Chapter Overview

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory
Introduction

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory

The Big Picture: Where are We Now?

The Five Classic Components of a Computer

- Processor
- Control
- Datapath
- Memory
- Input
- Output

• Topics In This Chapter:
  – SRAM Memory Technology
  – DRAM Memory Technology
  – Memory Organization
Introduction

The Big Picture: Where are We Now?

Capacity Speed (latency)
Logic: 2x in 3 years 2x in 3 years
DRAM: 4x in 3 years 2x in 10 years
Disk: 4x in 3 years 2x in 10 years

Technology Trends

### DRAM

<table>
<thead>
<tr>
<th>Year</th>
<th>Size</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64 Kb</td>
<td>250 ns</td>
</tr>
<tr>
<td>1983</td>
<td>256 Kb</td>
<td>220 ns</td>
</tr>
<tr>
<td>1986</td>
<td>1 Mb</td>
<td>190 ns</td>
</tr>
<tr>
<td>1989</td>
<td>4 Mb</td>
<td>165 ns</td>
</tr>
<tr>
<td>1992</td>
<td>16 Mb</td>
<td>145 ns</td>
</tr>
<tr>
<td>1995</td>
<td>64 Mb</td>
<td>120 ns</td>
</tr>
</tbody>
</table>

1000:1! 2:1!
Introduction

Processor-DRAM Memory Gap (latency)

Processor-Memory Performance Gap:
(grows 50% / year)

“Moore’s Law”

μProc
60%/yr.
(2X/1.5yr
(2X/10 yrs)

DRAM
9%/yr.

Who Cares About the Memory Hierarchy?

The Big Picture: Where are We Now?

Introduction

The Big Picture: Where are We Now?

Who Cares About the Memory Hierarchy?

Time

Chap. 5 - Memory
Introduction

The Big Picture: Where are We Now?

Today’s Situation: Microprocessor

- Rely on caches to bridge gap
- Microprocessor-DRAM performance gap
  - time of a full cache miss in instructions executed

1st Alpha (7000): $340 \text{ ns}/5.0 \text{ ns} = 68 \text{ clks} \times 2 \text{ or } 136 \text{ instructions}$
2nd Alpha (8400): $266 \text{ ns}/3.3 \text{ ns} = 80 \text{ clks} \times 4 \text{ or } 320 \text{ instructions}$
3rd Alpha (t.b.d.): $180 \text{ ns}/1.7 \text{ ns} = 108 \text{ clks} \times 6 \text{ or } 648 \text{ instructions}$

- $1/2X$ latency x $3X$ clock rate x $3X$ Instr/clock $\Rightarrow -5X$
Levels of the Memory Hierarchy

- **CPU Registers**
  - 100s Bytes
  - 1s ns
  - Capacity: 100s Bytes
  - Access Time: 1s ns
  - Cost: 1-0.1 cents/bit

- **Cache**
  - K Bytes
  - 4 ns
  - 1-0.1 cents/bit

- **Main Memory**
  - M Bytes
  - 100ns-300ns
  - $0.0001-.0001 cents/bit

- **Disk**
  - G Bytes, 10 ms (10,000,000 ns)
  - -5 to -6
  - 10^-5 - 10^-6 cents/bit

- **Tape**
  - infinite sec to min
  - 10^-3 - 10^-6
  - 10^-3 - 10^-6 cents/bit

Lower Level: faster

Upper Level: Larger

The Big Picture: Where are We Now?
In this section we will:

Learn lots of definitions about caches – you can’t talk about something until you understand it (this is true in computer science at least!)

Answer some fundamental questions about caches:

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*
The Principle of Locality:
- Program access a relatively small portion of the address space at any instant of time.

