Challenge Lesson: Ozobot Bit—Decimal to Binary Converter

Created by

Richard Born

Associate Professor Emeritus

Northern Illinois University

richb@rborn.org

Topics

Computer Science, Algorithm Design, Functions, Procedures, Decimal-to-Binary conversion, integer division, remainder, Random numbers

Ages

Grades 7-12

Duration

This should probably be given as a homework assignment for students to do with their own Ozobot Bit at home. It could take several hours to complete, depending upon knowledge of Mode 4 OzoBlocks.The teacher could make the due date a week after assigning the project.

Challenge Lesson:
Ozobot Bit—Decimal to Binary Converter

By Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn@niu.edu

Students of computer science need to become adept at algorithm design and the use of user-defined functions. They also need to understand how to convert decimal numbers to their equivalent in the binary number system—the ultimate language of digital computers. This challenge lesson will provide the student with lots of practice in these skills.

There are a variety of techniques for converting a decimal number to binary. One of the most common is successive divisions by 2, each time noting the quotient and remainder. Figure 1 illustrates this procedure starting with the decimal number 115.

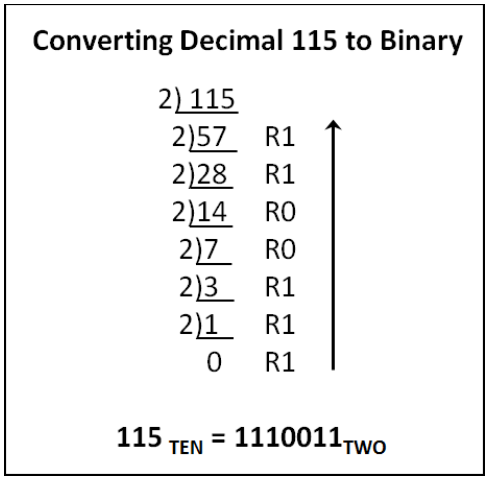


Figure 1

The procedure begins by writing the decimal number in an upside-down division symbol, and writing 2 as the divisor to the left of the symbol. When 115 is divided by 2 the quotient is 57 and the remainder is 1, indicated by R1 to the right of 57. 57 is then divided by 2, giving a quotient of 28 and a remainder of 1. This process continues until the quotient is zero, where we see that 1 divided by 2 gives a quotient of zero, and a remainder of 1. As shown at the bottom of Figure 1, the remainders are then written next to each other, starting with the remainder at the bottom of the list of remainders and ending with the remainder at the top of the list.

Figure 2 shows that 1110011_{TWO} can be obtained by adding the base 10 numbers 64, 32, 16, 2, and 1, for a result of 115_{TEN}.

1	1	1	0	0	1	1
64	32	16	8	4	2	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

Figure 2

There are two OzoBlockly blocks that are particularly helpful when considering how to implement the "division by 2" algorithm. They are shown in Figure 3.

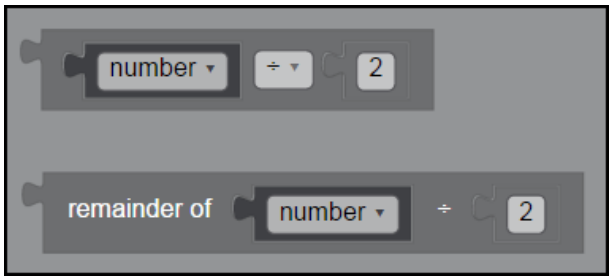


Figure 3

You need to remember that OzoBlockly division is integer division, and as such the division process results in the quotient with the remainder missing. Thus 115/2 = 57 in OzoBlockly arithmetic, not 57.5. If you want the remainder when dividing a number by 2, OzoBlockly provides the "remainder of" block, as shown at the bottom of Figure 3. The remainder when 115 is divided by 2 will be 1.

The Map for this Challenge Lesson

The real challenge when designing your OzoBlockly program is to come up with a way by which Ozobot Bit can display both a decimal number and the corresponding binary number in some way while moving about the map. Figure 4 shows a small copy of the map that we will use to discuss how your program should cause Ozobot Bit to move around on the map. A larger copy, suitable for printing and using with your Ozobot Bit and your OzoBlockly program, can be found on the last page of this document.

