COLLEGE of CHARLESTON

MATH MEET 2016

Message from Pluto

How do you get a computer to send a message back from Pluto? To begin with, computers speak binary. Instead of counting to ten, computers work in base two. Everything a computer knows is in terms of *bits*. A bit is a 0 or a 1. But bits are enough to represent every possible number, in the same way that it's possible to represent any number using just ten decimal numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and place value. To avoid confusion, I'll write a b at the end of binary numbers but not decimal, so 10 means ten, but 10b means two.

The binary number 1101b means

$$\begin{aligned} 1101\text{b} &= 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 \\ &= 1\times 8 + 1\times 4 + 0\times 2 + 1\times 1 \\ &= 13 = \text{ thirteen} \end{aligned}$$

What is the value in decimal notation of the binary number 11010011b?

Computers also like to work with fixed bit-width numbers. A *byte* is always eight bits, for example, or you could say that a byte is eight bits wide. A 32-bit computer works best with numbers that are 32 bits wide, and a 64-bit computer works best with numbers that are 64 bits wide. If you need to express a number with a fixed bit-width, you have a limited range. For example, it is possible to express each number from 0 through 255 as one byte, but 300 is too big:

$$\begin{aligned} 0 &= \underbrace{00000000}_{\text{eight bits}}\text{b} \\ 1 &= 00000001\text{b} \\ 2 &= 00000010\text{b} \\ 3 &= 00000011\text{b} \\ 4 &= 00000100\text{b} \\ &\vdots \\ 255 &= \underbrace{11111111}_{\text{eight bits}}\text{b} \\ 300 &= \underbrace{100101100}_{\text{nine bits}}\text{b} \end{aligned}$$

When working with fixed bit-width numbers, we write small numbers with extra leading 0s.

Express 37 as a binary number eight bits wide.

Let's suppose we want to send English text as numbers. The obvious encoding is to use numbers to represent letters, from 1 (meaning *A*) to 26 (meaning *Z*), plus a few extra numbers for things like spaces and punctuation.

What numbers (in decimal notation) represent the letters in the word *MATH*?

Let's use fixed bit-width numbers to represent letters. Sending data all the way from Pluto is expensive, so let's try to use as few bits as possible. We won't distinguish uppercase and lowercase letters.

What is the narrowest bit-width that will suffice to represent the 26 letters, including the use of 0 to represent a space?

An extra complication is the bit order. We normally write digital numbers in *big-endian* notation. That is, the units bit is on the far right and the digit on the far left is multiplied by the largest power of the base:

$$\text{four hundred twenty five } = 425 = 4 \times 100 + 2 \times 10 + 5 \times 1$$

But a lot of computers work in *little-endian* notation. That is, the units bit is written at the far left. I'll use angle brackets $\langle \ldots \rangle$ around numbers that are in little-endian notation. The bits are interpreted in the reverse order from numbers in ordinary big-endian notation:

$$\text{thirteen } = 1101\text{b} = \langle 1011 \rangle = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$$
$$\text{fourteen } = 1110\text{b} = \langle 0111 \rangle = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$$

By the way, the names big- and little-endian refer to a joke in *Gulliver's Travels* in which a war is fought over which end of a boiled egg to crack open first, the big end or the little end.

Using the representation that a space is 0, *A* is 1, *B* is 2, ..., *Z* is 26, represent the word *MATH* in little-endian binary, using the bit width you just solved for. Write the bits for *M* left to right, immediately followed by the bits for *A*, etc.

Sending bits over long distances is error prone. An asteroid might get in the way—or maybe even a Plutonian! We could send the entire message twice. The two copies will disagree on a bit when one transmission was disrupted but the other wasn't. But which one is correct? Hmm.

We could send the entire message three times. That way, if a bit in one copy is disrupted and the others aren't, the other two copies will agree, so the majority will be the correct bit. However, it turns

out there are clever ways to encode binary numbers that make it possible to detect and even correct errors without transmitting the entire message three times.

