

Controlling The Real World With Computers

::: Control And Embedded Systems :::

Table Of Contents

Previous: [Programming Part 1](#)

Next: [Boolean Logic](#)

Data lines, bits, nibbles, bytes, words, binary and HEX

Information (often called data) inside a computer is exchanged among the various components by means of metallic conductors called **data lines**. A group of data lines is called a **data bus**.

Each data line carries a unit of data called a **bit**. A bit is the smallest piece of information most computers can work with. A bit can be on or off. **On** is usually considered to be **5 volts**, and **off** is considered to be **0 volts**, although modern systems often use lower **on** voltages to decrease power consumption.

Data can be represented on paper as a series of ones and zeros. A one means a bit is on, and a zero means it is off. A group of 8 bits is called a **byte**. A byte with a value of 0 would be represented as 00000000. Non-zero bytes can be any combination of 1s and 0s. 01100010 will be used as an example here. In the C language, a byte is called a character and is abbreviated **char**.

When data is represented as a series of ones and zeros, it is said to be a **binary** representation, or to have a **base of 2 because it uses 2 digits**. We humans use a base of 10, probably because we have 10 fingers.

The **left-end bit** of a number represented in binary is called the **most significant bit**, abbreviated **msb**, and the **right-end bit** is called the **least significant bit**, abbreviated **lsb**.

A little review might be helpful to those who are a little rusty on raising a number to a power. No high math here; to raise a number (the base in a numbering system) to a positive power, just write it down the exponent number of times and multiply. The exponent is the power to which a number is raised. One way to recognize an exponent is by the fact that it is often **raised** when written:

$$5^2 = 5 * 5 = 25$$

$$2^3 = 2 * 2 * 2 = 8$$

$$4^4 = 4 * 4 * 4 * 4 = 256$$

In a numbering system, each position has a **weight**. For all numbering systems I am aware of (the mathematicians probably know of others), the right, least-significant position is known as the 1's place. There, the weight is equal to the base raised to the power of 0.

Any number raised to the power of 0 is equal to 1.

In a numbering system, the exponent is increased by 1 with each move to the left. Thus, the second place from the right has a weight equal to the base raised to the power of 1.

Any number raised to the power of 1 is equal to itself.

To raise a number to a negative power, first remember that it is understood that there is a decimal to the right of a whole number. Thus, the number 2 can be expressed as 2, 2. or 2.0 and all three are equal.

So $2^{-3} = 2.^{-3} = 2.0^{-3}$ To get the decimal value, just move the decimal to the left the number of times indicated by the exponent = .002 .

We were taught in grade school that the second place from the right is the 10's place. That's because we were using a base of 10 and we were raising it to the power of 1. Since a base of 2 is used in binary, the second place from the right has a weight of 2 because it is 2 raised to the power of 1. The next weight is $2^2 = 2 * 2 = 4$, then $2^3 = 2 * 2 * 2 = 8$ and so on. Notice that a weight is the previous weight times the base.

The exponents are often used to designate a bit in a binary number. **Bit 0** is on the **right end** of the byte and **bit 7** is on the **left end**. Bit 0 is the lsb and bit 7 is the msb. Data bits are often abbreviated using the letter D, thus, D0, D1, D2, etc.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Base ^{exponent}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weights	128	64	32	16	8	4	2	1

The example binary number above was 01100010. To figure out what the decimal value is, simply add the weights for the bits that are turned on. In this case, bits 6, 5 and 1 are on. The total of their weights equals $64 + 32 + 2 = 98$.

A more general description of the procedure is to multiply the position weights by the values at the positions, then add them up. The example 01100010 would be:
 $(0 * 128) + (1 * 64) + (1 * 32) + (0 * 16) + (0 * 8) + (0 * 4) + (1 * 2) + (0 * 1) = 98$.

A common way of showing numbers in a C program is to use **hexadecimal notation**, or **HEX**. It uses a base of 16. Break a byte into two groups of 4 bits each: **nnnn nnnn**. Each group is called a **nibble**. A nibble with all low bits, 0000, is equal to 0. With all of its bits turned on, 1111, a nibble has a value of 15 ($8 + 4 + 2 + 1$). Thus, we are dealing with the 16 values 0 through 15 inclusive, and a base of 16.

