Announcements

- **Reading**
  - Today: Chapter 3 (3.3-3.4)
    - Skip details of code
  - Thursday: Chapter 5 (5.1-5.2)

- **Program #1 Due at 10 PM not 10AM**

- **TA Office Hours**
  - Th 1-3
  - F 4-6
  - phone x5-2776
Error Codes (cont.)

- **Error Recovery**
  - Given $m$ bits of data and $r$ bits of error code
  - Want to correct any one bit error
  - There are $n$ words one bit from each valid message
    - so need $n+1$ words for each valid message
    - thus $(n + 1) \cdot 2^m \leq 2^n$
    - but $n = m + r$ so $(m + r + 1) \leq 2^r$

- **Hamming Code**
  - recovers from any one bit error
  - number bits from left (starting at 1)
    - power of two bits are parity
    - rest contain data
  - bit is checked by all parity bits in its sum of power expansion
    - bit 11 is used to compute parity bits 1, 2, and 8
### Hamming Code Example

<table>
<thead>
<tr>
<th>Char</th>
<th>ASCII</th>
<th>Hamming</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1001 000</td>
<td>0011 0010 000</td>
</tr>
<tr>
<td>a</td>
<td>1100 001</td>
<td>1011 1001 001</td>
</tr>
<tr>
<td>m</td>
<td>1101 101</td>
<td>1110 1010 101</td>
</tr>
<tr>
<td>l</td>
<td>1101 001</td>
<td>0110 1011 001</td>
</tr>
</tbody>
</table>

#### Burst Errors
- can send hamming codes by column rather than row
- if use $k$ rows, then can detect any burst error up to $k$ bits
  - uses $kr$ bits to check a block $km$ bits long
### Computing a Hamming Code

<table>
<thead>
<tr>
<th>Bit #s</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity/Data</td>
<td>P</td>
<td>P</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Data To Snd</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 2</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 4</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Message</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Checking & Correcting a Hamming Code

<table>
<thead>
<tr>
<th>Bit #s</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity/Data</td>
<td>P</td>
<td>P</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Data Sent</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Data Recv</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parity Bit 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parity Bit 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR Paritys</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Corrected Msg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Binary # when XOR the parity is the bit position with the error (e.g. 0011 = bit 3 is wrong)
Error Detection

- **Less bits are required**
  - if errors are infrequent, then this works better
  - assumes that re-transmission is possible

- **Cyclic Redundancy Codes (CRC)**
  - Use a generator function $G(x)$ of degree $r$
    - $r+1$ bits long
    - $x^5 + x^2 + 1$ is degree 5 and represented as 100101
  - let $M'$ be the message with $r$ 0’s on the end of it
  - divide $M'$ into $G(x)$ and compute remainder
    - use this as the $r$ bit CRC code
  - a code with $r$ bits will detect all burst errors less than $r$ bits
CRC’s

- several G’s are standardized
  - CRC-12 = \( x^{12} + x^{11} + x^3 + x^2 + x + 1 \)
  - CRC-16 = \( x^{16} + x^{15} + x^2 + 1 \)
  - CRC-CCITT = \( x^{16} + x^{12} + x^5 + 1 \)
- 16 bit CRC will catch
  - all single and double bit errors
  - all errors with an odd number of bits
  - all burst errors of length less than 16
CRC Example

Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after appending 4 zero bits: 1 1 0 1 0 1 1 0 0 0 0

Division is done using XOR

Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0
PPP Protocol

- **Link Protocol for Serial Lines**
  - Supports multiple network protocols: IP, IPX, CLNP, …
  - designed for dialup or leased lines
- **Link Establishment (via LCP - Link Control Protocol)**
  - Negotiate Options
    - configure-request: list of proposed options and values
    - configure-{ack/nack}: will (won’t) use the requested option
  - Allows for authentication

```
1 1 1 1 1 or 2 variable 2 or 4 1
flag 01111110 Address 11111111 control 00001110 protocol payload checksum flag 01111110
```

From: *Computer Networks*, 3rd Ed. by Andrew S. Tanenbaum, (c)1996 Prentice Hall.
PPP Cont.

- **NCP protocol**
  - per network level protocol
  - used to establish network attributes (e.g. addresses)
  - high bit of protocol # is a one

- **Notes on Link Format**
  - character stuff flag byte in data
    - Escape Character is 0x7d (0111 1101)
    - Escape Character and Frame Marker sent at
      - `<Esc-Char><data XOR 0x20>`
      - option to skip address and control fields (since constant)

- **IP**
  - Protocol byte (0x21) or 0x8021 for IP NCP
ATM Datalink Protocol

- **Header**
  - use CRC over the 32 bits of the header

- **How to find cell boundary?**
  - use shift register to check for valid checksum
    - 1/256 chance of a random match
  - use HUNT mode to increase chances
    - after a good cell, skip to the next cell boundary
    - must receive $\delta$ cells with checksum matches

- **Detecting loss of synchronization**
  - one bad cell is probably an error
  - many bad cells is likely a slip (loss of sync)
  - if $\alpha$ bad cells are seen in a row, switch to hunt mode