Lecture Topics

- Today: Memory Management
  (Stallings, chapter 7.1-7.4)
- Next: continued

Announcements

- Exam scores sent via email
- Cache covered in Stallings, chapter 1.5-1.6
- Self-Study Exercise #6
- Project #4 (due 10/11)
- Project #5 (due 10/18)
Locality of Reference

- Memory references for both instructions and data values tend to cluster over time.

- Example: once a loop is entered, there is frequent access to a small set of instructions.

- Hence: once an instruction is referenced, it is likely that the instruction (and nearby instructions) will be referenced again in the near future.

Types of Locality

- Temporal locality: same address referenced repeatedly in the near-term future
  - instructions: loops, functions
  - data: variables

- Spatial locality: nearby addresses referenced in the near-term future
  - instructions: sequential execution
  - data: arrays, similar data structures
Cache Memory

- Cache: fast (and thus small and expensive)
- Main memory: slow (and thus larger and cheaper)
- Processor first checks cache for requested word
- If not found in cache, a block of memory containing the word is moved to the cache

Cache and RAM Configuration

Unit of transfer between RAM and cache is one block
Each cache slot holds one block
RAM is viewed as being divided into fixed-size blocks
Read Operation

Load instruction: copy data from RAM to CPU

Check cache first – if desired item is already present in the cache, simply copy item from cache to CPU

If desired item is not already present in the cache, copy a block (item and its neighbors) from RAM to the cache and copy the item to the CPU
Write Operation

Store instruction: copy data from CPU to RAM

Check cache first – if desired item is already present in the cache, simply copy item from CPU to cache

If desired item is not already present in the cache, copy item (and its neighbors) from RAM to the cache and copy the item from the CPU

Write Policies

After a store instruction, cache and RAM are inconsistent: contents of block in cache and RAM are different

Two strategies:

- Write through
- Write back
Write Policies

- **Write through**: whenever a cache block is changed, the block is written (copied) to RAM
- **Write back**: cache block is only written (copied) to RAM when the cache line is evicted (replaced)
  - multiple store instructions can occur before block has to be written to RAM
  - modified bit used to indicate that block has been changed (and must be written to RAM)

Direct Mapping

![Diagram of Direct Mapping](image)

- First $m$ blocks of main memory (equal to size of cache)
- Cache memory
- $b = \text{length of block in bits}$
- $t = \text{length of tag in bits}$
- $m = \text{number of cache lines}$
Direct Mapping

Mapping function:

\[ i = j \mod m \]

where

- \( i \) = cache line number
- \( j \) = main memory block number
- \( m \) = number of lines in the cache

Each block maps to exactly one cache line

Example Configuration

- Cache: 64 KB
- RAM: 16 MB (24-bit addresses)
- Block size: 4 bytes
- Cache is organized as \(2^{14}\) lines, where each line holds 4 bytes
- RAM is viewed as 4M blocks of 4 bytes each
Example

- Address (24 bits) viewed as three fields:
  - Word: 2 bits to identify byte within word
  - Line: 14 bits to identify cache line
  - Tag: 8 bits (remaining bits)

Example (2)

Address: 16339C

In binary:

```
000101100011001110011100
```

Tag: 00010110 (16)
Line: 00110011100111 (0CE7)
Word: 00 (0)
### Example (3)

<table>
<thead>
<tr>
<th>Cache line</th>
<th>Addresses of RAM blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000, 010000, ..., FF0000</td>
</tr>
<tr>
<td>1</td>
<td>000004, 010004, ..., FF0004</td>
</tr>
<tr>
<td>2</td>
<td>000008, 010008, ..., FF0008</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>$2^{14} - 1$</td>
<td>00FFFC, 01FFFC, ..., FFFFFC</td>
</tr>
</tbody>
</table>

![Diagram of memory addressing and cache lines]

Note: Memory address values are in hexadecimal representation; other values are in decimal representation.
Direct-mapped cache

