Codes

A single bit is useful if exactly two answers to a question are possible. Examples include the result of a coin toss (heads or tails), Most situations in life are more complicated. This chapter concerns ways in which complex objects can be represented not by a single bit, but by arrays of bits.

It is convenient to focus on a very simple model of a system, shown in Figure 2.1, in which the input is one of a predetermined set of objects, or " symbols," the identity of the particular symbol chosen is encoded in an array of bits, these bits are transmitted through space or time, and then are decoded at a later time or in a different place to determine which symbol was originally chosen.

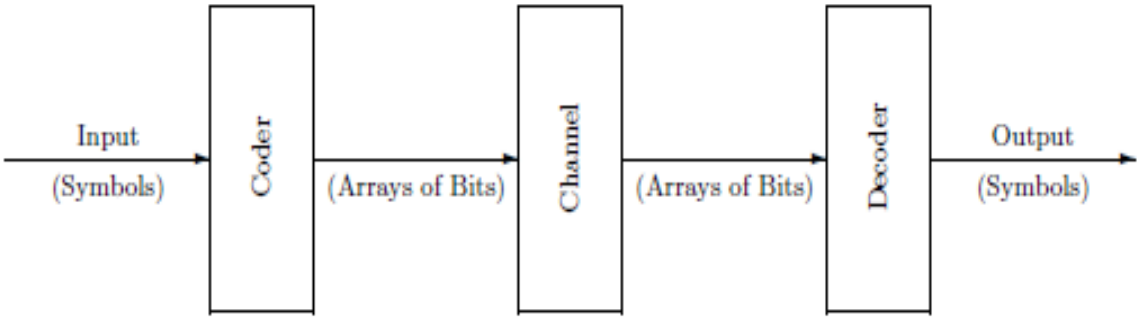

Figure 2.1: Simple model of a communication system

Some objects for which codes may be needed include:

- Letters: BCD, EBCDIC, ASCII, Unicode, Morse Code
  Integers: Binary, Gray, 2's complement
- Numbers: Floating-Point
  Proteins: Genetic Code

  Telephones: NANP, International codes
- Hosts: Ethernet, IP Addresses, Domain names
- Images: TIFF, GIF, and JPEG
  Audio: MP3

  Video: MPEG

Symbol Space Size

The first question to address is the number of symbols that need to be encoded. This is called the symbol space size. We will consider symbol spaces of different sizes:

1, 2, Integral power of 2, Finite, Infinite Countable, Infinite Uncountable.

a- As a special case, if there is only one symbol, no bits are required to specify it.

b- If the number of symbols is 2, then the selection can be encoded in a single bit.

c- If the number of possible symbols is 4, 8, 16, 32, 64, or another integral power of 2, then the selection may be coded in the number of bits equal to the logarithm, base 2, of the symbol space size. Thus 5 bits can encode the selection of one student in a class of 32.

d- If the number of symbols is finite but not an integral power of 2, then the number of bits that would work is the next higher integral power of 2 can be used to encode the selection, but there will be some unused bit patterns.

Examples include the 10 digits, the six faces of a cubic die, the 13 denominations of a playing card, and the 26 letters of the English alphabet. In each case, there is spare capacity (6 unused patterns in the 4-bit representation of digits, 2 unused patterns in the 3-bit representation of a die, etc.)

e - If the number of symbols is infinite but countable then a bit string of a given length can only denote a finite number of items from this infinite set. Thus, a 4-bit code for non-negative integers might designate integers from 0 through 15, but would not be able to handle integers outside this range.

f- If the number of symbols is infinite and uncountable (such as the value of a physical quantity like voltage) then some technique of "discretization" must be used to replace possible values by a finite number of selected values that are approximately the same. For example, if the numbers between 0 and 1 were the symbols and if 2 bits were available for the coded representation, one approach might be to approximate all numbers between 0 and 0.25 by the number 0.125, all numbers between 0.25 and 0.5 by 0.375, and so

on. Whether such an approximation is adequate depends on how the decoded data is used.

