ECE 204 - BINARY NUMBERS AND CODES - INV 11
INTRODUCTION TO BINARY CODES

FALL 2003 **A.P. FELZER**

To do "well" on this investigation you must not only get the right answers but must also do neat, complete and concise writeups that make obvious what each problem is, how you're solving the problem and what your answer is. You also need to include drawings of all circuits as well as appropriate graphs and tables.

From the last three Investigations we know how to convert numbers from decimal to binary and back again to decimal. And we know how to use 2's complement arithmetic to add and subtract positive and negative numbers. The objective of this Investigation is to introduce some other codes including octal, hex and BCD codes for representing numbers and Gray codes for representing physical locations.

1. We begin with some review problems
   a. Convert 87 to 8-bit binary
   b. Convert –87 to 8-bit signed magnitude
   c. Convert –87 to a 10 bit 2's complement number
   d. Calculate the following sum of 2's complement numbers: $A = 10110^2 + 01101^2$
   e. Calculate the following difference of 2's complement numbers: $B = 11101^2 - 00101^2$

2. Binary numbers like the following

   10110101100101011

   are of course great, but keeping track of them is in general tedious and prone to error. One way to reduce this problem is to write our binary numbers in **hexadecimal (hex)** which is base 16. The objective of this and the next problem is to introduce the hex number system with the following sixteen symbols

Hex	Decimal
0–9	0–9
A	10
B	11
C	12
D	13
E	14
F	15

   and place values equal to

   $$\ldots \quad 16^3 \quad 16^2 \quad 16^1 \quad 16^0$$

   Memorize these results and then
   a. Convert $B3_{16}$ to decimal
   b. Convert $2D_{16}$ to decimal
   c. Convert $2A7_{16}$ to decimal

3. What makes hex numbers so nice is that they're easy to convert back and forth to binary as illustrated in the following example

$$5B_{16} = 5\cdot 16^1 + 11\cdot 16^0 = (0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0)\cdot 2^4 + (1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0)\cdot 2^0$$
$$= 0\cdot 2^7 + 1\cdot 2^6 + 0\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$$
$$= 01011011_2$$

And so all we have to do to

(1) Convert hex to binary is replace every hex digit by its four digit binary equivalent
(2) Convert binary to hex is collect the 1's and 0's of the binary number into groups of four and then replace each one by its hex equivalent

Memorize these results. And then use them to

a. Convert $6D2_{16}$ to binary
b. Convert 10110101100101_2 to hex
c. How many digits does it take to represent a byte in hex

4. We call decimal, binary and hex codes **weighted codes** because the values of numbers like the following

$$A = 01011011_2 = 1\cdot 2^6 + 0\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$$

are equal to the sum of each of its digits times its *weight* - its place value. What makes these codes particularly useful - what makes the arithmetic so easy - is that the weights are equal to powers of the base like the following

$$2^0,\ 2^1,\ 2^2,\ 2^3,\ \cdots$$

But the weights don't have to be powers of the base. An example that isn't is the binary code 6, 3, 1, -1 for integers 0-9. In this code for example

$$1101 = 1\cdot 6 + 1\cdot 3 + 0\cdot 1 + 1\cdot(-1) = 8$$

Find and put in a Table the 4-bit weighted binary 6, 3, 1, –1 code for the numbers 0 to 9

5. Up to now we've been getting our decimal numbers in terms of 1's and 0's by converting the "whole number at once" to binary. An alternative approach is to separately convert each digit of a decimal number to a 4 bit binary number like in the following example

$$94_{10} = 1001\ 0100_{BCD}$$

We call this **Binary Coded Decimal or BCD** code. **Memorize** this definition. Now as it turns out BCD numbers are somewhat cumbersome to add and subtract but they are very easy to obtain and as a result very often the first step in converting decimal numbers to binary

