Chapter 2 Numeric Representation.

Most of the things we encounter in the world around us are analog; they don't just take on one of two values. How then can they be represented digitally? The key is that multiple signals can be grouped to represent a single quantity. A group of two or more signals is called a bus (or multi-bit signal). A bus consisting of two signals is called a 2-bit bus, and it can take on one of four values: low-low, low-high, high-low and high-high. A 3-bit bus can take on eight distinct values, and in general, an n-bit bus can take on 2^n distinct values.

The next question, then, is how can we make those 2^n values correlate to real-world signals? To answer that, we must first understand binary.

2.1 Binary

Binary, like decimal, is a *positional number system*. This means that numbers are represented by a sequence of symbols, and that the significance of each symbol depends both on its position and the base of the number system. Decimal numbers are base 10 and the symbols are called *digits*. Binary numbers are base 2 and the symbols are called *bits*.

Any positive integer can be represented in a positional number system by a sequence of symbols and a base. Specifically, a positive integer in base b that has m symbols is written:

$$S_{m-1}\, S_{m-2} \ldots S_1 S_0\, {}_b. \qquad (2.1)$$

The base is written in subscript and should be included with the number unless it is in decimal or the base is clear from context. A number, N, in this positional form can be evaluated using the expansion:

$$N = S_{m-1} \times b^{m-1} + \ldots + S_1 \times b^1 + S_0 \times b^0 \qquad (2.2)$$

Example 2.1. Find the decimal equivalents of 1101_2 and 734_8.

Solution: Using equation 1.2, 1101_2 and 734_8 expand to:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13$$
$$734_8 = 7 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 7 \times 64 + 3 \times 8 + 4 \times 1 = 476$$

The powers of *b* in equation 1.2 are called *weights*. Symbols with larger weights are more significant than those with smaller weights. The leftmost symbol is the most significant of all. In decimal, this is called the *most significant digit*. In binary it is called the *most significant bit*. Similarly, the rightmost symbol is called the *least significant digit*, or the *least significant bit*, depending on the base.

Positional numbering systems can be extended to fractions by adding a *radix point* and additional symbols:

$$S_{m-1}\, S_{m-2} \ldots S_1 S_0 . S_{-1} S_{-2} \ldots b \qquad (2.3)$$

In decimal or binary, the radix point is called the *decimal point* or the *binary point*, respectively. The equation for evaluating positional numbers with a fractional part is similar to that for integers:

$$N = S_{m-1} \times b^{m-1} + \ldots + S_1 \times b^1 + S_0 \times b^0 + S_{-1} \times b^{-1} + S_{-2} \times b^{-2} \ldots \qquad (2.4)$$

Example 2.2. Convert 1.011_2 and 3.14_5 to decimal.

Solution: Using equation 1.4, 1.011_2 and 3.14_5 expand to:

$$1.011_2 = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 1 + \frac{1}{4} + \frac{1}{8} = 1\frac{3}{8} = 1.375$$

$$3.14_5 = 3 \times 5^0 + 1 \times 5^{-1} + 4 \times 5^{-2} = 3 + \frac{1}{5} + \frac{4}{25} = 3\frac{9}{25} = 3.36$$

There is an easier technique to convert a binary number to decimal. Start by writing the binary number down, then write a weight of 1 above the 1's bit (just left of the binary point). Double the weight for each bit left of the 1's bit and halve it for each bit to the right. Finally, get the decimal equivalent by adding together all the weights that have a 1 beneath them.

For example, to use this technique to convert 10110.101_2 to decimal, we would write down the binary number and write the weight of each bit above it:

Weight:	16	8	4	2	1		0.5	0.25	0.125
Binary Number:	1	0	1	1	0	.	1	0	1

The sum the weights that that have a 1 beneath them is 16+4+2+0.5+0.125 = 22.625.

Example 2.3. Convert 11010010_2 to decimal.

Solution: Write down the binary number and write the weight of each bit above it. Since there is no binary point, we start with the rightmost bit.

Weight:	128	64	32	16	8	4	2	1
Binary Number:	1	1	0	1	0	0	1	0

Sum the weights that that have a 1 beneath them:

128+64+16+2 = 210

Suppose now we have a decimal number and wish to convert it to a binary number. It is possible to use Equation 1.4, but then all the arithmetic would have to be done in binary. There is another technique that is easier and uses decimal arithmetic. The steps are outlined on the top of the next page.