Hexadecimal notation is simple. Just use digits for 0 through 9, and A through F for 10 through 15. The following table shows all of the combinations.

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	00	0	1000	08	8
0001	01	1	1001	09	9
0010	02	2	1010	10	A
0011	03	3	1011	11	B
0100	04	4	1100	12	C
0101	05	5	1101	13	D
0110	06	6	1110	14	E
0111	07	7	1111	15	F

The right nibble of a byte is the least significant nibble. It's the 1's place because it's 16^0 . Next is the 16's place because it's 16^1 , then $16^2 = 256$, and so on. To get the decimal value, take the value of the nibbles, multiply by the position weight values and add them up. Thus, the HEX value 9B = $(9 * 16) + (11 * 1) = 155$.

To show a number is hexadecimal in the C language, prefix it with **0x**. The above would be represented as 0x9B or 0x9b. This particular notation is not case-sensitive, although many things in C are.

The following shows the byte table again, but this time with the weights also expressed in hexadecimal notation, as often seen in C operations.

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Base ^{exponent}	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
Weights	128	64	32	16	8	4	2	1
HEX Weights	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01

A **word** is commonly 16 bits in embedded systems, D0 through D15. A table with the bit names and their relationship to the binary base of 2 is below.

Bit	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Base ^{exponent}	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰

The following shows the bits with their HEX weights.

Bit	D15	D14	D13	D12	D11	D10	D9	D8
HEX Weights	0x8000	0x4000	0x2000	0x1000	0x0800	0x0400	0x0200	0x0100
Bit	D7	D6	D5	D4	D3	D2	D1	D0
HEX Weights	0x0080	0x0040	0x0020	0x0010	0x0008	0x0004	0x0002	0x0001

A word can be broken up into 4 nibbles. It can be represented by showing its 4 nibbles as a 4-place hexadecimal number. For example, the decimal number 19070 can be represented as the hexadecimal number 0x4A7E.

$$\begin{aligned}
 0x4A7E &= (4 * 16^3) + (10 * 16^2) + (7 * 16^1) + (14 * 16^0) \\
 &= (4 * 4096) + (10 * 256) + (7 * 16) + (14 * 1) \\
 &= 19070.
 \end{aligned}$$

In the C language, a word is most often called an **integer**, abbreviated **int**. An integer can be used to represent numbers that range from negative to positive values, or numbers that have only positive values. In other words, an integer can be **signed** or **unsigned**. A signed integer can have either positive or negative values. An unsigned integer can only be positive. An unsigned 16-bit integer can have values from 0 through 65535. It is often abbreviated simply as **unsigned**.

Bit 15 is used as a sign bit for signed integers. If it is on, the number is negative. If it is off, it is positive. Positive values can range from 0 to 32767. Negative values can range from -1 to -32768. Some examples are shown below. Notice that the signed version is equal to $-1 * (65536 - \text{unsigned version})$. For example, to get the signed number from the unsigned value 49151,
 $\text{signed} = -1 * (65536 - 49151) = -16385$.

HEX	8000	BFFF	FFFE	FFFF	0000	3FFF	7FFE	7FFF
Signed	-32768	-16385	-0002	-0001	00000	16383	32766	32767
Unsigned	32768	49151	65534	65535	00000	16383	32766	32767

A **long** word is generally considered to be 4 bytes or 32 bits. A long is used for very large numbers. Longs can also be signed or unsigned. Signed longs have a range from -2,147,483,648 to 2,147,483,647. The maximum unsigned value is $0xFFFFFFFF = 4,294,967,295$. The minimum unsigned value is 0.

The following is a self-test over this section. It would be a very good idea to make sure you know the answers to all of the questions since the sections that follow will build on this one.

1) Data inside computers is exchanged among the different components by means of metal conductors called _____. A group is called a _____.

