1.5. Digital Representation of Numbers

1.5.1. Fundamentals

Information is stored on the computer in binary form. A binary bit can exist in one of two possible states. In **positive logic**, the presence of a voltage is called the '1', true, asserted, or high state. The absence of a voltage is called the '0', false, not asserted, or low state. Conversely in **negative logic**, the true state has a lower voltage than the false state. Figure 1.25 shows the output of a typical complementary metal oxide semiconductor (CMOS) circuit. The left side shows the condition with a true bit, and the right side shows a false. The output of each digital circuit consists of a p-type transistor "on top of" an n-type transistor. In digital circuits, each transistor is essentially on or off. If the transistor is on, it is equivalent to a short circuit between its two output pins. Conversely, if the transistor is off, it is equivalent to an open circuit between its outputs pins. On TM4C microcontrollers powered with 3.3 V supply, a voltage between 2 and 5 V is considered high, and a voltage between 0 and 1.3 V is considered low. Separating the two regions by 0.7 V allows digital logic to operate reliably at very high speeds. The design of transistor-level digital circuits is beyond the scope of this book. However, it is important to know that digital data exist as binary bits and encoded as high and low voltages.

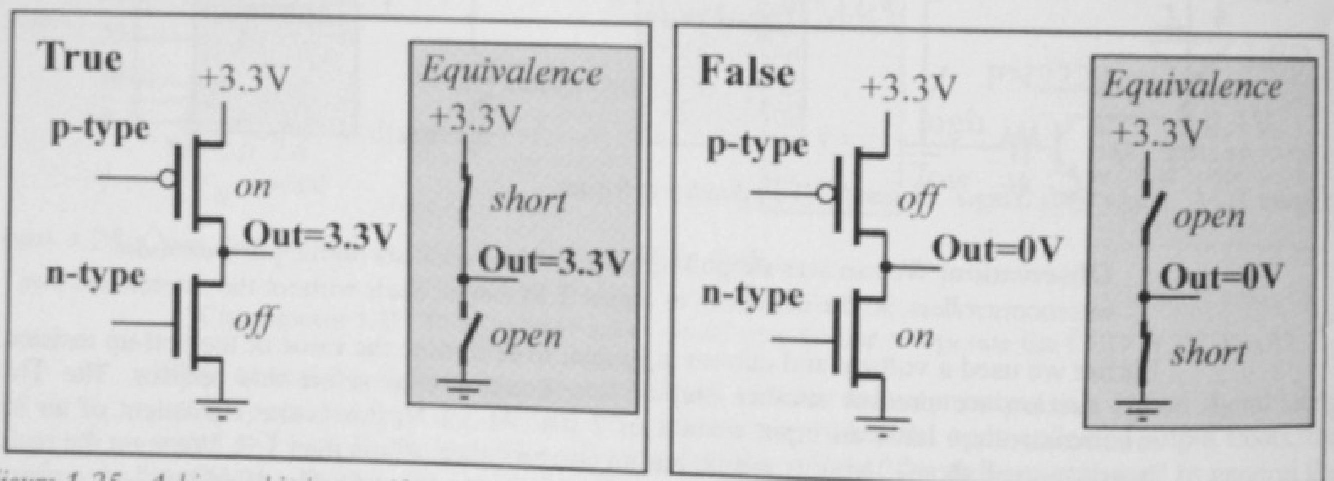

Figure 1.25. A binary bit is true if a voltage is present and false if the voltage is 0.

Start reading here

Numbers are stored on the computer in binary form. In other words, information is encoded as a sequence of 1's and 0's. On most computers, the memory is organized into 8-bit bytes. This means each 8-bit byte stored in memory will have a separate address. **Precision** is the number of distinct or different values. We express precision in alternatives, decimal digits, bytes, or binary bits. **Alternatives** are defined as the total number of possibilities as listed in Table 1.7. Let the operation [[x]] be the greatest integer of x. E.g., [[2.1]] is rounded up to 3. For example, an 8-bit number scheme can represent 256 different numbers, which means 256 alternatives. An 8-bit digital to analog converter (DAC) can generate 256 different analog outputs. An 8-bit analog to digital converter (ADC) can measure 256 different analog inputs.

Binary bits	Bytes	Alternatives
8	1	256
10	2	1024
12	2	4096
14	2	16,384
16	2	65,536
20	3	1,048,576
24	3	16,777,216
30	3	1,073,741,824
32	4	4,294,967,296
64	8	18,446,744,073,709,551,616
n	[[n/8]]	2^n

Table 1.7. Relationship between bits, bytes and alternatives as units of precision.

Observation: A good rule of thumb to remember is $2^{10 \cdot n}$ is approximately $10^{3 \cdot n}$.