Two Different Types of Locality:
- Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
- Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

Last 15 years, HW relied on locality for speed
The ABCs of Caches

Definitions

Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate**: $1 - \text{(Hit Rate)}$
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block to the processor
- **Hit Time << Miss Penalty (500 instructions on 21264!)**
The ABCs of Caches

Definitions

Cache Measures

• **Hit rate**: fraction found in that level
  – So high that usually talk about **Miss rate**
  – Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory

• Average memory-access time
  = Hit time + Miss rate x Miss penalty
  (ns or clocks)

• **Miss penalty**: time to replace a block from lower level, including time to replace in CPU
  – **access time**: time to lower level
    = f(latency to lower level)
  – **transfer time**: time to transfer block
    = f(Bandwidth between upper & lower levels)
The ABCs of Caches

Definitions

Simplest Cache: Direct Mapped

4 Byte Direct Mapped Cache

Memory Address  Memory

0 1 2 3 4 5 6 7 8 9 A B C D E F

• Location 0 can be occupied by data from:
  – Memory location 0, 4, 8, ... etc.
  – In general: any memory location whose 2 LSBs of the address are 0s
  – Address<1:0> => cache index

• Which one should we place in the cache?
• How can we tell which one is in the cache?
1 KB Direct Mapped Cache, 32B blocks

- For a $2^N$ byte cache:
  - The uppermost $(32 - N)$ bits are always the Cache Tag
  - The lowest $M$ bits are the Byte Select (Block Size = $2^M$)

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Example: 0x50</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0x01</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Valid Bit | Cache Tag | Cache Data
---|-----------|-----------
0x50       |           | Byte 31   |
|           |           | Byte 1    |
|           |           | Byte 0    |
|           |           | Byte 63   |
|           |           | Byte 33   |
|           |           | Byte 32   |
|           |           | Byte 1023 |
|           |           | Byte 992  |

Stored as part of the cache “state”
The ABCs of Caches

Definitions

Simplest Cache: Direct Mapped

16K Byte Direct Mapped Cache

Cache Index

0
1
~
511

Chap. 5 - Memory
The ABCs of Caches

Definitions

Simplest Cache: Direct Mapped

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>1413</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>32768</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>49152</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

Tag

Index

Loc. In Cache Line

Chap. 5 - Memory
The ABCs of Caches

Set Associative Memory

4 Byte 2-way Set Associative Cache

Memory Address  Memory

0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F

- Location 0 can be occupied by data from:
  - Memory location 0, 2, 4, 6, 8, ... etc.
  - In general: any memory location whose 1 LSBs of the address are 0s
  - Address<0> => cache index

- Which one should we place in the cache?
- How can we tell which one is in the cache?
The ABCs of Caches

Definitions

Set Associative Cache

16K Byte 4-way Set Associative Cache

Cache Index

0
1
~
127

Why???

Direct Mapped

000000000000000001 000000000 00000

00000000000000000100 0000000 00000

Direct Mapped

00000000000000000100000000000

00000000000000000100000000000
## The ABCs of Caches

### Definitions

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Memory</th>
<th>4-Way Set Associative Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>4096</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>12288</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>20480</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>24576</td>
<td></td>
<td>00000000000000000000000000</td>
</tr>
<tr>
<td>16384</td>
<td>00000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td>16415</td>
<td>00000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>00000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td>12288</td>
<td>00000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td>16416</td>
<td>00000000000000000000000000</td>
<td></td>
</tr>
</tbody>
</table>

Chap. 5 - Memory 4-Way Set Associative
The ABCs of Caches

Q1: Where Can A Block Be Placed In A Cache?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

![Diagram showing cache memory mapping]

FIGURE 5.2 This example cache has eight block frames and memory has 32 blocks.
Two-way Set Associative Cache

- **N-way set associative**: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- **Example: Two-way set associative cache**
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result
Disadvantage of Set Associative Cache

- **N-way Set Associative Cache v. Direct Mapped Cache:**
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss

- **In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:**
  - Possible to assume a hit and continue. Recover later if miss.
The ABCs of Caches

Q2: How is a block found if it is in the cache?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

This is Figure 5.3
The ABCs of Caches

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Associativity:</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>

This is Figure 5.4
The ABCs of Caches

Q4: What happens on a write?

