Micro-PROCESSOR MICROCOURSE

PART 1. NUMBER SYSTEMS

A series devoted to understanding and working with these omnipresent digital devices.

THE MICROPROCESSOR has ushered in a new era of electronics. Just as the transistor conquered the vacuum tube and the integrated circuit replaced a handful of transistors, the microprocessor can replace dozens or even a hundred or more IC's.

The conventional digital logic circuit is "hardwired" and its operation cannot be easily altered after it's built. The microprocessor, however, is functionally equivalent to the central processing unit of a digital computer. Add some memory, and the microprocessor can be programmed to function as a digital controller, calculator, computer, or a dedicated logic circuit. Merely replacing the instructions in the memory with new ones will completely change the role of the microprocessor.

Most electronics enthusiasts, from professionals to hobbyists, are aware of microprocessors and some of the things they can do. Computer hobbyists are particularly close to microprocessors since inexpensive hobby computers were first made possible by the Intel 8008 and 8080 microprocessors.

However, microprocessors are so new and different that many of those who are interested in electronics have not yet become familiar with their basic operating principle, much less their programming requirements. The POPULAR ELECTRONICS "Microprocessor Microcourse" is a series of articles that reviews many of the basic operating principles of digital logic circuits and culminates with a detailed description of the architecture and operation of PIP-2, a simple tutorial microprocessor.

The simplest digital logic elements operate on the basis of the presence or absence of an electrical signal. This two-state situation can be used to represent numbers and implement operations in the two-digit binary number system. We'll learn more about the devices and circuits that perform the functions later. First, let's review the basics of binary and a few other number systems.

If you learn how microprocessors work, you'll understand their role in microwave ovens, CB transceivers, autos and computers.

Number Systems. The ten-digit decimal number system is very easy to learn and use. At least that's what most of us were taught in school. But think about decimal arithmetic for a moment. To add any two decimal numbers, for example, you must first have memorized 100 individual addition rules!

What are these rules? They're numerical relationships like $1+1=2$; $4+5=9$; $3+7=10$; etc. Simple? Yes, almost transparently so, but only because we have already memorized them.

As you can see, the "simple" decimal number system isn't very simple at all. And we haven't even covered the rules required to subtract, multiply and divide decimal numbers. In all, there are literally *hundreds* of individual rules for performing the various operations of decimal arithmetic.

It took you five or six years to master the rules of decimal arithmetic, but you can master the rules of binary arithmetic in only five or six *minutes*. The binary system has only two digits or *bits*, 0 and 1, so only a few rules are necessary for performing binary arithmetic.

Here, for example, are the rules for binary addition:

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=0$, carry 1 or 10
$1+1+1=10+1=11$

You can use these five rules to add any two binary numbers. There are equally simple rules for binary subtraction. And since multiplication and division can be accomplished by, respectively, repeated addition and subtraction, the rules for binary arithmetic are far simpler than those for decimal.

You can also use the binary addition rules to count in binary. Start with 0, add 1, and continue adding 1 to consecutive sums. This procedure is called incre-

BY FORREST M. MIMS

menting, and it allows us to quickly generate the first sixteen binary numbers:

0	100	1000	1100
1	101	1001	1101
10	110	1010	1110
11	111	1011	1111

Computer specialists frequently refer to binary numbers like these as *words* or *bit patterns* since they are often used to represent computer instructions and other nonnumerical functions. Words having eight bits are commonly used; they are called *bytes*. A word having four bits is a *nibble*.

Though binary arithmetic is easy to learn, the binary number system has a major drawback from the human perspective. Binary numbers (or words) are often long and cumbersome, difficult to remember, prone to transpositional errors, and difficult to vocalize. For example, a decimal number that uses only a digit or two will require from one to seven bits when expressed in binary. The decimal number 99 is easy to pronounce and remember. Its binary counterpart is an awkward 1100011.

Computer enthusiasts have invented several handy shortcuts and tricks for remembering binary numbers and converting them into their decimal counterparts. These methods are going to become almost second nature to the microprocessor generation, so let's have a look at them.

Converting Binary to Decimal. Converting binary numbers to their decimal equivalents is easy once you know how to expand an ordinary decimal number into its component parts. For example, 653 is 600 + 50 + 3.

