PRACTICAL MICROCOMPUTER PROGRAMMING:

The INTEL 8080

W. J. Weller
A. V. Shatzel
H. Y. Nice

PRACTICAL MICROCOMPUTER PROGRAMMING:

The INTEL 8080

W. J. Weller
A. V. Shatzel
H. Y. Nice

Northern Technology Books

Library of Congress Catalog Card Number: 76-44565

PREFACE

It is appropriate to begin a work of this sort by paraphrasing a certain 19th century political economist to the effect that: A specter is haunting the industrialized world, it is the specter of the computer.

The home and village shops and industries of 200 years ago viewed the introduction of industrial machinery with apprehension and fear. These machines plainly threatened the existence of the old institutions and the threat was real. The old home and village industries perished with the former artisans and craftsmen forming the work force of the new factories. The social shock waves set into motion by the change are still being felt today while we stand at the edge of yet another industrial revolution which promises to have effects at least as far reaching as the first. It is the revolution of automation, that of the computer.

While computers were costly, cranky and unreliable contraptions we could still maintain the posture that they would cause no fundamental change, at least not in our time, since the cost of their use would be prohibitive, somewhat like the posture of Lord North in the early 1770's assuring George III that revolution in the American colonies was unthinkable, "at least not in our time". The appearance of the reliable low cost computer in the past few years has made it impossible to hold onto this illusion any longer.

As this is being written there is an advertisement on the desk for a small computer system in kit form which costs $345. It is not a toy but a full usable computer, capable of automating the security and environmental controls of a house, for example. No reasonable observer can find any ground to doubt that the time of mass use of computers has arrived, but there is a catch and a big one.

The catch is well illustrated in a gruesome short story by W. W. Jacobs, called "The Monkey's Paw". In this story a couple comes into possession of a talisman or magic piece which allows them to make three wishes. The first wish is for a large sum of money and it is fulfilled by insurance when the couple's son is mangled to death in the machinery of the factory where he works. The second wish is for the return of their son and they are then tortured by the sound of his mangled corpse pounding at the door in the night. The third wish is used to put their son back into his grave.

The point of this story, like many such stories of magic, is that the agency of magic is completely literal. It does *exactly* what it is told to do without regard to the consequences to the controller of the magic agency. Such stories also depend on a characteristic of human mental processes, namely that it is difficult for us to state our desires in exact literal terms, yet computers demand that we do just that, and therein lies the danger in using computers (and monkey's paws).

The problem of communicating with a computer is not a new one of course. It has been recognized as a problem since the beginning of the craft but was manageable since the number of people involved was fairly small, limited by the small number of existing computers due to high cost. The low cost of the new systems has resulted in their widespread dissemination among computer users not equipped by previous experience to use them effectively. The problem of bringing effective instruction to these new users demands solution, and it is to a part of this problem that this book is directed.

No book including this one can make an expert programmer out of a novice, nor can any combined number of them. Like most other significant human skills, expertise at programming has a large component of experience. Nothing can replace the knowledge that comes of having done a thing successfully. Experience is the only way to put meat on the bare bones of theoretical knowledge.

What a book can do is to guide the novice through some of the elementary solutions to problems which arise in using computers, saving him the time and frustration attendant to the reinvention of these solutions for himself. Reading about a method in a book is no substitute for having the real knowledge that comes of experience but it is better than no knowledge at all. An illustration of a method or solution is a guide to solution of similar problems encountered by the working programmer.

To talk about microcomputers involves talking about *a* microcomputer. To attempt to discuss a broad spectrum of machine types, or even worse, some idealized but imaginary machine is of no use to the beginner. It simply creates confusion. The example machine in this book is therefore a particular microcomputer, the 8080 manufactured by the Intel Corporation of Santa Clara, California.

The choice of the 8080 was fairly obvious. There are more 8080's in use at the time of writing than any other microcomputer. A book addressed to the 8080 directly therefore stands to yield the greatest benefit to the largest number of microcomputer users.

The particular 8080 configuration used by us was assembled by one of the authors (HYN) from a kit manufactured by MITS Inc. of Albu-

querque, New Mexico. The only peripheral device used in the mechanics of programming was an ASR-33 Teletype. Certain of the examples in later chapters make use of an analog to digital converter manufactured by the Bessell Corporation of Iowa City, Iowa.

The programming vehicle used throughout the book is an assembly program of the author's own manufacture. It is of the cross assembler variety, making use of a host minicomputer to perform the actual mechanics of assembly of code for the 8080. This cross assembler runs on a Computer Automation LSI/2 minicomputer. Its full source text is given in an appendix for those who may be interested in its design details. No understanding of the workings of this cross assembler is required in the book.

Finally, the authors wish to express their thanks to the following individuals and organizations who have contributed in various ways to this book. They are, alphabetically:

Bessell Corporation
Mr. Dave Bunnell
Computer Automation Inc.
Mrs. Jean Duke
Dr. Don Enemark
Mr. Ron Gabel
Mrs. Barbara Gibisch
Mr. Sal Graziano
The Hewlett-Packard Company
Intel Corporation
Mr. Wayne Johnson
Mr. Dave Methvin
MITS Inc.
Mr. Ed Roberts
Mrs. Gale Schonfeld
Mr. Ted Singer
Mr. Randy Smith
Victor Comptometer Inc.,
Components Division
Mr. Karl Weller
Mrs. Ruth Weller

Chicago, September 1976

TABLE OF CONTENTS

1 / BINARY OPERATIONS

Computer components have the property of being in an on or off state. Whether we choose to call these two states on and off, one and zero, yin and yang or set and reset is pretty much irrelevant. The fact that there are only two states makes itself felt in every dealing with the computer. The user is forced to think in a system in which there are only two digits. Conventionally we choose to call these states one and zero.