- A) Data Bus B) Nibble C) Binary D) Bit
 E) HEX F) Data Lines G) Weight

2) If the voltage is 5 volts, the bit is on. If the bit is off, the voltage is 0 volts.

- A) True B) False

3) A group of 8 bits is a _____ and might be called a _____ in the C programming language.

- A) Data Bus B) Nibble C) Byte D) Weight
 E) Bit F) Unsigned G) Signed H) Character (or char)

6

4) It is said to be a _____ when data is represented with a base of 2, because of the two digits used.

A) Integer (or int) B) Most Significant Bit C) Binary D) HEX

5) The left-end bit can also be referred to as the _____. The _____ is the right-end bit.

A) Word B) Integer (or int) C) Least Significant Bit (or lsb)
D) Character (or char) E) Data Bus F) Long Word
G) Data Lines H) Most Significant Bit (or msb)

6) In the 1's place, the _____ is equal to the base number raised to the power of 0.

A) Byte B) Weight C) Long Word D) Data Bus

7) _____ is a common way of showing numbers in a C program. It uses a base of 16.

A) Data Lines B) HEX C) Word D) Byte

8) If you break a byte into 2 groups of 4 bits each, then each group is a _____.

A) HEX B) Binary C) Nibble D) Volt

9) A _____ is commonly 16 bits, or 2 bytes in an embedded system. Something with 32 bits, or 4 bytes is usually called a _____.

A) Signed B) Long Word C) Byte D) Bit
E) Unsigned F) HEX G) Word H) Binary

10) The _____ integer can be a positive or negative word, and the _____ integer can only be a positive word.

A) Nibbles B) HEX C) Signed D) Data lines
E) Byte F) Bit G) Integer H) Unsigned

11. A _____ is the smallest piece of information most computers can work with.

A) Nibbles B) HEX C) Signed D) Data lines
E) Byte F) Bit G) Integer H) Unsigned

In the table on the next page, fill in the blanks and correct errors where needed.

Hints:

All of the information you need to figure out this table is above

There are 4 errors (HEX can be upper or lower case - it's not an error to use lower case for HEX).

MS Binary Nibble Means Most Significant Binary Nibble. ---- LS Binary Nibble Means Least Significant Binary Nibble.

MS HEX Nibble Means Most Significant HEX Nibble. ---- LS HEX Nibble Means Least Significant HEX Nibble.

Remember 8,4,2,1 for binary.

Remember that the base for HEX is 16.

The Weight for the Most Significant HEX Nibble = 16

The Weight for the Least Significant HEX Nibble = 1.

Remember, when you count in HEX you use 0 through 9, then A through F.

If you have binary but no HEX, use the binary to figure out the HEX.

If you have HEX but no binary, use the HEX to figure out the binary.

The first 16 numbers in the table simply count from 0 through 15.

Remember the binary and HEX for 0 to 15 and you have done over half of the table.

Notice that half of the numbers are on the left side of the table, and half are on the right side. 0 is on the left, 1 is on the right, 2 is on the left, 3 is on the right and so on.

That's true all the way through 15.

MS Binary Nibble	LS Binary Nibble	MS HEX Nibble	LS HEX Nibble	Decimal		MS Binary Nibble	LS Binary Nibble	MS HEX Nibble	LS HEX Nibble	Decimal
0000	0000	0	0	0		0000	0001			1
0000	0010	0	2					0	3	3
0000	0100			4		0000	0101	0	5	
		0	6	6		0000	00111			7
0000	1000	0	8					0	9	9
0000	1010			10		0000	1011	0	B	
0000	1100	0	H	12		0000	1101			13
0000	0000	0	E			0000	1111			15
1010	0000	A	0					6	4	100
				27				5	C	
0011	0111			55		0001	1101	1	d	
0011	0011	3	3	33			0000		0	192
		a	a					B	B	
		c	c					D	D	
		E	E					f	f	

Answers To The Above Questions

[Table Of Contents](#)

Previous: [Programming Part 1](#)

Next: [Boolean Logic](#)