• **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.

• **Write back**—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  – is block clean or dirty?

• **Pros and Cons of each?**
  – WT: read misses cannot result in writes
  – WB: no repeated writes to same location

• **WT always combined with write buffers so that don’t wait for lower level memory**
• **A Write Buffer is needed between the Cache and Memory**
  – Processor: writes data into the cache and the write buffer
  – Memory controller: write contents of the buffer to memory

• **Write buffer is just a FIFO:**
  – Typical number of entries: 4
  – Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

• **Memory system designer’s nightmare:**
  – Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  – Write buffer saturation
Write-miss Policy: Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we read in the block?
    - Yes: Write Allocate
    - No: Write Not Allocate

Q5: What happens on a write?
• Suppose a processor executes at
  – Clock Rate = 1000 MHz (1 ns per cycle)
  – CPI = 1.0
  – 50% arithmetic/logic, 30% load/store, 20% control
• Suppose that 10% of memory operations get 100 cycle miss penalty

• CPI = ideal CPI + average stalls time per instruction
  = 1.0(cycle)
• +( 0.30 (data-operations/instruction)
  x 0.10 (miss/data-op) x 100 (cycle/miss) )
  = 1.0 cycle + 3.0 cycle
  = 4.0 cycle

• 75% of the time the processor is stalled waiting for memory!

• a 1% instruction miss rate would add an additional 1.0 cycles to the CPI!
Which has a lower miss rate:

- A 16-KB instruction cache with a 16-KB data cache, or
- A 32-KB unified cache?

Assume a hit takes 1 clock cycle and the miss penalty is 50 cycles.
Assume a load or store takes 1 extra clock cycle on a unified cache since there is only one cache port.
Assume 75% of memory accesses are instruction references.

\[
(75\% \times 0.64\%) + (25\% \times 6.47\%) = 2.10\%
\]

Average memory access time (split)
\[
= 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50) = 0.990 + 1.059 = 2.05
\]

Average memory access time (unified)
\[
= 75\% \times (1 + 1.99\% \times 50) + 25\% \times (1 + 1 + 1.99\% \times 50) = 1.496 + 0.749 = 2.24
\]

<table>
<thead>
<tr>
<th>Size</th>
<th>Instruction Cache</th>
<th>Data Cache</th>
<th>Unified Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 KB</td>
<td>1.10%</td>
<td>10.19%</td>
<td>4.57%</td>
</tr>
<tr>
<td>16KB</td>
<td>0.64%</td>
<td>6.47%</td>
<td>2.87%</td>
</tr>
<tr>
<td>32 KB</td>
<td>0.39%</td>
<td>4.82%</td>
<td>1.99%</td>
</tr>
<tr>
<td>64KB</td>
<td>0.15%</td>
<td>3.77%</td>
<td>1.35%</td>
</tr>
</tbody>
</table>
The ABCs of Caches

The next few sections look at ways to improve cache and memory access times.

**Average Memory Access Time** = Hit Time + Miss Rate * Miss Penalty

Does this equation make sense??

\[
CP\text{time} = IC \times (CPI_{\text{Execution}} + \frac{\text{MemoryAccess}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty} \times \text{ClockCycleTime})
\]
Reducing Cache Misses

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory

Classifying Misses: 3 Cs

- **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses.* *(Misses in even an Infinite Cache)*

- **Capacity**—If the cache cannot contain all the blocks needed during execution of a program, *capacity misses* will occur due to blocks being discarded and later retrieved. *(Misses in Fully Associative Size X Cache)*

- **Conflict**—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses.* *(Misses in N-way Associative, Size X Cache)*
Reducing Cache Misses

