Cache Memory

CS 365 Lecture 11
Prof. Yih Huang

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)
Exploiting Memory Hierarchy

- Users want large and fast memories!

  SRAM access times are 2 - 25ns at cost of $100 to $250 per Mbyte.
  DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
  Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

Speed Discrepancy

- Main memory is made by DRAM.
- It is however too slow for the processor.
- In our example datapaths, we have assumed that memory reads/writes are done in one processor cycle time.
  - This is way out of mark in reality

- Solution?
Memory Hierarchy

- Use small amount of fast, expensive memory to keep frequently used data close to processor.
- Use large amount of slow, inexpensive memory for the rest of the data.

Locality

- An observation 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.
Focus of this course: upper two levels
- Cache (focus of this lecture); SRAM
- Main memory; DRAM

In 471, you will learn extensions to the memory hierarchy to include disks

---

The main memory is divided into blocks.
Cache memory keeps a subset of the memory blocks.
Blocks are dynamically moved in and out of cache.
- Block sizes depend on implementations.
- When the processor needs a data or instruction,
  - Identify the block of the data
  - Try its luck to see if the block is in cache
  - If yes, this is a **cache hit** --- get the data from cache
  - Otherwise, it is a **cache miss** --- get the data from main memory.

- Blocks have to be moved in and out of cache constantly, to keep the *presently* important data/instruction close to the processor.
- For most ISAs (but not all), this management of cache memory is entirely by hardware
  - Cache is transparent to programmers, besides performance boosts
Hits and Misses

- Read hits: no worries
- Read misses: stall the CPU, fetch block from memory, deliver to cache, read again
  - If all cache blocks are in use, which one to purge?
  - This is called the replacement policy

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

Write misses: read the entire block into the cache, then write the word
Design Parameters

- Block size: # of bytes/words per block
- Cache size: # of blocks
- Mapping: the mechanism to keep track of which cache block is to be used to store which main memory block.
- Separate instruction and data caches (split caches) versus an integrated cache
- Replacement policy

Cache Blocks in YH16

- Block size = 16 words
- The main memory has 4096 blocks.
- An address comprises two parts:

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

```plaintext
Block #  Offset
```

```
Block 0
Block 1
Block 2
Main Memory
```
Reminders

- The two parts in addresses are transparent to programmers/compilers.
  - Their existence is only for cache operations.
- Programs still deal with a linear address space that include $2^{16}$ words.

Fully Associative Caches

- A memory block can be stored in any cache block.
- When referencing a memory block, all cache blocks are searched in parallel to see if it contains the block.
Implementation

- A cache block is implemented by
  - An SRAM module to store the present memory block
  - A Valid flag (a 1-bit register) to specify if the cache block presently holds a memory block.
  - A tag register to remember the memory block # presently held, when V=1.

Searching Circuit of Cache Block $i$

- **Addr Port**: Address from processor
- **SRAM module $i$**: to store a block of words
- **Hit $i$**:
  - **MemRead From CPU**
  - **MemWrite From CPU**

 comparator: equal

- **Data Port**: 16 Data
2-Block Fully Associative Cache

2-Block Fully Associative Cache in Operation

Four Block Fully Associative Cache in Operation
Direct Mapped Cache

- Each memory block is mapped to one cache block.
  - It can be stored only in that block.
  - There are many other memory blocks mapped to the same cache block.
- Cache sizes are powers of 2.
- Modula 2 is used to do the mapping

Direct Mapped Cache: An Example

Direct Mapped Cache with 4 blocks
Four Block Fully Associative Cache in Operation

Tags:

Cache

Main Memory

Block #
Exercise

- Block size = 64
- Cache has 32 blocks
- Calculate
  - The width of a tag memory address
  
  - The width of an SRAM address
  
  - The width (in # of bits) of a tag.

