Topics

- Cache review
- Error detecting codes
- Error correcting codes
- Hamming algorithm

What is a Cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers a cache on variables
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
Types of Caches

- Direct-mapped
  - Cache line only has 1 possible location
  - Simple to implement

- Set-associative
  - Cache line has a “set” of possible locations
  - e.g. 4-way set-associative

- Fully-associative
  - Cache line can be placed anywhere
  - Costly in H/W to check entire cache for hit

Terminology

N-bit address

- TAG field
  - Indicates the address tag for comparison

- LINE field
  - Indicates the proper cache entry

- WORD field
  - Indicates which word referenced within a line

- BYTE field
  - Indicates a single byte, but not normally used
Direct-Mapped Cache

Terminology

- **Hit** – memory location found in cache
- **Miss** – memory location not in cache
  - Compulsory
    - Cache is empty at the start of a program
  - Conflict
    - Another valid cache line is currently stored in the location
  - Capacity
    - Cache isn’t large enough to hold the entire working set of a program
Homework Assignment

- Cache Homework due Friday (9/12)
  - Posted on web
  - [http://eecs.vanderbilt.edu/courses/cs231/CS_231_schedule.html](http://eecs.vanderbilt.edu/courses/cs231/CS_231_schedule.html)

A Small Error?

What if…

- You were to receive $17 million by EFT
- The “17” is encoded $10001_2$ and transmitted
- The received data is $00001_2$

Only one bit was changed, but the impact is quite significant!!!
Parity Bit – Simple Error Detection

For our $17 million scenario

• Transmit $10001_2$ and parity bit
  – Chosen to make number of 1 bits even (or odd)
  – Is there a difference between even and odd parity?
  – For even parity codeword: $100010_2$

• Received codeword is $000010_2$
  – Incorrect parity means error in received data

• Parity detects any single-bit error
  – Actually detects any odd number of errors

Terminology

• Hamming Distance
  – Number of bit positions in which two codewords differ

• Minimum Distance
  – Minimum Hamming distance between all distinct pairs of codewords

A code with minimum distance $d_{\text{min}}$ can detect all error patterns of weight less than or equal to $(d_{\text{min}} - 1)$
Parity Check

- For even or odd parity
  - $d_{\text{min}}$ equals 2
  - Detects error patterns $\leq (d_{\text{min}} - 1)$
  - Generates an error flag

- Uses of parity check
  - Memory
  - Disk storage

Where Do Errors Come From?

- Voltage spikes from power supply
- Coupling with nearby signals
- Noise on transmission line
- Radiation
- Manufacturing defect
- Etc.
One Step Further

• Scenario: transmit a yes/no answer
  – Codeword for “yes” = 1111_2
  – Codeword for “no” = 0000_2

• Received codeword is 0010_2
  – Is this valid?
  – What’s the likely answer?

• Closest to 0000_2 codeword
  – Hamming distance equals one
  – Correct the codeword

One Step Further

• Received codeword is 1110_2
  – Is this valid?
  – What’s the likely answer?

• Received codeword is 1010_2
  – Is this valid?
  – What’s the likely answer?
Terminology

- **Hamming Distance**
  - Number of bit positions in which two codewords differ

- **Minimum Distance**
  - Minimum Hamming distance between all distinct pairs of codewords

A code with minimum distance $d_{\text{min}}$ can correct all error patterns of weight less than or equal to $[(d_{\text{min}} - 1)/2]$.

Error Correction

- Scenario: transmit a yes/no answer
  - Codeword for “yes” = $1111_2$
  - Codeword for “no” = $0000_2$

- Minimum distance equals 4
  - Error correction less than or equal to $[(d_{\text{min}} - 1)/2]$

- Therefore our code corrects all errors $\leq 1.5$

- Do we need the 4th bit for the same capability?
Hamming Codes

- First major class of binary codes designed for error correction
- Originally used in error control for long-distance telephony
- Encodes parity for groups of bits within data

Error Correcting for 4-Bit Words

Encode $1101_2$ with even parity
Error Correcting for 4-Bit Words

What about a single error?

Hamming Algorithm

- In a Hamming code
  - $r$ parity bits added to $m$-bit word
  - Forms codeword with length $(m + r)$ bits

- Bit numbering
  - Starts at 1 with leftmost (high-order) bit
  - All powers of 2 are parity bits
  - Remaining bits are for data
Requirements for Single-Bit Error Correction

![Table: Number of check bits for a code that can correct a single error.]

Example

Given an 8-bit data word to encode

- Codeword has 12 bits (8 data, 4 parity)
  - Bit 1 checks: 1, 3, 5, 7, 9, 11
  - Bit 2 checks: 2, 3, 6, 7, 10, 11
  - Bit 4 checks: 4, 5, 6, 7, 12
  - Bit 8 checks: 8, 9, 10, 11, 12
# Parity Bit Assignment

<table>
<thead>
<tr>
<th>Bit no</th>
<th>Bit no in binary</th>
<th>Encoded by</th>
<th>Encodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8's</td>
<td>4's 2's 1's</td>
<td>3, 5, 7, 9 and 11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>3, 5, 7, 9 and 11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>3, 6, 7, 10 and 11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 1</td>
<td>1 1 &amp; 2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>5, 6, 7, and 12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1</td>
<td>1 1 &amp; 4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 1</td>
<td>0 2 &amp; 4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1</td>
<td>1 1, 2 and 4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>9, 10, 11 and 12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>8 and 1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 0 1 0</td>
<td>8 and 2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 0 1 1</td>
<td>8, 1 and 2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1 1 0 0</td>
<td>8 and 4</td>
<td></td>
</tr>
</tbody>
</table>