Classifying Misses: 3 Cs

3Cs Absolute Miss Rate (SPEC92)

Compulsory vanishingly small

Chap. 5 - Memory
Reducing Cache Misses

Classifying Misses: 3 Cs

2:1 Cache Rule

miss rate 1-way associative cache size $X$
= miss rate 2-way associative cache size $X/2$

Capacity

Conflict

1-way
2-way
4-way
8-way

Cache Size (KB)

Chap. 5 - Memory
Reducing Cache Misses

Classifying Misses: 3 Cs

3Cs Relative Miss Rate

Flaws: for fixed block size
Good: insight => invention

Cache Size (KB)

Chap. 5 - Memory
Reducing Cache Misses

1. Larger Block Size

Using the principle of locality. The larger the block, the greater the chance parts of it will be used again.

<table>
<thead>
<tr>
<th>Block Size (bytes)</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0%</td>
</tr>
<tr>
<td>32</td>
<td>5%</td>
</tr>
<tr>
<td>64</td>
<td>10%</td>
</tr>
<tr>
<td>128</td>
<td>15%</td>
</tr>
<tr>
<td>256</td>
<td>20%</td>
</tr>
<tr>
<td>1K</td>
<td>25%</td>
</tr>
<tr>
<td>4K</td>
<td>20%</td>
</tr>
<tr>
<td>16K</td>
<td>15%</td>
</tr>
<tr>
<td>64K</td>
<td>10%</td>
</tr>
<tr>
<td>256K</td>
<td>5%</td>
</tr>
</tbody>
</table>

Size of Cache

- 1K
- 4K
- 16K
- 64K
- 256K
Reducing Cache Misses

2. Higher Associativity

• **2:1 Cache Rule:**
  Miss Rate Direct Mapped cache size $N$
  $=$ Miss Rate 2-way cache size $N/2$

• **But Beware:** Execution time is the only final measure we can believe!
  – Will Clock Cycle time increase as a result of having a more complicated cache?
  – Hill [1988] suggested hit time for 2-way vs. 1-way is:
    external cache +10%,
    internal + 2%
Reducing Cache Misses

Example: Avg. Memory Access Time vs. Miss Rate

The time to access memory has several components. The equation is:

\[
\text{Average Memory Access Time} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty}
\]

The miss penalty is 50 cycles.

See data on next page.

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.10</td>
</tr>
<tr>
<td>3</td>
<td>1.12</td>
</tr>
<tr>
<td>8</td>
<td>1.14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache size (KB)</th>
<th>One-way</th>
<th>Two-way</th>
<th>Four-way</th>
<th>Eight-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.65</td>
<td>6.60</td>
<td>6.22</td>
<td>5.44</td>
</tr>
<tr>
<td>2</td>
<td>5.90</td>
<td>4.90</td>
<td>4.62</td>
<td>4.09</td>
</tr>
<tr>
<td>4</td>
<td>4.60</td>
<td>3.95</td>
<td>3.57</td>
<td>3.19</td>
</tr>
<tr>
<td>8</td>
<td>3.30</td>
<td>3.00</td>
<td>2.87</td>
<td>2.59</td>
</tr>
<tr>
<td>16</td>
<td>2.45</td>
<td>2.20</td>
<td>2.12</td>
<td>2.04</td>
</tr>
<tr>
<td>32</td>
<td>2.00</td>
<td>1.80</td>
<td>1.77</td>
<td>1.79</td>
</tr>
<tr>
<td>64</td>
<td>1.70</td>
<td>1.60</td>
<td>1.57</td>
<td>1.59</td>
</tr>
<tr>
<td>128</td>
<td>1.50</td>
<td>1.45</td>
<td>1.42</td>
<td>1.44</td>
</tr>
</tbody>
</table>
## 2. Higher Associativity

