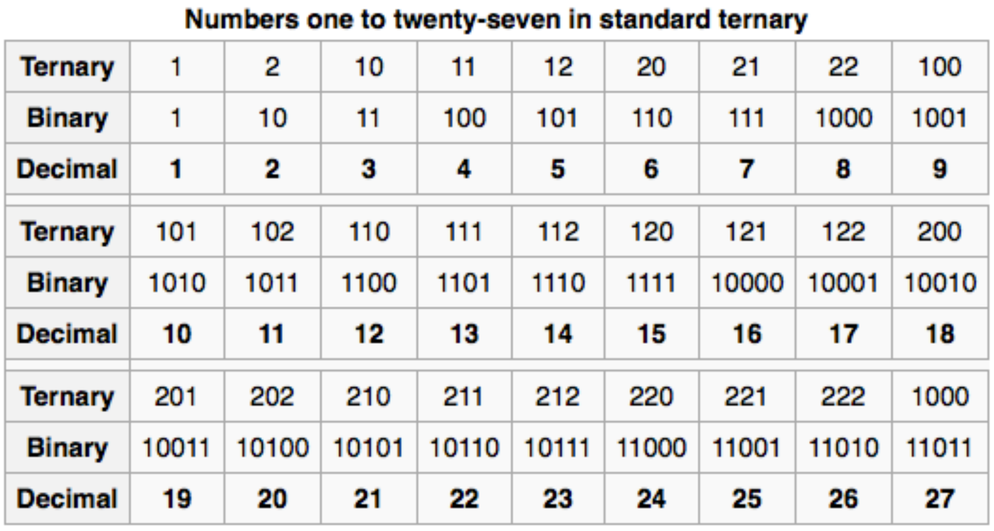
Binary to Ternary Numerals

Introduction

Ternary is the base-3 numeral system. Analogous to a bit, a ternary digit is a **trit** (**tr**inary dig**it**). One trit contains $\log_2 3$ (about 1.58496) bits of information. For reference, octal, $\log_2 8$ is 3. *Ternary* most often refers to a system in which the three digits 0, 1, and 2 are all non-negative numbers.



Numbers one to twenty-seven in standard ternary

Ternary	1	2	10	11	12	20	21	22	100
Binary	1	10	11	100	101	110	111	1000	1001
Decimal	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Ternary	101	102	110	111	112	120	121	122	200
Binary	1010	1011	1100	1101	1110	1111	10000	10001	10010
Decimal	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Ternary	201	202	210	211	212	220	221	222	1000
Binary	10011	10100	10101	10110	10111	11000	11001	11010	11011
Decimal	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**

In certain analog logic, the state of the circuit is often expressed ternary. This is most commonly seen in Transistor–Transistor logic using 7406 open collector logic. (remember your protoboard output circuit). The output is said to either be low (grounded), high, or open (high-Z). In this configuration the output of the circuit is actually not connected to any voltage reference at all. The state is said to be high impedance because it is open and serves as its own reference. Thus, the actual voltage level is sometimes unpredictable.

The **radix economy** of a number in a particular base is the number of digits needed to express it in that base, multiplied by the radix (the number of possible values each digit could have). Various proposals have been made to quantify the relative costs between using different radices in representing numbers, especially in computer systems.

For example,

100 in decimal has three digits, so its radix economy is 10×3 = 30;

Its binary representation has seven digits (1100100_2) so it has a radix economy 2×7 = 14 in base 2;

In base 3 its representation has five digits (10201_3) with a radix economy of 3×5 = 15

It turns out that the most economical base is a value that we know as *e*.

The value of *e* is approximately 2.718, so it follows that we should be able to make the most economical computing with a radix value of 3 rather than 2. However we have many more practical instances of on/off, 0/1 than with three states.

This assignment is intended to give you more practice with a number code conversion that relates binary and ternary. The need for conversion from binary to ternary is somewhat artificial but the exercise gives you an insight into the code conversion process that you will meet later in the course. The completion of the above task shows that you have mastered the ability to create an expression that represents a logic function and to translate this into a working circuit with multiple AND and OR gates.