The idea of expressing information by means of a two state code is not a new one and predates the computer. Situations in which only two states are allowed to express a message are familiar to the reader. Consider: "Hang a lantern aloft in the belfry arch of the North Church tower as a signal light. One if by land and two if by sea . . ." Or how about "If the shade is up don't come in. My husband is home". In both of these cases the agreed upon signal can only assume two states. Paul Revere expected either one lantern or two. No lanterns or three lanterns was not a legitimate signal. Similarly the shade is to be either up or down. In between is not a valid signal. Presumably this would leave the receiver of the shade message in a dither of passionate indecision. Morse Code is another example in which two states, dot and dash, are used. In this case more complex information, the alphabet, is built up from combinations of the two states, e.g., the $\cdot\cdot\cdot-$ code of World War II fame, being the code for the letter V.

The two digits zero and one can be used in a not dissimilar way to build up more complex numerical information by using them in a number system in which they are the only two existing digits, the so-called binary system.

Like the decimal system with which we are familiar, the binary system uses the position of a digit in a string of digits to indicate its value. In decimal for example the number 329 means nine units plus twice the base of the number system (ten) plus three times the square of the system base (10^2 or 100). The positions of the digits have specific meanings in decimal as they also do in binary. The values of the places in decimal are simply the successive powers of the number base, starting with zero. The number $10^0 = 1$ (any number to the zero power is one). The first power of the base $10^1 = 10$ the base itself, while the second

power $10^2 = 100$, and so forth. In a similar way the place values in a binary number are the successive powers of two. The first few of these are:

Powers of Two

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$

The decimal value of a binary number can be found by simply adding up the values of the columns in which ones are found. The binary number 110 has a zero in the 2^0 position, and ones in the 2^1 and 2^2 positions. Its value is therefore $2^2 + 2^1 = 4 + 2 = 6$. Another conversion to decimal is shown in example 1-1.

Example 1-1

Convert the binary number 11010011 to decimal.
First we write out a set of the values of the powers of two, and then write the digits of the binary string under the appropriate powers, as:

128	64	32	16	8	4	2	1
1	1	0	1	0	0	1	1

Ones appear in the 1, 2, 16, 64 and 128's columns, so the decimal value of the number is simply the sum of these powers of two:

$1 + 2 + 16 + 64 + 128 = 211$

The powers of two do not end with 128 of course but continue indefinitely. Like decimal numbers, binary numbers can be of any length.

More extensive tables of the powers of two are available in most mini and microcomputer manuals.

It will be noticed that the number in example 1-1 above required eight binary digits for the expression rather than the three required for its decimal equivalent. This is not any disability as far as the computer is concerned but is clumsy for the human programmer. Copying long strings of digits is an error prone process and methods have been developed for shortening this work. These are the so-called octal and hexadecimal systems. These systems depend for their ease upon the fact that converting binary to a number base which is an integral power of two amounts only to a regrouping of the binary digits or bits. The process is very simple. To convert the above number to octal we split it into groups of three bits beginning at the right as:

11 010 011

These groups are then read as if they were independent quantities. The number above is read 3-2-3. In writing such numbers they must be distinguished from decimal numbers in some way, since confusion can arise. This is done by adding a subscript which indicates the number base, in this case 8. The above number would be written:

$$323_8 = 11010011_2$$

A more convenient way to treat binary numbers is in base 16 or hexadecimal, since the eight bit grouping (word) of the 8080 then divides evenly into two groups which contain equal numbers of bits. Since 16 is the fourth power of two, the bits are grouped in fours as:

1 1 0 1 0 0 1 1

and the digits read as independent groups. For the rightmost group above this poses no problem, it is a 3, but the left group, 1 1 0 1, has no decimal digit equivalent. Its value is $8+4+1=13$ but we have no way of expressing 13 as a single symbol. We therefore assign the first six letters of the alphabet, A through F, as the symbols for the six possible combinations of four bits that cannot be expressed as normal digits. These values are:

Hexadecimal Digit Values

binary	*decimal*	*symbol*
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

While this notation may seem clumsy at first it quickly grows familiar. The above table can be used to convert any binary number to hexadecimal or vice versa, as shown in example 1-2.

Example 1-2

Convert the binary number 1100110101001011 to hexadecimal. First the number is written in groups of four bits, as:

1100 1101 0100 1011

and the value of each read from the table. The leftmost group is a twelve, symbol C, the next is a thirteen, symbol D, the next 4, and the rightmost is an eleven, symbol B. The hexadecimal value of this binary number is therefore:

CD4B

Arithmetic operations are performed on binary numbers in pretty much the same way as on decimal numbers. Addition is done by starting at the right and adding the first two digits, with the sum being written below. If the sum is greater than the capacity of a single digit a