Decimal digits are used to specify precision of measurement systems that display results as numerical values, as defined in Table 1.8. A full decimal digit can be any value 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. A digit that can be either 0 or 1 is defined as a ½ decimal digit. The terminology of a ½ decimal digit did not arise from a mathematical perspective of precision, but rather it arose from the physical width of the LED/LCD module used to display a blank or '1' as compared to the width of a full digit. Similarly, we define a digit that can be + or - also as a half decimal digit, because it has two choices. A digit that can be 0,1,2,3 is defined as a ¾ decimal digit, because it is wider than a ½ digit but narrower than a full digit. We also define a digit that can be -1, -0, +0, or +1 as a ¾ decimal digit, because it also has four choices. We use the expression 4½ decimal digits to mean 20,000 alternatives and the expression 4¾ decimal digits to mean 40,000 alternatives. The use of a ½ decimal digit to mean twice the number of alternatives or one additional binary bit is widely accepted. On the other hand, the use of ¾ decimal digit to mean four times the number of alternatives or two additional binary bits is not as commonly accepted. For example, consider the two ohmmeters. Assume both are set to the 0 to 200 kΩ range. A 3½ digit ohmmeter has a resolution of 0.1 kΩ with measurements ranging from 0.0 to 199.9 kΩ. On the other hand, a 4½ digit ohmmeter has a resolution of 0.01 kΩ with measurements ranging from 0.00 to 199.99 kΩ. Table 1.8 illustrates decimal-digit representation of precision.

Decimal digits	Alternatives
3	1000
3½	2000
3¾	4000
4	10,000
4½	20,000
4¾	40,000
5	100,000
n	10^n

Table 1.8. Definition of decimal digits as a unit of precision.

Checkpoint 1.18: How many binary bits correspond to 2½ decimal digits?

Checkpoint 1.19: How many decimal digits correspond to 10 binary bits?

Checkpoint 1.20: How many binary bits correspond to 6½ decimal digits?

Checkpoint 1.21: About how many decimal digits can be presented in a 64-bit 8-byte number? You can answer this without a calculator, just using the "rule of thumb".

The **hexadecimal** number system uses base 16 as opposed to our regular decimal number system that uses base 10. Hexadecimal is a convenient mechanism for humans to represent binary information, because it is extremely simple for us to convert back and forth between binary and hexadecimal. Hexadecimal number system is often abbreviated as "hex". A nibble is defined as four binary bits, which will be one hexadecimal digit. In mathematics, a subscript of 2 means binary, but in this book we will define binary numbers beginning with %. In assembly language however, we will use hexadecimal format when we need to define binary numbers. The hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Some assembly languages use the prefix $ to signify hexadecimal, and in C we use the prefix 0x. To convert from binary to hexadecimal, you simply separate the binary number into groups of four binary bits (starting on the right), then convert each group of four bits into one hexadecimal digit. For example, if you wished to convert 10100111_2, first you would group it into nibbles 1010 0111, then you would convert each group 1010=A and 0111=7, yielding the result of 0xA7. To convert hexadecimal to binary, you simply substitute the 4-bit binary for each hexadecimal digit. For example, if you wished to convert 0xB5D1, you substitute B=1011, 5=0101, D=1101, and 1=0001, yielding the result of 1011010111010001_2.

Checkpoint 1.22: Convert the binary number 111011101011_2 to hexadecimal.

Checkpoint 1.23: Convert the hex number 0x3800 to binary.

Checkpoint 1.24: How many binary bits does it take to represent 0x12345?

(The following info on large numbers is interesting but not essential for ENGR-384)

A great deal of confusion exists over the abbreviations we use for large numbers. In 1998 the International Electrotechnical Commission (IEC) defined a new set of abbreviations for the powers of 2, as shown in Table 1.9. These new terms are endorsed by the Institute of Electrical and Electronics Engineers (IEEE) and International Committee for Weights and Measures (CIPM) in situations where the use of a binary prefix is appropriate. The confusion arises over the fact that the mainstream computer industry, such as Microsoft, Apple, and Dell, continues to use the old terminology. According to the companies that market to consumers, a 1 GHz is 1,000,000,000 Hz but 1 Gbyte of memory is 1,073,741,824 bytes. The correct terminology is to use the SI-decimal abbreviations to represent powers of 10, and the IEC-binary abbreviations to represent powers of 2. The scientific meaning of 2 kilovolts is 2000 volts, but 2 kibibytes is the proper way to specify 2048 bytes. The term **kibibyte** is a contraction of kilo binary byte and is a unit of information or computer storage, abbreviated KiB.

