Caches: The Basic Idea

- A smaller set of storage locations storing a subset of information from a larger set (memory)
  - Unlike registers or memory, invisible to programmer/ISA

```
memory reference

found (hit)  Is in cache?  no (miss)

read data from cache

"Hit time" (1 cycle)

bring data & more (block) from memory into cache

Spatial locality

"Miss penalty" (10s or 100s cycles)

Temporal locality
```
Our first example:

- Block size is 16 bytes of data (0x10 bytes)
- Memory has 256 bytes (0x100 bytes, 0x10 blocks)
- Cache has 128 bytes (8 blocks)
- “Direct mapped"
Cache Block Placement

- Another similar example:
  - Block size is 16 bytes of data (0x10 bytes)
  - Memory has 256 bytes (0x100 bytes, 0x10 blocks)
  - **Cache has 64 bytes (4 blocks)**
  - “Direct mapped”

Direct-mapped Block Placement

```
Memory Address
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
```

```
Block Index
0 1 2 3
```

```
Memory access [0xD6]
```

Direct-mapped cache
Decoding Memory Address

- Memory address [D6] = [1101 0110]
  - offset within the block
  - cache block index (101 or 01)
  - for identifying the original memory block (1 or 11)

- Given the memory address (e.g. D6)
  - Extract the cache block index (1)
  - Check if the cache block #1 corresponds to memory D0-DF
    - For this, each cache entry remembers the “tag” data (e.g. “3”)
    - If “tag” matches with the first two bits in memory address, HIT
  - Extract the byte within the block with offset address (e.g. 6)

Direct-mapped Block Placement

- Memory Address
- 16 bytes
- Memory access [0xD6]
- Direct-mapped cache
- Tag
- 16 bytes
- Block Index
- 0
- 3
- c.yu91@csuohio.edu
Exercise 1: Make connections!

Exercise 2: Find them!
Decoding Memory Address

- Given the memory address (e.g. A3)
  - Extract the cache block index (2)
  - Check if the cache block #2 corresponds to memory A0–AF
    - Since tag is 3 and the first two bits in memory address is 2, “MISS”

- So, what happens when a miss occurs? (see pages 465-466)
  - The current cache block #2 contains B0–BF
  - Read memory A0–AF and replace the cache block #2
  - Change the Tag to “2”

---

Decoding Memory Address

- Given the memory address (e.g. 0B)
  - Extract the cache block index (0)
  - Check if the cache block #0 corresponds to memory 00–0F
    - Tag is 0 and therefore, “HIT”
  - However, it is possible that Tag=0 because the cache is initially empty
    - This is not a HIT ??? => We need a “valid bit” !
Exercise 3: What is invalid?

Decoding Memory Address

- Given the memory address (e.g. D9)
  - Extract the cache block index (1)
  - Check if the cache block #1 corresponds to memory D0~DF
    - Since “Tag”=3 AND “valid bit”=1, “HIT”

- Given the memory address (e.g. 0B)
  - Extract the cache block index (0)
  - Check if the cache block #0 corresponds to memory 00~0F
    - “Tag”=0 but “valid bit”=0, “MISS”

- So, what happens when a miss occurs?
  - The current cache block #0 contains an invalid data
  - Read memory 00~0F and replace the cache block #0
  - Change the Tag to “0” and set the valid bit to “1”
Exercise 4: What if [D6] is write?

16 bytes

Memory Address
00
10
20
30
40
50
60
70
80
90
A0
B0
C0
D0
E0
F0

16 bytes

Block Index
0
1
2
3
Tag
Valid
0
0
1
1

Direct-mapped cache

Memory

Handling Writes

Write strategy (see pages 466-468)

- When write hit
  - Write through: data is written to both the cache and the memory
  - Write back: data is written only to the cache
    - the modified (dirty) cache block is written to memory when replaced
    - requires dirty bit for each cache block (more complexity)
    - Better performance but semantic problem

- When write miss
  - Write allocate: fetch-on-write - with Write-back ???
  - No write allocate: write-around - with Write-through ???
Cache Block Placement

- 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 4 words of data
  - "direct mapped"

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

  e.g., lots of items in memory share locations in the cache

Single Word Cache Block: Exercise5

- Memory address
  - Memory size?
    (# bits for memory address?)
  - Block size?
    (# bits for offset?)
  - # Memory blocks?
    (# bits for block index?)

- Memory address to Cache address
  - # Cache blocks in direct-mapped cache?
    (# bits for block index?)
  - Which (how many) memory blocks are candidates for cache block #5?
  - Tag size?
  - How memory address is decomposed?
Exercise 6 – Multiword Cache Block

19

c.yu91@csuohio.edu

Block Identification

- Ex) Alpha 21064 (Direct-mapped cache)
  - 32-byte Block size : Requires 5-bit for block offset
  - 8KB cache size : 256 (=8KB/32B) cache blocks
    Requires 8-bit for block index
  - 34-bit memory address supports up to 16GB :
    512M (=16GB/32B) memory blocks
  - Mapping : 2M (=512M/256) candidates for a cache block (compete with each other) => Requires 21-bit to identify

20

c.yu91@csuohio.edu
Cache Performance: Tradeoffs

(1) Increasing block size
   + decreases miss rate, until block gets large (spatial locality)
   - increases miss penalty

(2) Increasing cache size
   + decreases miss rate
   - increases hit time
   - increases hardware cost

(3) Increasing associativity (Section 7.3)
   + increases hit rate
   - increases hit time
   - increases hardware cost

To make exact tradeoffs, need to know specific numbers.
Calculation & measurement
See book for formulae.
(4) 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 Buffer Saturation

Store frequency (w.r.t. time) > 1 / DRAM write cycle
   ➢ If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
     ▪ Write buffer will overflow no matter how big you make it
     ▪ The CPU Cycle Time <= DRAM Write Cycle Time

Solution for write buffer saturation:
   ➢ Use a write back cache
   ➢ Install a second level (L2) cache:
Cache Performance: Tradeoffs

(5) Multiple Levels of Caches

- Emphasize speed: Low hit rate
- Emphasize accuracy: High hit rate

- Larger, slower, cheaper
- Larger line size, higher associativity, more likely to write back

Understanding performance is tricky. Use lots of simulation & testing.

c.yu91@csuohio.edu

---

Cache Performance: Tradeoffs

(6) Independent Instruction & Data Caches

- Possible if locations accessed are independent (different segments)
  - Avoids conflict misses between instructions & data
  - But increases conflict misses among instructions & among data

- Can be of different sizes & strategies
  - Instructions typically have greater locality
  - ICache often smaller than DCache
  - ICache often don’t support writes
  - DCache often support multiple simultaneous accesses (multi-porting)

c.yu91@csuohio.edu
Cache Performance: Tradeoffs

(7) Memory organization

- Make reading multiple words easier by using banks of memory: It can get a lot more complicated...

(a) One-word-wide memory organization

(b) Wide-memory organization

(c) Interleaved memory organization

(8) Performance equation

- Simplified model:

  execution time = (execution cycles + stall cycles) x cycle time

  stall cycles = # of instructions x miss ratio x miss penalty

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

What happens if we increase block size?
Performance

- 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>

Cleveland State University

c.yu91@csuohio.edu