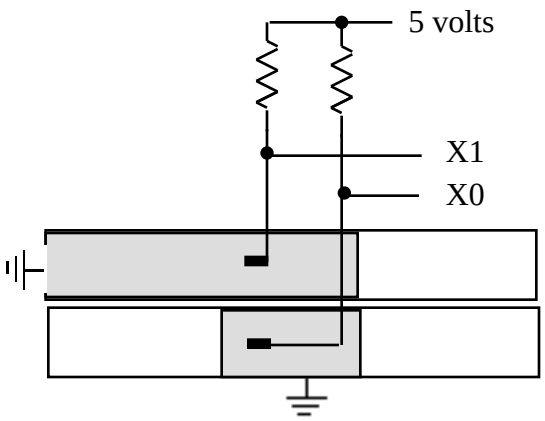
a. Convert 579 to BCD
b. Convert the BCD number 1001 0010 0110 to decimal

6. Up to now we've been solely focusing on getting numbers in terms of 1's and 0's. But for our digital systems to be able to interface with keyboards we need codes for the keyboard's letters and symbols. The standard code for doing this is referred to as the **ASCII code**.

a. Find out how many bits are used in the ASCII code
b. Find the ASCII codes for 3, A, a and ? in binary. Put your results in a Table

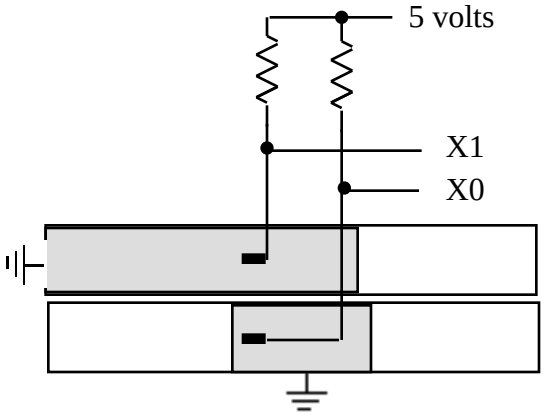
7. And last but not least if we're going to be able to build digital controllers for mechanical systems like automobile engines we need to be able to supply our digital systems with 1's and 0's for the physical locations of moving parts like the angle of a rotating shaft. Suppose in particular that

we're using wire brushes and conducting tape to specify the location of an object moving back and forth along a straight line as follows



When a brush is in contact with a grounded conductor then the corresponding X is LOW and when in contact with an insulator X is HIGH. So as the brushes move from left to right the code X_1X_0 giving us the location of the brushes produces the sequence $01 \rightarrow 00 \rightarrow 11$.

Now this all looks innocent enough but in the real world the conducting tapes will probably not be lined up exactly and we'll actually have something like the following

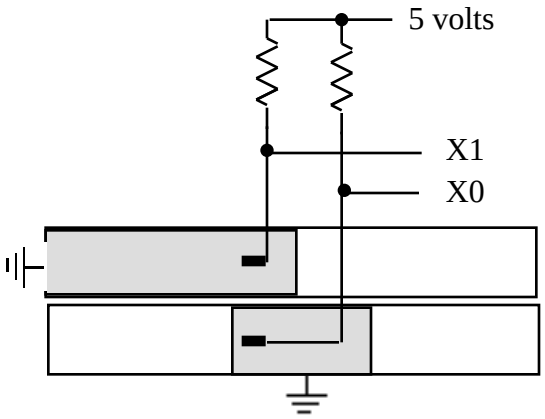


Then the brushes will generate the incorrect sequence $01 \rightarrow 00 \rightarrow 10 \rightarrow 11$ as the brushes move from left to right. The problem is that more than one bit is changing at a time as the brushes move from one section to the next.

The way we get around this problem is to use codes like the following

0 0 0
0 0 1
0 1 1
⋮

in which at most one bit changes at a time as follows



as the brushes move from one section to the next. We call these codes **Gray codes**. **Memorize** the definition of a Gray code. Then

a. Find the remaining 5 members of the 3-bit Gray code started above
b. Shade in the areas for your 3-bit Gray code on a horizontal strip like the one above
c. Why don't we have to worry about the exact alignment of the conducting tapes on your horizontal strip in part (b)