Binary Coded Decimal (BCD)

A common way to represent the digits 0 -9 is by the ten four-bit patterns shown in Table 2.1. There are six bit patterns (for example 1010) that are not used, and the question is what to do with them. Here are a few ideas that come to mind.

The unused patterns might be mapped into legal values. For example, the unused patterns might all be converted to 9, under the theory that they represent 10, 11, 12, 13, 14, or 15, and the closest digit is 9. Or they might be decoded as 2, 3, 4, 5, 6, or 7, by setting the initial bit to 0, under the theory that the first bit might have gotten corrupted.

Digit	Code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Table 2.1: Binary Coded Decimal

ASCII

The most commonly used code for text characters, ASCII (American Standard Code for Information Interchange, described in Section 2.5) reserves 33 of its 128 codes explicitly for control, and only 95 for characters. These 95 include the 26 upper-case and 26 lower-case letters of the English alphabet, the 10 digits, space, and 32 punctuation marks.

Uppercase

HEX	DEC	CHR
40	64	@
41	65	A
42	66	B
43	67	C
44	68	D
45	69	E
46	70	F
47	71	G
48	72	H
49	73	I

Detail: Integer Codes

There are many ways to represent integers as bit patterns. All suffer from an inability to represent arbitrarily large integers in a fixed number of bits. A computation which produces an out-of-range result is said to overflow.

The most commonly used representations are binary code for unsigned integers (e.g., memory addresses), 2's complement for signed integers (e.g., ordinary arithmetic), and binary gray code for instruments measuring changing quantities.

The following table gives five examples of 4-bit integer codes. The MSB (most significant bit) is on the left and the LSB (least significant bit) on the right

	Unsigned Integers		Signed Integers		
	Binary Code	Binary Gray Code	2's Complement	Sign/Magnitude	1's Complement
Range →	[0, 15]	[0, 15]	[-8, 7]	[-7,7]	[-7,7]
-8			1000		
-7			1001	1111	1000
-6			1010	1110	1001
-5			1011	1101	1010
-4			1100	1100	1011
-3			1101	1011	1100
-2			1110	1010	1101
-1			1111	1001	1110
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001
2	0010	0011	0010	0010	0010
3	0011	0010	0011	0011	0011
4	0100	0110	0100	0100	0100
5	0101	0111	0101	0101	0101
6	0110	0101	0110	0110	0110
7	0111	0100	0111	0111	0111
8	1000	1100			
9	1001	1101			
10	1010	1111			
11	1011	1110			
12	1100	1010			
13	1101	1011			
14	1110	1001			
15	1111	1000			

Table 2.4: Four-bit integer codes

Binary Code

This code is for nonnegative integers. For code of length n , the 2^n patterns represent integers 0 through 2^n -1. The LSB (least significant bit) is 0 for even and 1 for odd integers.

Binary Gray Code

This code is for nonnegative integers. For code of length n, the 2^n patterns represent integers 0 through 2^n-1. The two bit patterns of adjacent integers differ in exactly one bit. This property makes the code useful for sensors where the integer being encoded might change while a measurement is in progress.

2's Complement

This code is for integers, both positive and negative. For a code of length n, the 2^n patterns represent integers - 2^{n-1} through 2^{n-1} -1. The LSB (least significant bit) is 0 for even and 1 for odd integers. Where they overlap, this code is the same as binary code. This code is widely used.

Sign/Magnitude

This code is for integers, both positive and negative. For code of length n, the 2^n patterns represent integers - (2^{n-1} -1) through $2^n - 1$ -1. The MSB (most significant bit) is 0 for positive and 1 for negative integers; the other bits carry the magnitude. Where they overlap, this code is the same as binary code. While conceptually simple.

1's Complement

This code is for integers, both positive and negative. For code of length n, the 2^n patterns represent integers - (2^{n-1} - 1) through 2^{n-1} -1. The MSB is 0 for positive integers; negative integers are formed by complementing each bit of the corresponding positive integer. Where they overlap, this code is the same as binary code.