Hamming Codes in TOY

- Hamming codes
- TOY simulator
- bugs to avoid
Goals

- TOY: write two small machine-language programs.
- Hamming codes: learn about a widely used error-correcting code.
Hamming Codes in TOY

- Hamming codes
- TOY simulator
- Bugs to avoid
Noiseless communication channel
Noisy communication channel

1101...

bit flipped

1001...

noisy communication channel

?
Error-correcting codes

1101...

1101100...

noisy communication channel

1001100...

1101...

append redundant information

bit flipped

redundant information enables correction of single-bit error
Error-correcting codes

**Message bits:** \( m_1, m_2, m_3, m_4. \)

**Goal.** Send and receive 4 message bits at a time.

**Noiseless channel.** What you send is what you receive.

**Easy.** Send \( m_1, m_2, m_3, m_4. \)

**Noisy channel.** One of the 4 bits might get flipped during transmission.

**Attempt 1.** Send \( m_1, m_2, m_3, m_4. \)

**Attempt 2.** Send \( m_1, m_1, m_2, m_2, m_3, m_3, m_4, m_4. \)

**Attempt 3.** Send \( m_1, m_1, m_1, m_2, m_2, m_3, m_3, m_3, m_4, m_4, m_4. \)

This assignment. **7–4 Hamming code:** correct 1-bit errors, but using only 7 bits instead of 12.
Parity bits

Message bits: $m_1, m_2, m_3, m_4$.
Parity bits: $p_1, p_2, p_3$.

Parity bits. Uniquely chosen so that the sum of bits in each circle is even.

$$1 + 1 + 1 + p_1 = \text{even}$$
$$1 + 1 + 0 + p_2 = \text{even}$$
$$1 + 1 + 0 + p_3 = \text{even}$$
Hamming encoding quiz

Which 7 bits are sent for the message 1100?

A. 1100000
B. 1100010
C. 1100011
D. 1100111
Useful trick: the xor function

**Hint.** Can use the xor function to compute parity bits.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th><strong>x ^ y</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
p_1 = m_1 \land m_2 \land m_4
\]
\[
p_2 = m_1 \land m_3 \land m_4
\]
\[
p_3 = m_2 \land m_3 \land m_4
\]

**Ex 1.** \( p_1 = 1 \land 1 \land 1 = 1 \).  
**Ex 2.** \( p_2 = 1 \land 0 \land 1 = 0 \).  
**Ex 3.** \( p_3 = 1 \land 0 \land 1 = 0 \).
Message bit $m_2$ flipped

1 + 1 + 0 + 1 = odd
(top fails)

1 + 1 + 0 + 0 = even
(left passes)

1 + 0 + 0 + 0 = odd
(right fails)
Message bit $m_2$ flipped

- 1101100 transmitted
- Suppose 1001100 received
Message bit $m_4$ flipped

1101100 transmitted

1100100 received

suppose 1101100 transmitted

1100100 received

1 + 1 + 0 + 1 = odd (top fails)

1 + 0 + 0 + 0 = odd (left fails)

1 + 0 + 0 + 0 = odd (right fails)
Message bit $m_4$ flipped

1101100 transmitted

suppose 1001100 received

top fails
left fails
right fails
Parity bit $p_3$ flipped

$1 + 1 + 1 + 1 = \text{even}$ (top passes)

$1 + 1 + 0 + 0 = \text{even}$ (left passes)

$1 + 1 + 0 + 1 = \text{odd}$ (right fails)

110110 transmitted

suppose 1101101 received
Parity bit p₃ flipped

1101100 transmitted

suppose 1101110 received

right fails
Error correction rule

Compute parity bits $p_1$, $p_2$, and $p_3$ and compare against received bits.

- If at most 1 parity check fails, all message bits are correct.
- If all 3 parity checks fail, then message bit $m_4$ was flipped.
- If only checks $p_1$ and $p_2$ fail, then message bit $m_1$ was flipped.
- If only checks $p_1$ and $p_3$ fail, then message bit $m_2$ was flipped.
- If only checks $p_2$ and $p_3$ fail, then message bit $m_3$ was flipped.

Caveat. 7–4 Hamming code are not designed to detect (or correct) multiple flipped bits.
Hamming decoding quiz

You receive the bits 1000101. Which were the original 4 message bits?

A. 0000
B. 1001
C. 1010
D. 1100
Hamming Codes in TOY

- Hamming codes
- TOY simulator
- bugs to avoid
TOY file format

```plaintext
% more echo.toy
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
* Name:    Kevin Wayne
* NetID:   wayne
* Precept: P00
*
* Description: Reads integers from standard input until 0000;
*               prints each integer to standard output.
* "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

10: 81FF read R[1]
11: C114 if (R[1] == 0) goto 14
12: 91FF write R[1]
13: C010 goto 10
14: 0000 halt

while (!StdIn.isEmpty()) {
    a = StdIn.readInt();
    StdOut.println(a);
}
```

TOY instruction (in hex)

memory address (in hex) followed by colon

Java-style comments (optional)
Edit file. Use any text editor (such as DrJava).

Not-so-useful feature in DrJava.
- DrJava auto-indents lines.
- Preferences → Miscellaneous → Indent Level = 0.
  [switch back to 4 after this assignment]

Execute. Execute TOY program from command line.
- TOY.java must be in same directory as .toy files.
- java-introcs TOY encode.toy < encode3.txt
- java-introcs TOY decode.toy < decode5.txt

% more encode3.txt
0001 0001 0000 0001
0001 0001 0001 0000
0001 0001 0001 0001
FFFF

for simplicity, each bit stored as 16-bit TOY word
4 bits to encode

end of file convention

% more decode3.txt
0001 0000 0000 0001 0001 0000 0000
0000 0001 0001 0000 0000 0000 0000
0001 0001 0001 0001 0001 0001 0000
FFFF

7 bits to encode
Visual X-TOY simulator

Edit mode. Write your TOY program.
Debug mode. Execute your TOY program.
Simulation mode. For historical context.

Useful features.
- Syntax highlighting.
- Automatically generates TOY pseudo-code.
- Tools → Check Syntax.
- Mode → Load File to Stdin.

```java
/**
 * Name: Kevin Wayne
 * NetID: wayne
 * Precept: P00
 *
 * Description: Reads integers from standard input until 0000;
 *              prints integers to standard output.
 */

10: 81FF read R[1]
11: C114 if (R[1] == 0) goto 14
12: 91FF write R[1]
13: C010 goto 10
14: 0000 halt

written by Brian Tsang '04
```
Hamming Codes in TOY

- Hamming codes
- TOY simulator
- bugs to avoid
Tips to avoid common bugs

- Start your TOY code at line 10.
- Check that each line of TOY code has format XX:YYYY.
- Remember that “everything” is in hex (line 1A follows 19).
- Make sure TOY code and pseudo-code match.
- Document the purpose of each register (and don’t reuse).
- Use care when inserting a line of code:
  - might need to update jump statement if line to goto changes.
- Repeatedly read 4- or 7-bits from standard input until FFFF.