1 KiB = 2^{10} bytes = 1024 bytes
1 MiB = 2^{20} bytes = 1,048,576 bytes
1 GiB = 2^{30} bytes = 1,073,741,824 bytes

These abbreviations can also be used to specify the number of binary bits. The term **kibibit** is a contraction of kilo binary bit, and is a unit of information or computer storage, abbreviated Kibit.

1 Kibit = 2^{10} bits = 1024 bits
1 Mibit = 2^{20} bits = 1,048,576 bits
1 Gibit = 2^{30} bits = 1,073,741,824 bits

A **mebibyte** (1 MiB is 1,048,576 bytes) is approximately equal to a megabyte (1 MB is 1,000,000 bytes), but mistaking the two has nonetheless led to confusion and even legal disputes. In the engineering community, it is appropriate to use terms that have a clear and unambiguous meaning.

Value	*SI Decimal*	*SI Decimal*	*Value*	*IEC Binary*	*IEC Binary*
1000^1	k	kilo-	1024^1	Ki	kibi-
1000^2	M	mega-	1024^2	Mi	mebi-
1000^3	G	giga-	1024^3	Gi	gibi-
1000^4	T	tera-	1024^4	Ti	tebi-
1000^5	P	peta-	1024^5	Pi	pebi-
1000^6	E	exa-	1024^6	Ei	exbi-
1000^7	Z	zetta-	1024^7	Zi	zebi-
1000^8	Y	yotta-	1024^8	Yi	yobi-

Table 1.9. Common abbreviations for large numbers.

1.5.2. 8-bit numbers

A **byte** contains 8 bits as shown in Figure 1.26, where each bit $b_7,...,b_0$ is binary and has the value 1 or 0. We specify b_7 as the most significant bit or MSB, and b_0 as the least significant bit or LSB. In C, the **unsigned char** or **uint8_t** data type creates an unsigned 8-bit number.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Figure 1.26. 8-bit binary format, created using either char or unsigned char (in C99 ***int8_t*** *or* ***uint8_t****).*

If a byte is used to represent an unsigned number, then the value of the number is

$$N = 128 \cdot b_7 + 64 \cdot b_6 + 32 \cdot b_5 + 16 \cdot b_4 + 8 \cdot b_3 + 4 \cdot b_2 + 2 \cdot b_1 + b_0$$

Notice that the significance of bit n is 2^n. There are 256 different unsigned 8-bit numbers. The smallest unsigned 8-bit number is 0 and the largest is 255. For example, 10000100_2 is 128+4 or 132.

Checkpoint 1.25: Convert the binary number 01101001_2 to unsigned decimal.

Checkpoint 1.26: Convert the hex number 0x23 to unsigned decimal.

The basis of a number system is a subset from which linear combinations of the basis elements can be used to construct the entire set. The basis represents the "places" in a "place-value" system. For positive integers, the basis is the infinite set {1, 10, 100…} and the "values" can range from 0 to 9. Each positive integer has a unique set of values such that the dot-product of the value-vector times the basis-vector yields that number. For example, 2345 is (…, 2,3,4,5)•(…, 1000,100,10,1), which is 2*1000+3*100+4*10+5. For the unsigned 8-bit number system, the basis is

{1, 2, 4, 8, 16, 32, 64, 128}

The values of a binary number system can only be 0 or 1. Even so, each 8-bit unsigned integer has a unique set of values such that the dot-product of the values times the basis yields that number. For example, 69 is (0,1,0,0,0,1,0,1)•(128,64,32,16,8,4,2,1), which equals 0*128+1*64+0*32+0*16+0*8+1*4+0*2+1*1.

Checkpoint 1.27: Give the representations of decimal 37 in 8-bit binary and hexadecimal.

Checkpoint 1.28: Give the representations of decimal 202 in 8-bit binary and hexadecimal.

One of the first schemes to represent signed numbers was called one's complement. It was called **one's complement** because to negate a number, you complement (logical not) each bit. For example, if 25 equals 00011001 in binary, then –25 is 11100110. An 8-bit one's complement number can vary from 127 to +127. The most significant bit is a sign bit, which is 1 if and only if the number is negative. The difficulty with this format is that there are two zeros +0 is 00000000, and –0 is 11111111. Another problem is that one's complement numbers do not have basis elements. These limitations led to the use of two's complement.

In C, the **char** or **int8_t** data type creates a signed 8-bit number. The **two's complement** number system is the most common approach used to define signed numbers. It was called two's complement because to negate a number, you complement each bit (like one's complement), and then add 1. For example, if 25 equals 00011001 in binary, then -25 is 11100111. If a byte is used to represent a signed two's complement number, then the value is

$$N = -128 \cdot b_7 + 64 \cdot b_6 + 32 \cdot b_5 + 16 \cdot b_4 + 8 \cdot b_3 + 4 \cdot b_2 + 2 \cdot b_1 + b_0$$

There are 256 different signed 8-bit numbers. The smallest signed 8-bit number is -128 and the largest is 127. For example, 10000010_2 equals -128+2 or -126.