Prerequisites

This lab assignment contains some elements of design but is only part of a larger problem. It does assume that you know the structure of writing logic expressions. This assignment requires that you understand how to create a logic circuit from a Boolean expression though only inverters, two-input ANDs, and two-input OR gates are available.

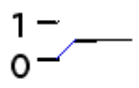
Objectives

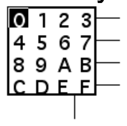
This lab is intended to follow the basic understanding of number systems. As this is an early LogicWorks assignment, considerable skills need to be built and, though the circuitry in this lab is not complex, the intricacy of connecting paths lends itself to the need for great care and attention to detail.

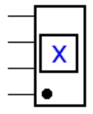
Theory

The table on the following page shows the input and output values. We are only to design the conversions for the eleven decimal values 0 to 10. The equivalent binary values are labeled DCBA from 0000 to 1010. The ternary equivalents XYZ are from 000 to 101.

Our total design would be to display the three ternary values XYZ but we will limit this exercise to the value of Z only. The inputs DCBA can be via LogicWorks switches or a keyboard, the output via a hex display binary switches: hex keyboards: hex displays







Example process: consider output Y. Let us break the value of Y into four parts (left to right, MSB to LSB) Y3Y2Y1Y0.
By inspection, within our limited problem set,
Y3=0 in all cases, similarly Y2 is also 0.
Y1 is a 1 when the inputs DCBA are 0110, 0111, and 1000. In all other values of the inputs Y1 is a 0.
Y0 is a 1 for inputs 0011, 0100, and 0101. In all other values of the inputs Y0 is a 0.

	DCBA	XYZ	X	Y	Z	H
Dec	Bin	Tern	Bin	Bin	Bin	Hex
0	0000	0	0000	0000	0000	0
1	0001	1	0000	0000	0001	1
2	0010	2	0000	0000	0010	2
3	0011	10	0000	0001	0000	0
4	0100	11	0000	0001	0001	1
5	0101	12	0000	0001	0010	2
6	0110	20	0000	0010	0000	0
7	0111	21	0000	0010	0001	1
8	1000	22	0000	0010	0010	2
9	1001	100	0001	0000	0000	0
10	1010	101	0001	0000	0001	1

Your task is to design the logic to create the correct outputs for Z from decimal 0 to10 only.

Procedure

You must complete the following steps:

Z is composed of the four output bits Z3, Z2, Z1, & Z0.
Create the equations for each of these outputs in terms of DCBA.

Create the circuit to implement these terms using only 2-input gates as higher multiple input chips are not available today. *You do not have to reduce or minimize the expressions but you can if you wish*.

Build and exercise the LogicWorks circuit that implements your design.

Check that the results of your implementation are the same as the specifications.

Testing

Since you only have 11 possible inputs (decimal 0 to 10), you can test your circuit exhaustively. All of your results should agree with the specifications. You should also test your circuit by applying the 5 remaining, unused, input combinations and document the results.

Deliverables

Once you've tested and thoroughly debugged you circuit, demonstrate your results to your TA. Then write a report which follows the required format. Submit the report and the circuit file through HusckyCT.

Questions

What do you think you could do to reduce the number of gates in your solution?

Converting to Base-3 with the Remainder Method – does this look familiar?

Let's convert the number 4,517 as an example. The first step is to divide it by three and note the remainder. This gives us a quotient of 1,505 remainder **2**.
Next, divide 1,505 by 3. This gives us a quotient of 501 remainder **2**.
501/3 = 167 remainder **0**.
167/3 = 55 remainder **2**.
55/3 = 18 remainder **1**.
18/3 = 6 remainder **0**.
6/3 = 2 remainder **0**.
2/3 = 0 remainder **2**.

From these remainders, 4,517 (base 10) is equal to 20012022 (base 3).

Answer: yes.
This is the same process to generate a binary number from decimal!