1. First separate the integer part from the fractional part.
2. Repeatedly divide the integer part by 2, keeping the remainders. Do this until the integer part becomes 0.
3. Write down the remainders in reverse order. This is the binary representation of the integer part.
4. Repeatedly multiply the fractional part by 2, setting aside the integer part each time. Do this until the desired number of bits of precision have been obtained.
5. Write down a binary point and the integer parts from step 4 in forward order.

This may seem complicated, but it isn't really, especially since many binary numbers are integers and for integers, steps 4 and 5 may be omitted.

Example 2.4. Convert 217 to binary.

Solution: Repeatedly divide 217 by 2, keeping the remainders:

$$
\begin{aligned}
217 \div 2 &= 108 \text{ remainder } 1 \\
\div 2 &= 54 \text{ remainder } 0 \\
\div 2 &= 27 \text{ remainder } 0 \\
\div 2 &= 13 \text{ remainder } 1 \\
\div 2 &= 6 \text{ remainder } 1 \\
\div 2 &= 3 \text{ remainder } 0 \\
\div 2 &= 1 \text{ remainder } 1 \\
\div 2 &= 0 \text{ remainder } 1
\end{aligned}
$$

Write the remainder down in reverse order:

$$11011001_2$$

Example 2.5. Convert 0.78 to binary. Provide 7 fractional bits.

Solution: Multiply 0.78 by 2, setting aside the integer part.

$$
\begin{aligned}
0.79 \times 2 &= 1.58 \\
0.58 \times 2 &= 1.16 \\
0.16 \times 2 &= 0.32 \\
0.32 \times 2 &= 0.64 \\
0.64 \times 2 &= 1.28 \\
0.28 \times 2 &= 0.56 \\
0.56 \times 2 &= 1.12
\end{aligned}
$$

Write the binary point and the integer parts down in forward order:

$$0.1100101_2$$

2.2 Hexidecimal

It should now be clear that binary is the ideal numeric representation for digital circuitry. Each bit must either be 1 or 0 just as each signal must either be high or low. Binary is not, however, a convenient representation for humans. Digital computers, for example, routinely deal with 32-bit and 64-bit numbers. Writing down a number of this size with ones and zeros would be awkward to say the least. What we need is a number system with a richer set of symbols that still corresponds directly to digital signals, and that is exactly what the hexadecimal (base 16) number system provides.

Because hexadecimal is base 16, it requires 16 symbols. We use the familiar 0-9 for the first ten symbols and hijack the first six letters of the English alphabet for the rest. The utility of hexadecimal (or hex) stems from the fact that $16 = 2^4$. This means that a hexadecimal digit comprises exactly four bits, which makes conversion from binary almost trivial. To convert a binary number to hex, group the bits four at a time (padding with zeros if necessary), starting at the binary point. Then replace each group with the corresponding hexadecimal digit from Table 1-2.

Table 1-2 Binary and Decimal Representation of Hexadecimal Digits

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Hex	Binary	Decimal
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Example 2.6. Convert 10110110100.011011_2 to hexadecimal.

Solution: Group bits, four at a time, starting at the binary point.

101 1011 0100 . 0110 11

The leftmost and rightmost groups have fewer than four bits, so pad them with zeros.

0101 1011 0100 . 0110 1100

Replace the groups with hexadecimal digits from Table 1-2:

$5B4.6C_{16}$

Converting a number from hexadecimal to binary is equally easy. Simply replace each hexadecimal digit with its binary equivalent from Table 1-2.

Example 2.7. Convert $3D.7A_{16}$ to binary.

Solution: Replace each hexadecimal digit with its binary equivalent.

$$0011\ 1101\ .\ 0111\ 1010$$

Discard unnecessary zeros:

$$111101.0111101_2$$

2.3 Two's Complement

Binary representation works well for positive numbers, but what about negative numbers? How can they be represented with digital signals? One approach is to dedicate one bit, called the sign bit, to indicate whether or not a number is negative. Despite the nuisance of having both a positive and negative zero, this approach is commonly used to represent numbers in *floating point*. Unlike the binary representation discussed in Section 1.3, floating point representations allow the binary point to move (or float) in order to take maximum advantage of the available bits. Floating point representation is beyond the scope of this text, but students interested in the subject may refer to the IEEE 754 floating point standard for additional information.