Example

- Address (32 bits) viewed as three fields:
  - Byte offset: 8 bits to identify byte within block
  - Line: 4 bits to identify cache line
  - Tag: 20 bits (remaining bits)

- Example: \texttt{FFF7C408}
  
  \begin{verbatim}
  111111111111011110001000001000100
  \end{verbatim}
### Example (2)

- **How many lines in the cache?**
  
  \[2^4 = 16 \text{ lines}\]

- **How many bytes in one block?**
  
  \[2^8 = 256 \text{ bytes}\]

- **How many control bits in one line?**
  
  \[V + M + \text{Tag} = 1 + 1 + 20 = 22 \text{ bits}\]

- **How many total bits in one line?**
  
  \[\text{control + data} = 22 + 2048 = 2070 \text{ bits}\]

### Example (3)

<table>
<thead>
<tr>
<th>I</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
<th>I</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>FF641</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0004A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0003A</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>FF593</td>
<td>A</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>FFF7C</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>FFF7C</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>00EA1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>00014</td>
<td>D</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>E</td>
<td>1</td>
<td>1</td>
<td>0003A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>1</td>
<td>1</td>
<td>0003A</td>
</tr>
</tbody>
</table>
Example (4)

- Index – line number (not stored)
- Valid bit (V) – initially 0, set to 1 when that entry in the cache is in use
- Modified bit (M) – set to 1 when at least one byte in the block has been modified by a "write" operation (aka dirty bit)
- Tag bits – compared to tag bits from address
- Block – 256 bytes (not shown)

Example (5)

- Consider the cache entry at index 4:

  1  1  FFF7C

  - What are the addresses of the first and last bytes in that cache entry?
    
    first byte: FFF7C400
    
    last byte: FFF7C4FF

  - Has the contents of that cache block been modified?
    
    Yes, M = 1
Example (6)

- Consider a request to read from address 00028A14
  
  Line in address is A, so check cache line at index A:
  
  1  0  00028
  
  Hit: \( V = 1 \) and tag in cache line matches tag in address
  
  Transfer 4 bytes (14, 15, 16, 17) from cache block to CPU

Example (7)

- Consider a request to read from address 0007260C
  
  Line in address is 6, so check cache line at index 6:
  
  0  0  00014
  
  Miss: \( V = 0 \)
  
  Transfer 256 bytes from RAM to cache
  
  Set \( V \) bit to 1
  
  Set \( M \) bit to 0
  
  Set tag to 00072
  
  Transfer 4 bytes (0C, 0D, 0E, 0F) from cache block to CPU
Example (8)

- Consider a request to write to address 0003AED8
  
  Line in address is E, so check cache line at index E:
  
  1 1 0003A
  
  Hit: V = 1 and tag in cache line matches tag in address
  
  Transfer 4 bytes from CPU to cache block (D8, D9, DA, DB)
  
  Set M bit to 1
  
  Note that some of the 256 bytes in the cache block are no longer the same as the corresponding bytes in RAM (copy block to RAM later)

Example (9)

- Consider a request to write to address 0003A344
  
  Line in address is 3, so check cache line at index 3:
  
  0 1 FF593
  
  Miss: V = 0
  
  Transfer 256 bytes from RAM to cache
  
  Set V bit to 1
  
  Set M bit to 1
  
  Set tag to 0003A
  
  Transfer 4 bytes (44, 45, 46, 47) from CPU to cache block
Example (10)

- Consider a request to read from address 002C5934
  
  Line in address is 9, so check cache line at index 9:
  
  1  0  00028
  
  Miss: V = 1, but tags don't match
  
  Transfer 256 bytes from RAM to cache
  Set V bit to 1
  Set M bit to 0
  Set tag to 002C5
  Transfer 4 bytes (34, 35, 36, 37) from cache block to CPU

Example (11)

- Consider a request to read from address 002D1F98
  
  Line in address is F, so check cache line at index F:
  
  1  1  0003A
  
  Miss: V = 1, but tags don't match
  
  Transfer 256 bytes from cache to RAM (write back)
  Transfer 256 bytes from RAM to cache
  Set V bit to 1
  Set M bit to 0
  Set tag to 002D1
  Transfer 4 bytes (98, 99, 9A, 9B) from cache block to CPU
With fully associative mapping, any block can be placed in any cache line.