### Example: Avg. Memory Access Time vs. Miss Rate

<table>
<thead>
<tr>
<th>Cache size</th>
<th>Degree associative</th>
<th>Total miss rate</th>
<th>Compulsory</th>
<th>Capacity</th>
<th>Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KB</td>
<td>1-way</td>
<td>0.133</td>
<td>0.002</td>
<td>1%</td>
<td>0.080</td>
</tr>
<tr>
<td>1 KB</td>
<td>2-way</td>
<td>0.105</td>
<td>0.002</td>
<td>2%</td>
<td>0.080</td>
</tr>
<tr>
<td>1 KB</td>
<td>4-way</td>
<td>0.095</td>
<td>0.002</td>
<td>2%</td>
<td>0.080</td>
</tr>
<tr>
<td>1 KB</td>
<td>8-way</td>
<td>0.087</td>
<td>0.002</td>
<td>2%</td>
<td>0.080</td>
</tr>
<tr>
<td>2 KB</td>
<td>1-way</td>
<td>0.098</td>
<td>0.002</td>
<td>2%</td>
<td>0.044</td>
</tr>
<tr>
<td>2 KB</td>
<td>2-way</td>
<td>0.076</td>
<td>0.002</td>
<td>2%</td>
<td>0.044</td>
</tr>
<tr>
<td>2 KB</td>
<td>4-way</td>
<td>0.064</td>
<td>0.002</td>
<td>3%</td>
<td>0.044</td>
</tr>
<tr>
<td>2 KB</td>
<td>8-way</td>
<td>0.054</td>
<td>0.002</td>
<td>4%</td>
<td>0.044</td>
</tr>
<tr>
<td>4 KB</td>
<td>1-way</td>
<td>0.072</td>
<td>0.002</td>
<td>3%</td>
<td>0.031</td>
</tr>
<tr>
<td>4 KB</td>
<td>2-way</td>
<td>0.057</td>
<td>0.002</td>
<td>3%</td>
<td>0.031</td>
</tr>
<tr>
<td>4 KB</td>
<td>4-way</td>
<td>0.049</td>
<td>0.002</td>
<td>4%</td>
<td>0.031</td>
</tr>
<tr>
<td>4 KB</td>
<td>8-way</td>
<td>0.039</td>
<td>0.002</td>
<td>5%</td>
<td>0.031</td>
</tr>
<tr>
<td>8 KB</td>
<td>1-way</td>
<td>0.046</td>
<td>0.002</td>
<td>4%</td>
<td>0.023</td>
</tr>
<tr>
<td>8 KB</td>
<td>2-way</td>
<td>0.038</td>
<td>0.002</td>
<td>5%</td>
<td>0.023</td>
</tr>
<tr>
<td>8 KB</td>
<td>4-way</td>
<td>0.035</td>
<td>0.002</td>
<td>5%</td>
<td>0.023</td>
</tr>
<tr>
<td>8 KB</td>
<td>8-way</td>
<td>0.029</td>
<td>0.002</td>
<td>6%</td>
<td>0.023</td>
</tr>
</tbody>
</table>
• How to combine fast hit time of direct mapped yet still avoid conflict misses?

• Add buffer to place data discarded from cache
• A 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
• Used in Alpha, HP machines.

• In effect, this gives the same behavior as associativity, but only on those cache lines that really need it.
Reducing Cache Miss Penalty

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory

Time to handle a miss is becoming more and more the controlling factor. This is because of the great improvement in speed of processors as compared to the speed of memory.

