Review

Cache memories
Block or line size = number of bytes per line
Set associativity = number of lines per set
Direct mapped = 1 way set associative (one line per set)
2-way = 2 lines per set
fully associate = 1 set (all line form a single set)

Translating a memory address into a cache address
TAG, set/line index, byte offset

<table>
<thead>
<tr>
<th>16-bit block ID (or tag)</th>
<th>10-bit block address</th>
<th>6-bit byte address</th>
</tr>
</thead>
</table>

There are 512 sets (each with two lines)
One line from each group is a set
We need 9 bits to select a set
CSCE 4610: Computer Architecture

We can reduce cache conflicts by increasing the set size (set associativity).
For example, consider 4-way associative (each set has 4 cache lines).

One block from each group forms a set.

There are 256 sets and we need 8-bits to select a set.

CSCE 4610: Computer Architecture

The 3 C’s Cache Misses

- Can we classify cache misses? (page B-23)
- Compulsory (or Cold)
- Capacity - misses due to the limited size of caches
- Conflict - due to limited associativity

(we will add a 4th C later – Coherency when we deal with multicore)

Once we understand these types of misses, we can then talk of techniques to reduce misses.

Note, reducing miss rates is only one aspect of cache performance.
We also need to understand access times, both on hit and miss (or miss penalty).

Let us look at the table on page B-24. This shows how different cache designs impact the 3 C’s.
CSCE 4610: Computer Architecture

Reading vs Writing

To read:
locate the cache line and read the line (and use the needed byte)
If not in cache, “cache miss”
You can fetch data and Tag together and ignore data if tag does not match

To write:
We need to fetch tag, check if a hit, only then write data
Should we update higher level caches and main-memory also?

If a cache miss, do we simply update higher level memories
(L2 cache, main memory)
or bring the line into cache and then update the cache
only cache or both cache and higher level memories?

Write-through and write-back
Write allocate vs write no allocate
Write-through with write buffers

CSCE 4610: Computer Architecture

Split cache vs Unified Cache (separate Instruction cache and Data cache)

Why is this important?

LW $s2, 0($s1)
Add $s3, $t1, $s2
Addi $s1, $s1, 4
BNE $s3, $t2, loop

LW $s2, 0($s1)
Add $s3, $t1, $s2
Addi $s1, $s1, 4
BNE $s3, $t2, loop

CSCE 4610: Computer Architecture
CSCE 4610: Computer Architecture

**Split cache vs Unified Cache** (separate Instruction cache and Data cache)

An example
36% are load/store instructions and miss penalty is 100 cycles

Should we use unified 32KB cache or separate 16KByte data and 16Kbyte instruction cache?

We will use data from table B.6 on page B-15.

Unified: Miss rate = 43.3 misses per 1000 instructions
Each instruction has 1.36 memory accesses
Miss rate = \( \frac{43.3}{1000} \) = 0.0318 or 3.18%

16K Instruction cache: 3.82 misses per 1000 instructions or Miss rate
= \( \frac{3.82}{1000} \) = 0.00382

16K Data cache: 40.9 misses per 1000 instructions or Miss rate [40.9/1000] = 0.114

How do we combine these miss rates into a single number?

We need on average 1.36 memory access per instruction
\( \frac{1}{1.36} \) = 0.74 or 74% to access are for instructions
0.36/1.36 = 0.26 or 26% accesses are to data

Total miss rate = 0.00382*0.74 + 0.114*0.26 = 0.0.324 or 3.24%

Unified Cache is slightly better (3.18% vs 3.24%)

But let us consider actual CPI related performance. Remember

Memory access time = Hit_time + (miss_rate)* (miss_penalty)

Split cache: 1 + 3.24% *100 = 4.24 cycles

Unified cache: need to add an extra cycle (even on hit) for load and store
1.36 (on hit) + 3.18% *100 = 4.54 cycles

In terms of access times, split caches are better
There is another reason for separating instructions and data

**Different behaviors**

Caches are useful because of a phenomenon known as “locality”

Two types of localities

**Spatial Locality** Nearby data will be needed soon
- Instructions, Arrays, streams

**Temporal Locality** The same data is likely to be used again in near future
- Index variables
- Instructions inside a loop

Do all data items behave the same way?
- Consider Arrays or matrices
- Consider structure or linked lists?

Why not split data cache into array cache, structure cache and scalar cache?

---

Likewise, we can consider separate caches or split caches for each data structure. Stack vs Heap variables

Class Project possibilities
- Use Gleipnir to separate memory accesses into Stack and Heap accesses
- Modify Dinero to use two separate data caches

Appendix B presents 6 ways to improve cache performance

And Chapter 3 adds several more techniques

6 basic techniques from Appendix B

1. Larger blocks size – bring more data when you fetch a block (page B-27)
   - due to locality of data/instructions, you are prefetching data that will be needed
   - miss penalty increases – need to get more data on miss

<table>
<thead>
<tr>
<th>Block size</th>
<th>4K</th>
<th>16K</th>
<th>64K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>8.57%</td>
<td>3.94%</td>
<td>2.04%</td>
<td>1.09%</td>
</tr>
<tr>
<td>32</td>
<td>7.24%</td>
<td>2.87%</td>
<td>1.35%</td>
<td>0.76%</td>
</tr>
<tr>
<td>64</td>
<td>7.00%</td>
<td>2.64%</td>
<td>1.06%</td>
<td>0.51%</td>
</tr>
<tr>
<td>128</td>
<td>7.78%</td>
<td>2.77%</td>
<td>1.02%</td>
<td>0.49%</td>
</tr>
<tr>
<td>256</td>
<td>9.51%</td>
<td>3.29%</td>
<td>1.15%</td>
<td>0.49%</td>
</tr>
</tbody>
</table>

Figure B.11 Actual miss rate versus block size for the five different-sized caches

Larger blocks reduce cold/compulsory misses

But increases miss penalty

<table>
<thead>
<tr>
<th>Block size</th>
<th>4K</th>
<th>16K</th>
<th>64K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>82</td>
<td>8.027</td>
<td>4.231</td>
<td>2.973</td>
</tr>
<tr>
<td>32</td>
<td>84</td>
<td>7.082</td>
<td>3.411</td>
<td>2.134</td>
</tr>
<tr>
<td>64</td>
<td>88</td>
<td>7.160</td>
<td>3.323</td>
<td>1.933</td>
</tr>
<tr>
<td>128</td>
<td>96</td>
<td>8.469</td>
<td>3.659</td>
<td>1.979</td>
</tr>
<tr>
<td>256</td>
<td>112</td>
<td>11.651</td>
<td>4.685</td>
<td>2.288</td>
</tr>
</tbody>
</table>

Figure B.12 Average memory access time versus block size for five different sizes

Miss penalties
- 82 cycles for 16 bytes
- 84 cycles for 32 bytes
- 88 cycles for 64 bytes

More data to bring to cache