A second way to represent negative numbers that is far more prevalent for integers and fixed point numbers is called *Two's Complement Representation*, or simply *Two's Complement*. The reason for this popularity will become clear when we discuss addition and subtraction in Chapter 5.

Two's complement is very much like binary, so much in fact that there is no notation to distinguish the two. It is always necessary to know from context whether a sequence of bits (or hexadecimal digits) represents a binary or two's complement number.

Two's complement differs from binary in two ways. First, the number of bits used to represent a number is fixed beforehand. Second, the weight of the most significant bit is negative. To be precise, if an m-bit two's complement number ($B_{m-1}\ldots B_1B_0$) is used to represent an integer, N, then the value of N can be evaluated using the expansion:

$$N = -B_{m-1} \times 2^{m-1} + \ldots + B_1 \times 2^1 + B_0 \times 2^0 \qquad (2.5)$$

Extending Equation 1.5 to include fractional bits is straightforward and is left as an exercise for the student.

Example 2.8: Convert the 8-bit two's complement number 10010011_2 to decimal.

Solution: Using equation 1.5, 10010011_2 expands to:

$$10010011_2 = -1 \times 2^7 + 1 \times 2^4 + 1 \times 2^1 + 1 \times 2^0 = -128 + 16 + 2 + 1 = -109$$

The technique of writing weights above each bit and summing (See Example 2.3) may also be used for two's complement numbers provided (a) all m bits are shown and (b) the weight of the leftmost bit is negated.

Example 2.9: Convert the 8-bit two's complement number 10010011_2 to decimal by assigning weights and summing.

Solution:

Weight:	–128	64	32	16	8	4	2	1
Binary Number:	1	0	0	1	0	0	1	1

Sum the weights that that have a 1 beneath them:

$$-128+16+2+1 = -109$$

If a number is negative, its leftmost bit is always 1. If a number is 0 or positive, its leftmost bit is 0. For this reason, the leftmost bit of a two's complement number is often called the *sign bit*.

Now suppose we have a decimal integer and wish to convert it into an *m*-bit two's complement number. One way to accomplish the conversion is given below:

1. Verify that the number is in range. The largest positive integer that can be represented by an *m*-bit two's complement number is $2^{m-1} - 1$. The most negative integer that can be represented is -2^{m-1}. If the number is outside of this range, conversion is not possible.
2. If the number is negative, write 1 for the sign bit and add 2^{m-1}. Otherwise write 0 for the sign bit.
3. Divide the number by 2, keeping the remainder. Repeat this step *m*-1 times.
4. Write down the remainders in reverse order to the right of the sign bit.

Example 2.10. Convert –109 to 8-bit two's complement.

Solution: $2^{m-1} = 128$, so the valid range of integers is (–128…127). –109 is within this range so conversion is possible. The number is negative so we write down 1 for the sign bit, then add it to 128: 128 + (–109) = 19.

Divide by 2 seven times, keeping the remainder each time:

$$\begin{aligned} 19 \div 2 &= 9 \text{ remainder } 1 \\ \div 2 &= 4 \text{ remainder } 1 \\ \div 2 &= 2 \text{ remainder } 0 \\ \div 2 &= 1 \text{ remainder } 0 \\ \div 2 &= 0 \text{ remainder } 1 \\ \div 2 &= 0 \text{ remainder } 0 \\ \div 2 &= 0 \text{ remainder } 0 \end{aligned}$$

Finally, write down the remainders in reverse order to the right of the 1 (sign bit) we wrote earlier:

$$10010011_2$$

Note that this same sequence of bits would be interpreted 128+16+2+1 = 147 in binary. This is why it is important to know from context whether binary or two's complement representation is being used.

Example 2.11. Convert 2056 into 12-bit two's complement form.

Solution: $2^{m-1} = 2048$, so the valid range of integers is (–2048…2047). 2056 falls outside of this range so conversion is not possible.

Example 2.12 Convert 108 to 8-bit two's complement.

Solution: $2^{m-1} = 128$, so the valid range of integers is (–128…127). 108 is within this range so conversion is possible. The number is not negative so we write down 0 for the sign bit.