Average Memory Access Time
= Hit Time + Miss Rate * Miss Penalty
Reducing Cache Miss Penalty

- Write through with write buffers offer RAW conflicts with main memory reads on cache misses
- If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
- Check write buffer contents before read; if no conflicts, let the memory access continue
- Write Back?
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
Reducing Cache Miss Penalty

- Don’t have to load full block on a miss
- Have valid bits per subblock to indicate valid
- (Originally invented to reduce tag storage)
Reducing Cache Miss Penalty

- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first

- Generally useful only in large blocks,
- Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart
Reducing Cache Miss Penalty

• L2 Equations
  
  \[ \text{Average Memory Access Time} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]

  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]

  \[ \text{Average Memory Access Time} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} + \text{Miss Penalty}_{L2}) \]

• Definitions:
  
  – \textit{Local miss rate}—misses in this cache divided by the total number of memory accesses \textit{to this cache} (Miss rate\text{L2})
  
  – \textit{Global miss rate}—misses in this cache divided by the total number of memory accesses \textit{generated by the CPU} (Miss Rate\text{L1} \times Miss Rate\text{L2})
  
  – Global Miss Rate is what matters
Comparing Local and Global Miss Rates

- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don’t use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction
Reducing Hit Time

This is about how to reduce time to access data that IS in the cache.

What techniques are useful for quickly and efficiently finding out if data is in the cache, and if it is, getting that data out of the cache.

Average Memory Access Time
= Hit Time + Miss Rate * Miss Penalty
Reducing Hit Time

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?
  - Small data cache and clock rate

- Direct Mapped, on chip

- Since most data DOES hit the cache, saving a cycle on data access in the cache is a significant result.
Reducing Hit Time

- **Send virtual address to cache?** Called *Virtually Addressed Cache* or just *Virtual Cache vs. Physical Cache*
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - Cost is time to flush + “compulsory” misses from empty cache
  - Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address
  - I/O must interact with cache, so need virtual address

- **Solution to aliases**
  - HW guarantees (anti-aliasing) that every cache block has unique physical address
  - SW guarantee: lower n bits must have same address; as long as covers index field & direct mapped, they must be unique; called *page coloring*

- **Solution to cache flush**
  - Add *process identifier tag* that identifies process as well as address within process: can’t get a hit if wrong process
Reducing Hit Time

Avoid Address Translation During Indexing of Cache

How to do parallel operations with Virtually Addressed Caches

Conventional Organization

Virtually Addressed Cache
Translate only on miss

Synonym Problem

Overlap $ access with VA translation:
requires $ index to remain invariant across translation
Reducing Hit Time

Avoid Address Translation During Indexing of Cache

Fast Cache Hits by Avoiding Translation: Process ID impact

Black is uni-process
Light Gray is multi-process when flushing cache
Dark Gray is multi-process when using Process ID tag
Y axis: Miss Rates up to 20%
X axis: Cache size from 2 KB to 1024 KB
Reducing Hit Time

- Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update
- Only STORES in the pipeline; empty during a miss

Store r2, (r1)
Add
Sub
Store r4, (r3)

Check r1
--
M[r1]<-r2&
Check r3

- In shade is “Delayed Write Buffer”; must be checked on reads; either complete write or read from buffer
## Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Priority to Read Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Subblock Placement</td>
<td>+</td>
<td>+</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Second Level Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Small &amp; Simple Caches</td>
<td>-</td>
<td>+</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Avoiding Address Translation</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pipelining Writes</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Chap. 5 - Memory
Main Memory

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory

- Performance of Main Memory:
  - **Latency**: Cache Miss Penalty
    - **Access Time**: time between request and word arrives
    - **Cycle Time**: time between requests
  - **Bandwidth**: I/O & Large Block Miss Penalty (L2)

- Main Memory is **DRAM**: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - **RAS** or **Row Access Strobe**
    - **CAS** or **Column Access Strobe**

- Cache uses **SRAM**: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor/bit, area is 10X)
  - Address not divided: Full address

- **Size**: DRAM/SRAM - 4-8,
  - **Cost/Cycle time**: SRAM/DRAM - 8-16
Main Memory

- “Out-of-Core”, “In-Core,” “Core Dump”?  
- “Core memory”?  
- Non-volatile, magnetic  
- Lost to 4 Kbit DRAM (today using 64Kbit DRAM)  
- Access time 750 ns, cycle time 1500-3000 ns
• Square root of bits per RAS/CAS
Main Memory