You will immediately notice that the map consists of two main parts. In the red portion Ozobot Bit will display a red LED. Ozobot Bit will display a random three digit decimal number *xyz* between 0 and 127 by traversing the red loop three times, stopping at one of the numbered

locations each time to show the user the digits *x*, *y*, and *z*, in that order. While stopped on one of the numbers on the red loop, he will show a 2-second white LED. Then, after displaying all three decimal digits, he will enter the black line on the far left. There will be a delay of 40 seconds, giving the user a chance to determine (on a piece of scratch paper, if necessary) the binary number corresponding to decimal *xyz*. Ozobot Bit will then proceed to the blue portion of the map and display the binary result. He will do this by stopping at either 0 or 1 at each of the powers of two shown at the bottom of the map, as appropriate. He will show a 3-second white LED if the bit is 0 and a 3-second green LED if the bit is 1.

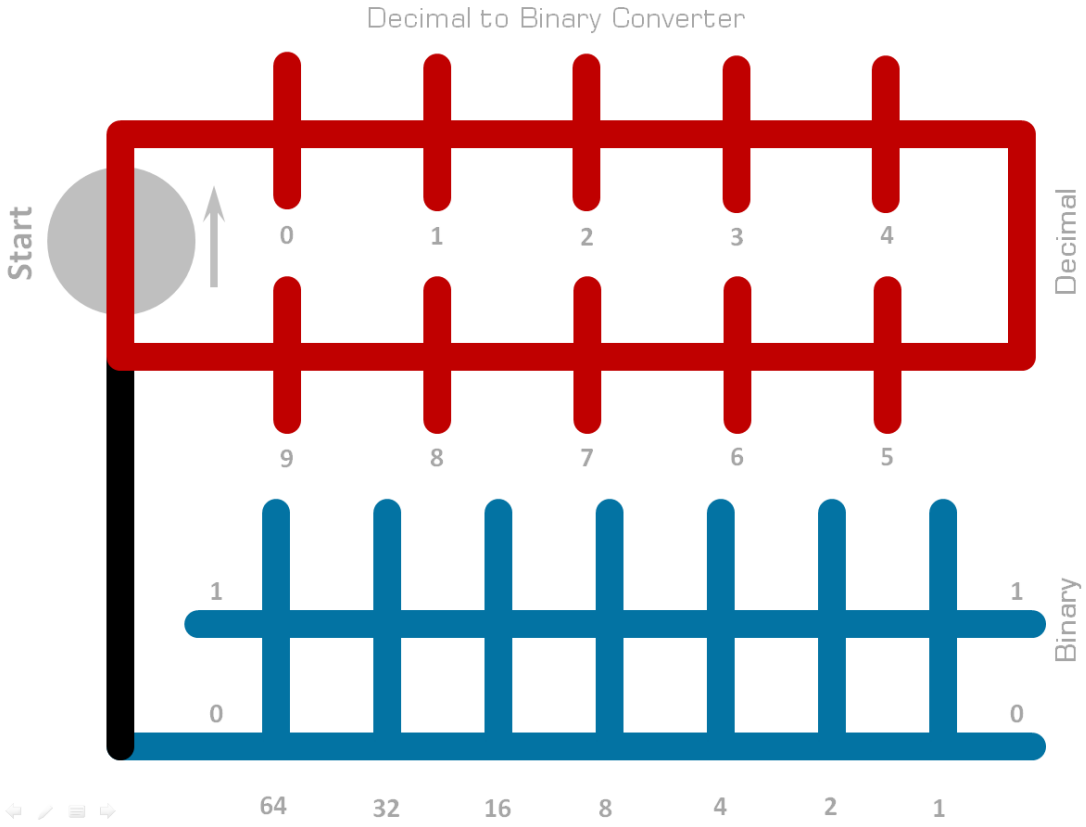


Figure 4

Some Suggestions

This challenge is complex enough that the use of functions (also known as procedures) will help to break the coding into more manageable parts. In addition, it will help keep the number of OzoBlocks required and Ozobot memory for storing the program smaller.

