real memory and disk.

- **Page Frame**: A unit of main memory which can hold a page.

- **Page Fault**: When a program references a page which is not in real memory.

- **Virtual Address**: An address in the virtual address space.

- **Real Address**: An address in real memory.
Virtual Memory Advantages

- Machine appears to have more memory that it really does.
- Eases memory allocation.
- Eases memory protection.
Cache vs. Virtual Memory

- Cache: Hardware Only
- Virtual Memory: Hardware and Software (O/S).
Address Translation

Virtual address

31 30 29 28 27 .............. 15 14 13 12 11 10 9 8 ........ 3 2 1 0

Virtual page number       Page offset

Translation

29 28 27 .............. 15 14 13 12 11 10 9 8 ........ 3 2 1 0

Physical page number       Page offset

Physical address
The page table is maintained in memory by the O/S.
Page Table
Address Translation Example

Page Table

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
</table>

Real Memory

<table>
<thead>
<tr>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000–0FFF</td>
</tr>
<tr>
<td>1000–1FFF</td>
</tr>
<tr>
<td>2000–2FFF</td>
</tr>
<tr>
<td>3000–3FFF</td>
</tr>
</tbody>
</table>

Disk
**Address Translation (1)**

Split virtual address to page num & offset
Look up the page number in the page table
{
    if valid
    Combine to get physical address
    Fetch or store at that address
}

else
{
    Trap to the O/S:
    Read page from disk
    Update page table
    Resume program
}
Too Slow

- Two memory references for each memory reference.
- Add a *Translation Lookaside Buffer (TLB)* to cache page table entries.
- Each TLB entry contains one page table entry.
- The TLB may use any of the previously-discussed cache organizations.
Page Table with TLB
Address Translation (2)

Split virtual address to page num & offset
Look up the page number in the TLB
if present and valid then
    Combine to get physical address
    Fetch or store at that address
else
    Trap to the O/S:
        Find the page number in page table
        if not in memory then
            Read page from disk
            Update page table
        Update TLB
        Resume program

May not trap until the page is found not to be in memory.
Cache, Too!

[Diagram of virtual address translation and cache access]

Virtual address
31 30 29 · · · · · · · · · · 14 13 12 11 10 9 8 · · · 3 2 1 0

Virtual page number | Page offset

Valid/Dirty | Tag | Physical page number

TLB

TLB hit

Physical page number | Page offset

Physical address tag | Cache index | Byte offset

Cache

Valid | Tag | Data

Cache hit

Data

Physical address

CSc 314 · T W Bennet · Mississippi College 41
Role of the Operating System

- On a TLB miss, (may) update the TLB.
- On a page fault, performs disk I/O and updates the page table.
- May schedule another process to run during disk I/O.
Returning Pages to Disk

- Each page table entry contains a *dirty bit* indicating if the page has been written to.

- When a clean page is moved out of real memory, it can be discarded.

- When a dirty page is moved out of real memory, it must be written to disk.
Choosing a Page Frame

- Least-Recently Used: Best, too expensive.

- Approximate: Add a used bit to the TLB. O/S can estimate relative ages.

- May prefer to replace clean pages.
Memory Protection

Preventing processes from interfering with each other and the O/S.

- Separate virtual spaces: Processes cannot see each other’s memory.

- Add permission bits to the page table entries.

  For example, user processes are forbidden to write the O/S address space.
Segmentation

- Some systems group pages into “segments.” Segment table entries point to page table entries for the segment.

- Avoids very large page tables.
Cache Policy Decisions

- Where can a block be placed?
- How is the block found?
- Which block to replace on a miss?
- How do we handle writes?
Where can a block be placed?

- Fixed Place. *Standard cache.*
- Small Set of Places. *Set associative cache.*
-Anywhere. *Fully associative cache; virtual memory.*
How is the block found?

- Fixed: *No Problem*.

- Otherwise: *Table Look-up*.
Which block to replace on a miss?

- Fixed: *No Choice*
- Otherwise: *LRU, Approximate LRU, Random*
How do we handle writes?

- Write-Through: Write Both levels. *Impractical for virtual memory.*