More flexible than direct mapping, but requires more circuitry to find a particular entry: comparator for every cache line.
Fully Associative Cache
Example

- Address (32 bits) viewed as two fields:
  - Byte offset: 8 bits to identify byte within block
  - Tag: 24 bits (remaining bits)

- Example: FFF7C408
  11111111111110111100010000001000

Example (2)

- Assume 16 lines in the cache
- How many bytes in one block?
  \[ 2^8 = 256 \text{ bytes} \]
- How many control bits in one line?
  \[ V + M + \text{Tag} = 1 + 1 + 24 = 26 \text{ bits} \]
- How many total bits in one line?
  \[ \text{control + data} = 26 + 2048 = 2074 \text{ bits} \]
Example (3)

<table>
<thead>
<tr>
<th>I</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
<th>I</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>FF7814</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0004A7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>000146</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>000286</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0003AF</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>000289</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0003AC</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>FFF7C0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>FF781C</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>000146</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>000142</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>000287</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>000000</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>000000</td>
<td>F</td>
<td>0</td>
<td>0</td>
<td>000000</td>
</tr>
</tbody>
</table>

Example (4)

- Index – not derived from address
- Valid bit (V) – initially 0, set to 1 when that entry in the cache is in use
- Modified bit (M) – set to 1 when at least one byte in the block has been modified by a "write" operation (aka dirty bit)
- Tag bits – compared to tag bits from address
- Block – 256 bytes (not shown)
Example (5)

- Consider the cache entry at index 0:

  1  0  FF7814

  - What are the addresses of the first and last bytes in that cache entry?
    
    first byte: FF781400
    last byte: FF7814FF

  - Has the contents of that cache block been modified?
    
    No, M = 0

Example (6)

- Consider a request to read from address 0003AC64

  Search all cache lines simultaneously, found at 3:

  1  0  0003AC

  Hit: V = 1 and tag in cache line matches tag in address

  Transfer 4 bytes (64, 65, 66, 67) from cache block to CPU
Example (7)

- Consider a request to read from address 0034560C
  
  Search all cache lines simultaneously, not found (miss)
  
  Transfer 256 bytes from RAM to cache
  
  Set V bit to 1
  
  Set M bit to 0
  
  Set tag to 003456
  
  Transfer 4 bytes (0C, 0D, 0E, 0F) from cache block to CPU

  Note: where should the line be placed in the cache?

Summary: Four Questions

1. Where can a block be placed in the upper level? (block placement)

2. How is a block found if it is in the upper level? (block identification)

3. Which block should be replaced on a miss? (block replacement)

4. What happens on a write? (write strategy)
Block Placement

- Direct mapped – exactly one location where a given block can be placed in the upper level
- Fully associative – block can be placed anywhere in the upper level
- Set associative – block can only be placed in a restricted set of locations (block mapped onto a set, and then the block is placed anywhere in that set)

Block Identification

- Valid bit used to indicate whether an entry contains valid data or not
- Tag bits from address compared to tag bits in entry
- All possible locations searched at once:
  - direct mapped – only one location possible
  - fully associative – all locations possible
  - set associative – subset of locations possible
Block Replacement

- Hardware must choose victim (block to be evicted)

- Direct mapped – only one choice

- Fully associative and set associative – choose one of multiple entries
  - Random
  - LRU (least recently used)
  - FIFO (first in, first out)

Write Strategy

- Store operations modify data objects (about 10% of instructions in a typical program)

- Write through – block in upper level and in lower level updated together

- Write back – block in lower level updated when block is evicted from upper level
  - block in upper level may be updated multiple times
  - modified bit used to flag modified blocks