DRAM Performance

- A 60 ns ($t_{RAC}$) DRAM can
  - perform a row access only every 110 ns ($t_{RC}$)
  - perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).
    - In practice, external address delays and turning around buses make it 40 to 50 ns

- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead!
Main Memory

**Simple:**
- CPU, Cache, Bus, Memory same width (32 or 64 bits)

**Wide:**
- CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits; UtraSPARC 512)

**Interleaved:**
- CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is word interleaved
Main Memory

- Timing model (word size is 32 bits)
  - 1 to send address,
  - 6 access time, 1 to send data
  - Cache Block is 4 words

- **Simple M.P.** = 4 x (1+6+1) = 32
- **Wide M.P.** = 1 + 6 + 1 = 8
- **Interleaved M.P.** = 1 + 6 + 4x1 = 11
Main Memory

• Memory banks for independent accesses vs. faster sequential accesses
  – Multiprocessor
  – I/O
  – CPU with Hit under n Misses, Non-blocking Cache
• Superbank: all memory active on one block transfer (or Bank)

• Bank: portion within a superbank that is word interleaved
Permits a program's memory to be physically noncontiguous so it can be allocated from wherever available. This avoids fragmentation and compaction.

WHY VIRTUAL MEMORY?

- We've previously required the entire logical space of the process to be in memory before the process could run. We will now look at alternatives to this.

- Most code/data isn't needed at any instant, or even within a finite time - we can bring it in only as needed.

VIRTUES

- Gives a higher level of multiprogramming

- The program size isn't constrained (thus the term 'virtual memory'). Virtual memory allows very large logical address spaces.

- Swap sizes smaller.
Virtual Memory

Virtual memory is the conceptual separation of user logical memory from physical memory. Thus we can have large virtual memory on a small physical memory.
Virtual Memory

Permits a program's memory to be physically noncontiguous so it can be allocated from wherever available. This avoids fragmentation and compaction.

Frames = physical blocks
Pages = logical blocks

Size of frames/pages is defined by hardware (power of 2 to ease calculations)

HARDWARE
An address is determined by:

page number (index into table) +
offset

----> mapping into --->
base address (from table) + offset.
Virtual Memory

Paging Example - 32-byte memory with 4-byte pages

Logical Memory

0 a
1 b
2 c
3 d
4 e
5 f
6 g
7 h
8 l
9 j
10 k
11 l
12 m
13 n
14 o
15 p

Page Table

0
1
2
3

5
6
1
2

Physical Memory

0
4
8
12
16
20
24
28

Virtual Memory

Chap. 5 - Memory
Virtual Memory

IMPLEMENTATION OF THE PAGE TABLE

– A 32 bit machine can address 4 gigabytes which is 4 million pages (at 1024 bytes/page). WHO says how big a page is, anyway?
– Could use dedicated registers (OK only with small tables.)
– Could use a register pointing to table in memory (slow access.)
– Cache or associative memory (TLB = Translation Lookaside Buffer): simultaneous search is fast and uses only a few registers.
IMPLEMENTATION OF THE PAGE TABLE

Issues include:

- key and value
- hit rate 90 - 98% with 100 registers
- add entry if not found

Effective access time = \( \%\text{fast} \times \text{time\_fast} + \%\text{slow} \times \text{time\_slow} \)

Relevant times:
- 20 nanoseconds to search associative memory – the TLB.
- 200 nanoseconds to access memory and bring it into TLB for next time.

Calculate time of access:
- hit = 1 search + 1 memory reference
- miss = 1 search + 1 mem reference(of page table) + 1 mem reference.
Virtual Memory

SHARED PAGES

Data occupying one physical page, but pointed to by multiple logical pages.

Useful for common code - must be write protected. (NO write-able data mixed with code.)

