computers

HOW THEY WORK, HOW YOU USE THEM, WHAT THEY CAN DO FOR YOU

BY PETE STARK

Bytes, bits, decimals, octals, hexadecimals: what are they? Here's the easy-to-understand story of computer machine language.

Numbers such as 96 or 302,529 are made up of *digits*; the number 96 consists of the two digits 9 and 6, while the number 302,529 consists of six digits. In general, all the numbers we use in day-to-day life are made up of the ten different digits 0 through 9. The number system we use is called the *decimal* system since the prefix *deci* means ten. It isn't just coincidence that we also happen to have exactly ten fingers. Early man used his fingers for counting, and another word for finger is digit. If we had twelve fingers, then most likely our numbers would use twelve different digits rather than just ten.

As the name implies, *digital* computers also work with digits. But since a computer does not have ten fingers, there is no reason why it should have to use exactly ten different digits. Because it happens to be convenient to build computers that way, computers work with just two different digits—0 and 1. Since these digits have only two values, we call them binary digits, or *bits*. Putting several of these bits together gives us a *binary number* such as 010 or 11001011.

The reason why bits are used rather than the decimal digits 0 through 9 is that a 0 or 1 is very easy to represent in an electrical circuit. Inside the computer, the 1 may be represented by a voltage on a wire, while the 0 is represented by either no voltage or a very small voltage.

Following the same idea, bits may be stored in a punched card as a hole for a 1 and no hole for a 0. In general, it is easy to store a 1 as the presence of something while a 0 is the absence of it. This can apply to the voltage in a circuit, a hole in a card, a magnetic field in a piece of recording tape, or a light in a bulb. Since there are only two possible bits, there is little chance for error—the bit is either a 0 or a 1 with no digits between.

The disadvantage of binary numbers is that each digit only carries a very small amount of information. Large numbers require very many digits to express them. For example, the binary number 1001101001 seems quite large and yet is equal to only slightly more than six hundred. As a result even fairly common numbers turn out to be quite long in binary. This creates problems, not so much for the computers as for the people who use them. More on this in a moment.

Binary numbers would not be too useful if there wasn't general agreement on what they mean and how to use them, and, if it was difficult to convert numbers to and from binary. Fortunately, this is not the case. Binary numbers follow some very simple rules.

Starting at the right end of a simple binary whole number, the rightmost bit has a value of 1 and each bit to its left has a value twice that of the one to its right. For example, in the binary number 101 the rightmost 1 has a value of 1. The digit to its left has a value of 2. The next digit a value of 4, and so on. To convert this number from binary to decimal we would simply write the value of each digit underneath it, multiply each digit by its value, and add the results, like this:

1	0	1	
x4	x2	x1	
4	+ 0	+ 1	= 5

It is fairly easy to make a table of some simple binary numbers and their decimal equivalents:

Binary	Decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	5
111	7

To continue the table, we would have to add more bits to the binary numbers, since with three bits we can only go up to 7. Four bits will take us from 8 to 15, but then we must add more bits so that quite a few may be needed to represent some common numbers in binary.

When a binary number is handled by a computer, the number always has a certain number of bits regardless of the decimal number it represents. If fewer bits are needed, then the computer pads the number with zeroes at the left. Each binary number as stored by the computer is called a *word*. Its length is called the *word length*.

Large computers may have word lengths of 32, 36, or even 60 bits; small computers may have word lengths of just 8 or 12 bits. Eight-bit words, or eight-bit chunks of larger words, are often called *bytes*. Most of the small home or hobby computers use such bytes.

When a small number is stored in a byte, extra zeroes are inserted in front of it to stretch it to the full eight bits, as in 00000101 which is simply 101, or 5 in decimal. The largest number which will fit into a byte is one having eight ones, that is, 11111111. In decimal, this translates into 255, not large enough for most applications.

To store really large numbers having perhaps ten or twenty bits we must use two or more bytes. For example, the binary number 1001101001 would be stretched out to a full sixteen bits by adding zeroes to make it 0000001001101001. Then the first eight bits would be handled as one byte and the last eight as another.

Although the use of binary numbers is convenient from the point of view of the computer's internal circuitry, it is quite difficult for us humans who have to look at them printed on paper or displayed on a TV screen. Just a few minutes spent

www.americanradiohistory.com

studying long binary numbers or trying to tell them apart is enough to give anyone a headache. For this reason we use *octal* or *hexadecimal* numbers rather than binary.

An octal number is one which uses only the digits 0 through 7. Eights and nines are not allowed. It is called octal since the prefix *oct* means eight. This number system has exactly eight different digits. It is used because the conversion between binary and octal is very simple and requires only a small amount of circuitry. In fact, the computer circuitry which connects to a keyboard or printer already has most of what is needed to do the conversion.

To convert a binary number to octal, the computer simply separates the bits into groups of three and converts each group into a decimal number from 0 through 7 using the conversion table we have already discussed. For example, the binary number 101101 would be split into 101 and 101, which converts into the octal number 55.