New Horizons uses a sophisticated error correcting code. We're going to do something simpler but still effective. Here's the first step: One way to check for an error is to use *parity*. The parity of a string of bits is 0 if there are an even number of ones, and 1 if there are an odd number of ones. That is, you add up the bits modulo 2:

$$P(\langle 0000\rangle) = 0$$
$$P(\langle 1000\rangle) = 1$$
$$P(\langle 0100\rangle) = 1$$
$$P(\langle 1100\rangle) = 0$$

If we transmit a parity bit along with the data bits, the receiver can detect when one bit is wrong somewhere. For example, we could transmit four data bits $\langle b_1 b_2 b_3 b_4\rangle$ as the message $\langle b_1 b_2 b_3 b_4 p\rangle$ where $p = P(\langle b_1 b_2 b_3 b_4\rangle)$. If we receive $\langle 01001\rangle$, the parity bit is correct, so the all bits were probably correctly transmitted. If we receive $\langle 00100\rangle$, we know that transmission of one of the five bits was disrupted, but there's no way to tell which one. It might even be the parity bit itself.

Assuming these messages are encoded with a final parity bit, which ones indicate a transmission error and which ones don't? Check a box for each.

Message	Error	No error
$\langle 10110\rangle$	☐	☐
$\langle 01001\rangle$	☐	☐
$\langle 0010101\rangle$	☐	☐
$\langle 101101\rangle$	☐	☐

If we include several parity bits, each computed from only some of the data bits, it's possible to construct the following *Hamming code*. Given data bits $\langle b_1 b_2 b_3 b_4\rangle$, transmit the message

$$\langle p_0 p_1 b_1 p_2 b_2 b_3 b_4\rangle$$

where

$$p_0 = P(\langle b_1 b_2 b_4\rangle)$$
$$p_1 = P(\langle b_1 b_3 b_4\rangle)$$
$$p_2 = P(\langle b_2 b_3 b_4\rangle)$$

The parity and data bits are ordered as they are in the message because of the following remarkable property of this code. Compute

$$q_0 = P(\langle p_0 b_1 b_2 b_4\rangle)$$
$$q_1 = P(\langle p_1 b_1 b_3 b_4\rangle)$$
$$q_2 = P(\langle p_2 b_2 b_3 b_4\rangle)$$

The binary number $n = \langle q_0 q_1 q_2\rangle$ comes out $\langle 000\rangle$ if there's no transmission error. But if one bit is wrong, n is the index from 1 to 7 of exactly which bit is wrong. For example, suppose we receive the

message ⟨1110010⟩. Then

$$\begin{aligned} &\langle 1110010\rangle \\ q_0 &= P(\langle 1\_1\_0\_0\rangle) = 0 \\ q_1 &= P(\langle \_11\_\_10\rangle) = 1 \\ q_2 &= P(\langle \_\_\_0010\rangle) = 1 \\ n = \langle q_0 q_1 q_1\rangle &= \langle 011\rangle = 110\text{b} = 6 \end{aligned}$$

That means the sixth bit is wrong, and the message should have been ⟨1110000⟩ instead, which means the data bits should be ⟨1000⟩.

What are the four data bits for the incorrect message ⟨1001000⟩?

Now, here's how to decode the message on the tee shirt: The English text for the message from Pluto was translated from letters and spaces to numbers from 0 to 26, which were in turn encoded as fixed-width binary numbers in little-endian bit order. The minimum bit width you determined earlier was used. Those bits were strung together and broken into groups of 4 bits, padded with extra 0s at the very end. Each group of 4 data bits was expanded into a message of 7 bits using the Hamming code. To reverse the process, read columns of seven bits, from top left (near Earth) to bottom right (near Pluto). Within each column, the seven bits are to be read from top to bottom as $\langle p_0 p_1 b_1 p_2 b_2 b_3 b_4\rangle$. An all-white dot is a 1, and a black dot with a white outline is a 0. Some bits have been lost, blocked by asteroids or Saturn. Some have been obscured by the glint of star light. A few were caught by a passing Plutonian. Use the error correcting code and a little extra cleverness to translate the message from Pluto!

What is the message from Pluto?