Exercise

- Consider the MIPS architecture.
- Block size = 8 words (32 bytes)
- Cache has 1024 blocks
- Calculate
  - The width of a tag memory address
  
  - The width of an SRAM address
  
  - The width of a tag.
Comparisons

- Fully associative caches
  - Flexibility in using cache blocks
  - Higher hit ratio
  - Use more hardware

- Direct mapped caches
  - Inflexibility in using cache blocks
  - Lower hit ratio
  - Less hardware demanding

Set Associative Caches

- The compromises between fully associative and direct mapped caches
- Cache is divided into sets
- Each memory block is mapped to a particular set
- Each set can have more than one block
  - Number of blocks in a set = associativity of cache
  - Direct mapped caches have an associativity of 1
- Each memory block can be placed in any of the blocks of the set to which it maps
**Associativities**

- **One-way set associative**
  - Direct mapped
  - Block Tag Data
  - Set Tag Data Tag Data

- **Two-way set associative**
  - Set Tag Data Tag Data

- **Four-way set associative**
  - Set Tag Data Tag Data Tag Data Tag Data

- **Eight-way set associative (fully associative)**

---

**Block Replacement Policies**

- What block to replace on a cache miss?
  - We have multiple candidates (unlike direct mapped caches)
  - Random
  - FIFO (First In First Out)
  - LRU (Least Recently Used)

- Typically, cpu uses Random or Approximate LRU because easier to implement in hardware
LRU Replacement

- The processor maintains a counter of memory references.
  - The counter is called the clock, metaphorically.
- Every cache block has a time-of-use register.
- Each time a cache block is accessed, its time of use is set to the current clock.
- Find the block with the earliest time-of-use for replacement.

Example: LRU, block=1

<table>
<thead>
<tr>
<th>Addr</th>
<th>Hit/Miss</th>
<th>Set 0</th>
<th>Set 0</th>
<th>Set 0</th>
<th>Set 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>M</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>M</td>
<td>0000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>H</td>
<td>0000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>M</td>
<td>0000</td>
<td>1000</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>H</td>
<td>0000</td>
<td>1000</td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example: LRU, block=1

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
<th>Set 0</th>
<th>Set 0</th>
<th>Set 1</th>
<th>Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>M</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>M</td>
<td>0000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>H</td>
<td>0000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>M</td>
<td>0000</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>M</td>
<td>1000</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Example: LRU, block=1

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>M</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>M</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>M</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>M</td>
<td>0000</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>M</td>
<td>1000</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Exercise: LRU, block=2, 2-way

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
</tr>
<tr>
<td>011001</td>
<td></td>
</tr>
<tr>
<td>100010</td>
<td></td>
</tr>
<tr>
<td>001101</td>
<td></td>
</tr>
</tbody>
</table>

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

---

### Exercise: LRU, block=4, 1-way

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
</tr>
<tr>
<td>011001</td>
<td></td>
</tr>
<tr>
<td>100010</td>
<td></td>
</tr>
<tr>
<td>001101</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>
### Exercise: LRU, block=4, 2-way

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Exercise: LRU, block=4, 4-way

<table>
<thead>
<tr>
<th>Addr</th>
<th>H/M</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CS 365
**Performance Model**

- **Simplified model:**
  \[
  \text{exec time} = (\text{cycles} + \text{stall cycles}) \times \text{cycle time}
  \]
  \[
  \text{stall cycles} = \# \text{ of instrs} \times \text{miss ratio} \times \text{miss penalty}
  \]

- **Two ways of improving performance:**
  - decreasing the miss ratio
  - decreasing the miss penalty

---

**Performance**

![Graph showing miss rate per type versus cache size (KB).](image-url)
Real World Example: AMD 64 FX

<table>
<thead>
<tr>
<th>CPU</th>
<th>Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 integer reg.</td>
<td></td>
</tr>
<tr>
<td>16 fp reg.</td>
<td></td>
</tr>
<tr>
<td>64 KB Instr. Cache</td>
<td></td>
</tr>
<tr>
<td>2-way associative</td>
<td></td>
</tr>
<tr>
<td>64 KB Data Cache</td>
<td></td>
</tr>
<tr>
<td>2-way associative</td>
<td></td>
</tr>
<tr>
<td>1MB integrated 16-way associative level-2 cache</td>
<td></td>
</tr>
</tbody>
</table>

Discussions

- Split caches allows CPU to fetch instructions and data at the same time, without actually separate memories.
- Some Intel chips even have a 2MB level-3 cache.
More discussions

- Processor speeds continue to increase very fast
  - much faster than either DRAM or disk access times
- How to deal with this growing disparity?
- Trends:
  - synchronous RAMs (provide a burst of data)
  - redesign DRAM chips to provide bursts of data (higher bandwidth)
    - Don’t confuse bandwidth with latency
  - restructure code to increase locality
  - Larger and larger caches on chip