Extremely useful for read/write communication between processes.
INVERTED PAGE TABLE:

One entry for each real page of memory.

Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Essential when you need to do work on the page and must find out what process owns it.

Use hash table to limit the search to one - or at most a few - page table entries.
Virtual Memory

MULTILEVEL PAGE TABLE

A means of using page tables for large address spaces.
HARDWARE  -- Must map a dyad (segment / offset) into one-dimensional address.

Virtual Memory

Chap. 5 - Memory
Basic Issues in VM System Design

- Size of information blocks that are transferred from secondary to main storage \((M)\)
- Block of information brought into \(M\), and \(M\) is full, then some region of \(M\) must be released to make room for the new block → replacement policy
- Which region of \(M\) is to hold the new block --> placement policy
- Missing item fetched from secondary memory only on the occurrence of a fault --> demand load policy

Questions about Memory

Virtual Memory
4 Questions for Virtual Memory

- Q1: Where can a block be placed in the upper level?
  Fully Associative, Set Associative, Direct Mapped

- Q2: How is a block found if it is in the upper level?
  Tag/Block

- Q3: Which block should be replaced on a miss?
  Random, LRU

- Q4: What happens on a write?
  Write Back or Write Through (with Write Buffer)
It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible
A Brief Detour

Why access cache with PA at all? VA caches have a problem!

**synonym / alias problem:** two different virtual addresses map to
same physical address => two different cache entries holding
data for the same physical address!

For update: must update all cache entries with same
physical address or memory becomes inconsistent

Determining this requires significant hardware, essentially an
associative lookup on the physical address tags to see if you
have multiple hits; or

**Software enforced alias boundary:** same least significant bit of VA &
PA > cache size
A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access</th>
</tr>
</thead>
</table>

Really just a cache on the page table mappings

TLB access time comparable to cache access time
(much less than main memory access time)
Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped.

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

Translation with a TLB

CPU → TLB Lookup → Cache → Main Memory

VA → PA

hit

miss

hit

1/2 t

20 t

Chap. 5 - Memory
Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access:

high order bits of the VA are used to look in the TLB while low order bits are used as index into cache
IF cache hit AND (cache tag = PA) then deliver data to CPU
ELSE IF [cache miss OR (cache tag = PA)] and TLB hit THEN access memory with the PA from the TLB
ELSE do standard VA translation
Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation.

This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache.

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

- This bit is changed by VA translation, but is needed for cache lookup.

Solutions:
- go to 8K byte page sizes;
- go to 2 way set associative cache; or

Diagram:
- 2 way set assoc cache
Protection and Examples

The Goal:

One process should not interfere with another

Process model
- privileged kernel
- independent user processes

Privileges vs policy
- architecture provided primitives
- OS implements policy
- problems arise when h/w implements policy
Protection and Examples

Protection Primitives

user vs kernel
• at least one privileged mode
• special case of rings
• usually implemented as mode bits

how do we switch to kernel mode?
• protected ``gates''
• change mode and continue at pre-determined address

h/w to compare mode bits to access rights
• only access certain resources in kernel mode
Protection and Examples

Protection Primitives

base and bounds
- privileged registers
- base $\leq$ address $\leq$ bounds

segmentation
- multiple base and bound registers
- protection bits for each segment

page-level protection
- protection bits in page entry table
- cache them in TLB for speed
Pentium contains:

- Four segments – a program can run in any of them.
- Four segments – a program may or may not be able to touch data in a particular segment.
- How would a user and an operating system use these features?
Protection and Examples

Segment register contains:
- Base
- Limit
- Present
- Readable
- Writable
Summary

5.1 Introduction
5.2 The ABCs of Caches
5.3 Reducing Cache Misses
5.4 Reducing Cache Miss Penalty
5.5 Reducing Hit Time
5.6 Main Memory
5.7 Virtual Memory
5.8 Protection and Examples of Virtual Memory