Checkpoint 1.29: Are the signed and unsigned decimal representations of the 8-bit hex number 0x35 the same or different?

For the signed 8-bit number system the basis is

{1, 2, 4, 8, 16, 32, 64, -128}

The most significant bit in a two's complement signed number will specify the sign. An error will occur if you use signed operations on unsigned numbers, or use unsigned operations on signed numbers. To improve the clarity of our software, always specify the format of your data (signed versus unsigned) when defining or accessing the data.

Checkpoint 1.30: Give the representations of -31 in 8-bit binary and hexadecimal.

Observation: To take the negative of a two's complement signed number, we first complement (flip) all the bits, then add 1.

Many beginning students confuse a signed number with a negative number. A signed number is one that can be either positive or negative. A negative number is one less than zero. Notice that the same binary pattern of 11111111_2 could represent either 255 or -1. It is very important for the software developer to keep track of the number format. The computer cannot determine whether the 8-bit number is signed or unsigned. You, as the programmer, will determine whether the number is signed or unsigned by the specific assembly instructions you select to operate on the number. Some operations like addition, subtraction, and shift left (multiply by 2) use the same hardware (instructions) for both unsigned and signed operations. On the other hand, multiply, divide, and shift right (divide by 2) require separate hardware (instruction) for unsigned and signed operations.

1.5.3. Character information

We can use bytes to represent characters with the American Standard Code for Information Interchange (ASCII) code. Standard ASCII is actually only 7 bits, but is stored using 8-bit bytes with the most significant byte equal to 0. Some computer systems use the 8th bit of the ASCII code to define additional characters such as graphics and letters in other alphabets. The 7-bit ASCII code definitions are given in the Table 1.10. For example, the letter 'V' is in the 0x50 row and the 6 column. Putting the two together yields hexadecimal 0x56. The NUL character has the value 0 and is used to terminate strings. The '0' character has value 0x30 and represents the zero digit. In C and C99, we use the **char** data type for characters.

BITS 0 TO 3 \ BITS 4 to 6	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Table 1.10. Standard 7-bit ASCII.

Checkpoint 1.31: How is the character '0' represented in ASCII?

From "Real-Time Interfacing to ARM Cortex-M Microcontrollers", 5ed
by Jonathan Valvano

Further information on binary numbers

It can useful to know some powers of two, particularly powers from 0 to 7:

N	2^N	8-bit representation
0	1	0 0 0 0 0 0 0 1
1	2	0 0 0 0 0 0 1 0
2	4	0 0 0 0 0 1 0 0
3	8	0 0 0 0 1 0 0 0
4	16	0 0 0 1 0 0 0 0
5	32	0 0 1 0 0 0 0 0
6	64	0 1 0 0 0 0 0 0
7	128	1 0 0 0 0 0 0 0
8	256	
9	512	
10	1024	
11	2048	
12	4096	
13	8192	
14	16384	
15	32768	
16	65536	

The left most bit of a binary number is called the Most Significant Bit (MSB)
The right most bit of a binary number is called the Least Significant Bit (LSB)

4-bit binary numbers to illustrate signed numbers in 2's complement format.

The most significant bit indicates sign: 0 means positive
1 means negative

+7	0 1 1 1
+6	0 1 1 0
+5	0 1 0 1
+4	0 1 0 0
+3	0 0 1 1
+2	0 0 1 0
+1	0 0 0 1
0	0 0 0 0
-1	1 1 1 1
-2	1 1 1 0
-3	1 1 0 1
-4	1 1 0 0
-5	1 0 1 1
-6	1 0 1 0
-7	1 0 0 1
-8	1 0 0 0

Given a binary number with N bits, you can’t tell if it is signed or unsigned just by looking at the number. Someone has to tell you if it is signed or unsigned.

Here are questions to answer for Homework # 3:

(please don't just plug numbers into a calculator and use a convert button to convert between binary and decimal or decimal to binary, figure it out based on digit values)

1) What is the base 10 number for the unsigned binary value 0 1 0 1 0 0 1 1 ?

2) What is the unsigned 8-bit binary value for decimal 37?

3) What is the binary representation for minus one using a 5-bit binary number?

4) To represent a decimal 300 as an unsigned binary number, how many binary digits are required?

5) What is the hexadecimal representation of the binary number given in problem 1?

6) Assume 4-bit signed numbers are being used as shown on the precious page. What is the hexadecimal representation for minus one (-1)?

7) Hexadecimal numbers use characters A to F in addition to 0 to 9. List the letters A to F and give the base 10 equivalent value for each.