The main program can perform tasks including:

- Set the initial LED color to red.
- Get a random integer, *threeDigNum*, from 0 to 127.

- Use integer division and the "remainder of" block to compute the *hundreds*, *tens*, and *ones* places of *threeDigNum.*
- Call a user-defined function, maybe named *displayBase10Digit*, three times, one time each for the *hundreds*, *tens*, and *ones* places. This function will move Ozobot Bit around the red potion of the maze where he will stop once each time at the numbered location corresponding to the decimal digit. Ozobot Bit will show a white LED for 2 seconds while stopped on the appropriate numbered location for the digit. Figure 5 shows what a call to this function might look like.

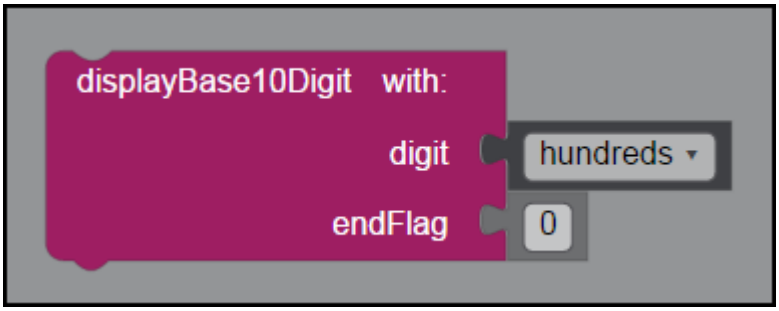


Figure 5

The purpose of the *endFlag* is that after displaying the hundreds and tens digits, Ozobot must turn right (*endFlag* 0) toward the Start position. However, after displaying the ones place, Ozobot must turn left (*endFlag* 1) onto the black line. This flag gives you a way to identify this in your *displayBase10Digit* function.

- Use the "successive division by 2" algorithm to create a function that will convert *threeDigNum* to its binary digits 0 or 1, which you might store in variables named *B64, B32, B16, B8, B4, B2, and B1*. Figure 6 shows what a call to this function might look like.