The position of the digits in a number like 653 determines the power of ten by which the respective digits are multiplied. Thus,

$$
\begin{aligned}
653 = 6 \times 10^2 &= 600 \\
5 \times 10^1 &= \ \ 50 \\
3 \times 10^0 &= \ \ \ \ 3 \\
\hline
&\ \ \ 653
\end{aligned}
$$

Binary numbers can be expanded using this same method—and in the process converted into their decimal counterparts. Since the binary system has only two bits, the position of a bit in a binary number determines by which power of *two* the bit is multiplied. Thus,

$$
\begin{aligned}
1001 = 1 \times 2^3 &= 1000 \\
0 \times 2^2 &= 0000 \\
0 \times 2^1 &= 0000 \\
1 \times 2^0 &= 0001 \\
\hline
&\ \ 1001
\end{aligned}
$$

We can carry this expansion one step further and convert 1001 into its decimal equivalent. Just convert the powers of two into their decimal values and add the products:

$$
\begin{aligned}
1001 = 1 \times 8 &= 8 \\
0 \times 4 &= 0 \\
0 \times 2 &= 0 \\
1 \times 1 &= 1 \\
\hline
& \ \ 9
\end{aligned}
$$

An even faster way to convert a binary number to its decimal form is to list the ascending powers of two over each bit in the number beginning with the least significant bit. Then add the powers of two over the 1 bits and ignore those over the 0 bits. Thus, to convert 1100110 to decimal:

64 32 16 8 4 2 1
1 1 0 0 1 1 0
64 + 32 + 4 + 2 = 102

Converting Decimal Numbers to Binary. A quick way to convert decimal numbers into their binary counterparts is to repeatedly divide the decimal number by two. The remainders of each division, which will always be 0 or 1, become the binary number. Let's convert 102 into binary using this method:

102/2 = 51, remainder 0
51/2 = 25, " 1
25/2 = 12, " 1
12/2 = 6, " 0
6/2 = 3, " 0
3/2 = 1, " 1
Final " 1
102 = 1 1 0 0 1 1 0

Octal and Hexadecimal Numbers. Often binary numbers are used to represent computer instructions and operations. For example, 01110110 is the binary equivalent of the decimal number 118. 01110110 is also the instruction code selected by Intel to represent the instruction HLT (halt) for its 8080 microprocessor.

Binary numbers are also used to represent memory addresses inside a computer. Thus 01110110 can represent the decimal number 118, the instruction HLT, or the 119th address in a computer memory (the first address being 00000000).

Since binary numbers play such an important role in microprocessors and computers, you'll want to learn about a couple of very handy time and space saving shortcuts called the *octal* and *hexadecimal* number systems.

Decimal numbers have ten as their base; therefore the largest decimal digit is 9. Octal numbers have eight as their base, and that means the largest octal digit is 7. Since the binary equivalent of the decimal digit 7 (which is equivalent to the octal digit 7) is 111, it's easy to convert any binary number into its octal counterpart by simply dividing the bits in the number into groups of three and converting each group into its decimal equivalent. Thus, the binary number 01110110 becomes 01 110 110 or 166 in octal.

When listing numbers having different bases, it's customary to indicate each number's base with a subscript. Therefore 166_8 is an octal number. Obviously 166_8 is much easier to remember than 01110110_2. And it's easy to convert 166_8 back to binary by simply writing out the binary equivalent for each digit:

1 = 01
6 = 110
6 = 110
01 110 110

(continued overleaf)

Hexadecimal numbers have sixteen as their base. They're commonly used to simplify 8-bit bytes into easily remembered two-character numbers.

The hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Don't let the letters A-F confuse you. There are more than enough decimal digits for the binary and octal systems, but not enough for all sixteen hexadecimal digits. The letters A-F complete the six digit spaces beyond the ten digits 0-9.

It's easy to convert a binary byte into hexadecimal or simply *hex*. First, divide the byte into two nibbles. Then assign the hex equivalent to each nibble. 1111_2 is F_{16} and 0110_2 is 6_{16}. Therefore, 11110110_2 is $F6_{16}$.

To convert a hex number to binary, just assign the binary equivalent to each hex digit. Thus $F6_{16}$ is 1111_2 and 0110_2 or 11110110_2.

Incidentally, though it's correct to identify a hex number with a subscript 16, it's not necessary to tag on the subscript if the number includes one of the six digits borrowed from the alphabet. Everyone seeing it will know it's hex. Also, some computer companies identify hex numbers with the $ sign. So F6E9 is the same as $F6E9.

Most of today's microprocessors use 8-bit address and instruction words, so you'll often see programs given in octal or hexadecimal. While it takes time to become used to these new number systems, especially hex, you'll find them very handy as you become more involved with microprocessors. The conversion table given below will help you become more familiar with both octal and hexadecimal numbers. ◇

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

(Series continues next month)