Divide by 2 seven times, keeping the remainder each time:

$$\begin{aligned} 108 &\div 2 = 54 \text{ remainder } 0 \\ &\div 2 = 27 \text{ remainder } 0 \\ &\div 2 = 13 \text{ remainder } 1 \\ &\div 2 = 6 \text{ remainder } 1 \\ &\div 2 = 3 \text{ remainder } 0 \\ &\div 2 = 1 \text{ remainder } 1 \\ &\div 2 = 0 \text{ remainder } 1 \end{aligned}$$

Write down the remainder in reverse order to the right of the 0 (sign bit) we wrote earlier:

$$01101100_2$$

If we wish to add a binary point to the two's complement representation so that there are *n* fractional bits, conversion from decimal becomes a little more involved. The easiest approach is to first multiply the decimal number by 2^n and then proceed as if it were an integer. After the integer conversion is complete, divide the result by 2^n by moving the binary point to the left *n* bits.

Example 2.13. Convert 12.38 to an 8-bit two's complement number with 3 bits of fraction.

Solution: First multiply –12.38 by $2^3 = –99.04$. Next, convert –99 to 8-bit two's complement:

$$–99 \rightarrow 01011101_2$$

Finally, move the binary point left 3 bits.

$$01011.101_2$$

Two's Complement numbers may also be represented in hexadecimal. The conversion to hexadecimal starts the same as it does for binary: Group bits four at a time starting with the binary point, then pad the rightmost group with zeros if it contains less than four bits. The difference is that if the leftmost group has less than four bits, pad it with copies of the sign bit instead of zeros. Finally, convert each group to a hexadecimal digit as before.

Example 2.14. Represent the following 10-bit two's complement numbers in hexadecimal:

(a) 1001011011_2

(b) 0110001.110_2

Solution: Group bits, pad if necessary and convert:

$$1001011011_2 \rightarrow 1110\ 0101\ 1011 \rightarrow \mathrm{E5B}_{16}$$
$$0110001.110_2 \rightarrow 0011\ 0001.1100 \rightarrow 31.\mathrm{C}_{16}$$

The technique of duplicating the sign bit is called *sign extension*. It provides a way to increase the number of bits used to represent a two's complement number without changing the number itself.

Example 2.15. Convert the 9-bit two's complement numbers 1011.10101_2 and 011010011_2 into 12-bit two's complement numbers.

Solution: duplicate the sign bit 12-9 = 3 times:

$$1011.10101_2 \rightarrow 1111011.10101_2$$
$$011010011_2 \rightarrow 000011010011_2$$

Exercises

1. Convert the following binary numbers to decimal.
   (a) 11011 (c) 10110010 (e) 100101101101
   (b) 1.0110 (d) 11101.011 (f) 1100111101.10
2. Convert the following decimal numbers to binary. If the fractional part does not terminate, give four (4) bits of precision.
   (a) 49 (c) 185 (e) 2943
   (b) 6.125 (d) 15.82 (f) 361.5
3. Convert the binary numbers in problem 4 to hexadecimal.
4. Convert the following hexadecimal numbers to binary.
   (a) FE (c) A57 (e) 1FFF
   (b) 7.A (d) 19.8 (f) 13B.D
5. Convert the hexadecimal numbers in Problem 7 to decimal, first by using Equation 1-2 then by converting your (binary) answers from Problem 7.
6. Extend Equation 1.5 to include *n* fractional bits. The total number of bits remains *m*.
7. Convert the following 8-bit two's complement numbers to decimal.
   (a) 01100101 (c) 11111111 (e) 01101101
   (b) 110.01100 (d) 010110.01 (f) 111101.10
8. Convert the following decimal numbers to 8-bit two's complement. Assume no fractional bits. If a number cannot be represented, explain why it cannot.
   (a) 97 (c) –128 (e) –3
   (b) –100 (d) 128 (f) 56

9. Convert the following decimal numbers to 8-bit two's complement. Assume 4 fractional bits. If a number cannot be represented, explain why it cannot.
   (a) 6.44 (b) -7.93 (c) –4.25
10. Convert the two's complement numbers in problem 10 to hexadecimal.
11. Convert the following 2's complement numbers (base 16) into 12-bit two's complement (base 2).
   (a) 1FE (b) A63 (c) 80B
12. Sign-extend the 12-bit 2's complement numbers (base 16) in Problem 14 to 16 bits.

Bibliography