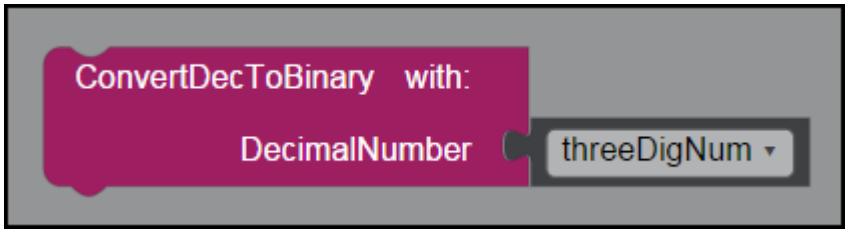


Figure 6

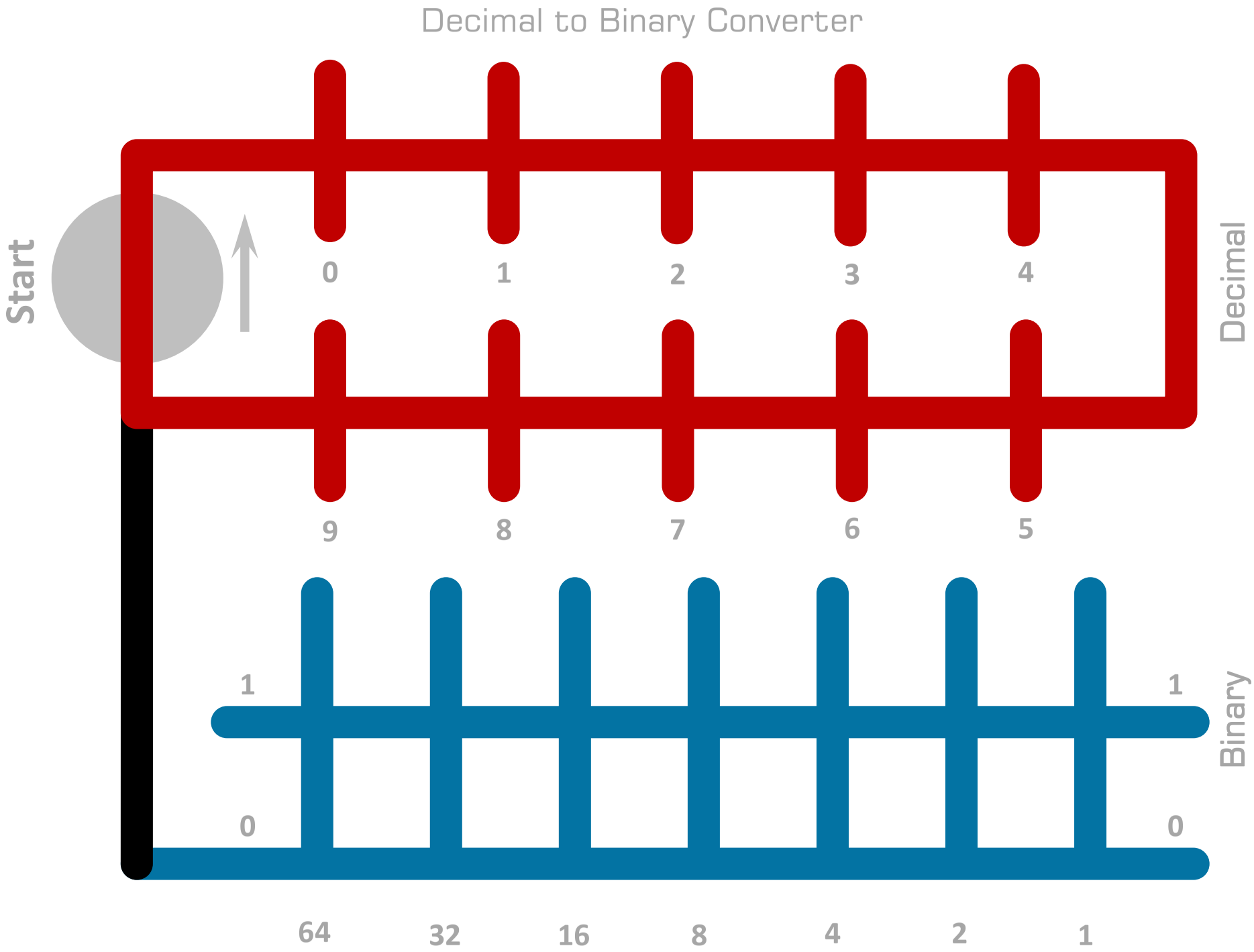
- Call a user-defined function, perhaps named *displayBinaryDigit* that will move Ozobot Bit to the next binary digit location on the blue part of the maze, stop on the 0 line or the 1 line as appropriate, and display a white LED if the bit is a 0, and a green LED if the bit is a 1. A typical call to this function might be similar to that shown in Figure 7.

Figure 7

At this writing, Level 5 [Master] is not yet available. It is supposed to contain some blocks related to the use of arrays. If you program this challenge using arrays or other Level 5 features, then your program may be significantly different than outlined in these suggestions.

Challenge Requirements

1. Use OzoBlockly codes from Mode 4 (Advanced) or higher.
2. Make sure that you calibrate Ozobot Bit on paper before testing and running your program.
3. Make sure that Ozobot Bit's battery is fully charged and the wheels are clean.
4. To start Ozobot Bit's program, you need to double-press the start button.
5. Ozobot Bit should be started on the "Start" circle in the upper left corner of the maze, facing the direction shown by the arrow.
6. Ozobot Bit should behave in the manner described on the previous pages of this document.


Decimal to Binary Converter
Start
0
1
2
3
4
Decimal
9
8
7
6
5
1
1
Binary
0
0
64
32
16
8
4
2
1