Sections Week 3

Mohan Kukreja, Anirudh Kumar
Administrivia

- Project-1 is due Today at 11:00 PM
- Homework 2 is due on 30th January 11:00PM
Internet Checksum

- Sum is defined in 1s complement arithmetic (must add back carries)
  - And it’s the negative sum
- “The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words ...” – RFC 791
- In other words, it’s the value that when added to the header, the result is 0xffff
Example Problem 1

Message: 0x466F726F757A616E
# Solution 1

1. **First sum normally**

2. **Add the back carry**

3. **Negate**

\[
\begin{array}{c}
\quad 
\end{array}
\]

<table>
<thead>
<tr>
<th>466F</th>
<th>726F</th>
<th>757A</th>
<th>616E</th>
<th>8FC6</th>
<th>1</th>
<th>~8FC7</th>
<th>7038</th>
</tr>
</thead>
<tbody>
<tr>
<td>18FC6</td>
<td>8FC7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1) First sum normally

2) Add the back carry

3) Negate
Example Problem 2

Message: 0x466F726F757A616E7038
Solution 2

1) First sum normally

2) Add the back carry

3) Negate
0x0000
CRC

- Uses a generator polynomial and polynomial division to calculate an error-detecting code.
- For a polynomial of degree $n$, it creates a check of $n$ bits.
Example Problem 1

Message: 0b10100110
Polynomial: x + 1
Solution 1

\[
\begin{array}{c}
1100010 \\
11 \overline{0100110} \\
\hline
011 \\
11 \\
00 \\
00 \\
00 \\
00 \\
01 \\
00 \\
11 \\
00 \\
\end{array}
\]

← Note: use XOR instead of minus.

← The actual remainder is 0, and thus the CRC remainder is 0.
Example Problem 2

Message: 0b11100101
Polynomial: $x^3 + x^2$
Solution 1

The actual remainder is 1, we add n bits then re-zero out to get CRC, done above.
Interesting Things to Note

- $x + 1$ as a generator polynomial results in a parity bit.
- Has the nice property of being easy to implement in hardware.
- Doesn’t guard against intentional changing of data.

\[ \text{CRC}(x \oplus y) = \text{CRC}(x) \oplus \text{CRC}(y) \]
Hamming Distance & Hamming Code

- Review: Distance is the number of bit flips needed to change D1 to D2
- Hamming distance of a coding is the minimum error distance between any pair of codewords (bit-strings) that cannot be detected

- Error detection:
  - For a coding of distance $d+1$, up to $d$ errors will always be detected
- Error correction:
  - For a coding of distance $2d+1$, up to $d$ errors can always be corrected by mapping to the closest valid codeword
Why Error Correction is Hard

- If we had reliable check bits we could use them to narrow down the position of the error
  - Then correction would be easy
- But error could be in the check bits as well as the data bits!
  - Data might even be correct
Hamming Code

- Gives a method for constructing a code with a distance of 3
  - Uses $n = 2^k - k - 1$, e.g., $n=4$, $k=3$
  - Put check bits in positions $p$ that are powers of 2, starting with position 1
  - Check bit in position $p$ is parity of positions whose $p$-th LSBit is same as $p$’s
- Plus an easy way to correct [soon]
Hamming Code

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
Hamming Code

- Example: data=0101, 3 check bits
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

\[
\begin{array}{cccccccc}
\text{0} & \text{1} & \text{0} & \text{0} & \text{1} & \text{0} & \text{1} & \rightarrow \\
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[p_1 = 0 + 1 + 1 = 0, \quad p_2 = 0 + 0 + 1 = 1, \quad p_4 = 1 + 0 + 1 = 0\]
Hamming Code

- To decode:
  - Recompute check bits (with parity sum including the check bit)
  - Arrange as a binary number
  - Value (syndrome) tells error position
    - Value of zero means no error
    - Otherwise, flip bit to correct
Hamming Code

• Example, continued

\[ \begin{array}{cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \rightarrow & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array} \]

\[ \begin{array}{c}
 p_1 = \\
p_2 = \\
p_4 = \\
\text{Syndrome} = \\
\text{Data} = 
\end{array} \]
Hamming Code

- Example, continued

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 1 & 0 & 0 & 1 & 0 & 1
\end{array} \]

\[ p_1 = 0+0+1+1 = 0, \quad p_2 = 1+0+0+1 = 0, \]

\[ p_4 = 0+1+0+1 = 0 \]

Syndrome = 000, no error
Data = 0 1 0 1
Hamming Code

Example, continued

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\begin{align*}
p_1 &= \\
p_2 &= \\
p_4 &= \\
\text{Syndrome} &= \\
\text{Data} &=
\end{align*}
Hamming Code

• Example, continued

\[
\begin{array}{cccccc}
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]

\[p_1 = 0+0+1+1 = 0, \quad p_2 = 1+0+1+1 = 1,\]
\[p_4 = 0+1+1+1 = 1\]

Syndrome = 1 1 0, flip position 6
Data = 0 1 0 1 (correct after flip!)
Hamming Code

- Example: bad message 0100111
  - 7 bit code, check bit positions 1, 2, 4
  - Check 1 covers positions 1, 3, 5, 7
  - Check 2 covers positions 2, 3, 6, 7
  - Check 4 covers positions 4, 5, 6, 7

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[p_1 = 0+0+1+1 = 0, \quad p_2 = 1+0+1+1 = 1, \quad p_4 = 0+1+1+1 = 1\]