Octal numbers are a bit awkward when the binary number to be converted doesn't have the right word length. A six-bit or nine-bit number is easily split into groups of three, but an eight-bit byte is not. The solution is to add a few extra zeroes at the left if needed. For instance, the byte 01010011 would be stretched to 001010011, then split into 001-010-011 and converted into octal 123.

Hexadecimal numbers, on the other hand, use sixteen different digits (hex means six and deci means ten.) Since we only have the ten digits 0 through 9 available, hexadecimal numbers 'invent' a few more digits simply by calling them, A, B, C, D, E, and F; these letters take the place of the digits which would stand for 10 through 15 if they existed. We can prepare a short table which relates binary and hexadecimal (or *hex* for short):

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Converting a binary number to hex is done by splitting it into groups of four bits, and then converting each group with the aid of this table. For instance, the byte 01010011 would be split into 0101-0011 and translated into 53 in hex.

Octal and hexadecimal numbers, although not internally used by the computer, are often used by the people who have to communicate with computers. It is easier to read and visualize a short octal or hexadecimal number than a long binary number. It would be most convenient if everything could be done in decimal, but this is often not practical for various reasons and octal or hex numbers are the best compromise.

But why use both octal and hexadecimal? Why not just one? The answer lies in the fact that some binary numbers are easily split into groups of three bits but not four. Others are easy to split into groups of four but not three. Obviously a byte of eight bits is easier to split into two groups of four than into three groups of three bits. On the other hand, a fifteen-bit word would be easily split into groups of three bits, but not four. Hence the word length of the computer determines which number system is used much of the time.

Since most microprocessor-based home and hobby computers use eight-bit bytes, most use hexadecimal numbers. Many years ago, hexadecimal was almost unknown. Even early microprocessors such as the Intel 8008 used octal rather than hex (despite their 8-bit bytes.) But today hexadecimal numbers are most common. A popular misconception is that hex is modern and octal is in some way old fashioned.

This prejudice against octal showed up recently when Heath introduced their H8 computer system which uses the 8-bit 8080A microprocessor. At a time when most 8080 systems used hex, Heath chose to use octal numbers in all their programs and literature. Almost immediately there were some who claimed that Heath made a mistake in going back to an 'obsolete' system. Yet there are many good reasons for going to octal rather than hexadecimal.

Look at the 8080 microprocessor a bit more closely. It is an 8-bit computer, and we have agreed that splitting an 8-bit byte into three-bit groups requires the addition of an extra zero at the left. This is more awkward than simply dividing it into two groups of four bits. If this were all there were to it, then hex would be preferable to octal. But there is another side to the story.

In any computer system, a *program* is required to instruct the computer what to do. This program is written in a special code called a *programming language*. The most fundamental language (and the only on which the computer understands without prior translation) is called *machine language*. In machine language simple machine instructions are coded as binary numbers and stored in the computer's memory.

In the 8080 processor there is a group of temporary storage circuits called *registers*, used to hold numbers or other data while they are being processed. These are called the A, B, C, D, E, H, and L registers. A common instruction in a program is to move a number from one register to another. Let's look at a typical pair of these 'Move' instructions.

Move A to B: 01000111
Move B to A: 01111000

There is something very orderly about these machine language codes. Both start with 01, and that means 'Move'. The next three bits specify the *destination* where the number is to be moved to. The last three bits specify the *source* of the data. Now rewrite these codes in hexadecimal:

Move A to B: 47
Move B to A: 78

There certainly doesn't seem to be much sense to this. But rewrite in octal and a pattern emerges:

Move A to B: 107
Move B to A: 170

When you write down some more possible Move instructions, the picture becomes much better:

Instruction	Octal	Hex
Move A to B	107	47
Move A to C	117	4F
Move A to D	127	57
Move A to E	137	5F
Move B to A	170	78
Move C to A	171	79
Move D to A	172	7A
Move E to A	173	7B

In octal these instructions make some sense. The first digit is always a 1 for Move. The second digit specifies the destination. The third specifies the source. In hex there is not much similarity.

Now, when you consider that the 8080 has seven of these registers, plus an external memory which is programmed as if it were an eighth register, there are 56 possible Move instructions. A number can be moved from any one of eight sources to any one of the other seven destinations, and 8x7=56. If you had to memorize the 56 codes, which would be easier, octal or hexadecimal? Octal, of course.

The Move is just one of many 8080 instructions. Many of the others are based on the same kind of source-to-destination idea. In each case, octal notation makes the instructions easier to remember than hexadecimal.

So there is more to choosing between octal and hexadecimal than just the word length of a computer. We choose octal over hex (or vice versa) because it makes programming simpler or more obvious, even though it may at first seem a bit more awkward. Either way, whether you write it, 1750 or 3E8, it is still easier for a human than to write it as 001111101000.

www.americanradiohistory.com