

PRACTICAL MICROCOMPUTER PROGRAMMING:

The INTEL 8080

W. J. Weller

A. V. Shatzel

H. Y. Nico

PRACTICAL MICROCOMPUTER PROGRAMMING:

The INTEL 8080

W. J. Weller

A. V. Shatzel

H. Y. Nice

Northern Technology Books

Copyright © 1976 by W. J. Weller, A. V. Shatzel and H. Y. Nice

All rights reserved. No part of this publication may be reproduced, copied or transmitted in any form or by any mechanical or electronic means without written permission of the copyright holders.

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

Preface

Chapter 1 Binary Arithmetic and logical operations. 1

Two state codes. Powers of two. Conversion to and from binary and decimal. Hexadecimal notation. Binary addition. Negative binary numbers. Ones complement. Twos complement. The AND function. Masking. The Inclusive OR function. Merging. The Exclusive OR function. Shifting. Logical shifts. Rotary shifts.

Chapter 2 Organization and Structure of a Computer. 15

Definition of memory. Definition of word and address of a word. Memory sizes. RAM and ROM memories. The Arithmetic and Logical unit. Registers. CPU definition. The program counter. The instruction register. Concept of a pointer. Basic instruction execution cycle of the computer.

Chapter 3 Machine Language Programming and the Assembly Program. 19

Writing binary instructions. Mnemonics for binary instructions. Translation of symbolic programs by an assembly program. The source program. Labels and operands. The assembly listing. The symbol table. The object program. The loader. Assembler error flags. Undefined and multiply defined symbols. Pseudo operations. Cross assemblers.

Chapter 4 Moving Data with 8080 Instructions. 26

Working registers in the 8080. Use of MOV between registers. The ADD and SUB

instructions. ORA and XRA. Use of AND. Movement of data between registers and memory – LDA and STA. Swapping contents of memory locations. Movement of data between memory and registers with MOV. Use of the H-L pair. The DATA and DBL pseudo-ops. Use of LHLD. Using an incremented address pointer. The SHLD instruction. Use of LXI. Using B-C and D-E as pointers. LDAX and STAX. INX and DCX.

Chapter 5 Binary Arithmetic on the 8080. 36

Definition of arithmetic flags – carry – zero – sign. The F register. Definition of overflow. Definition of carry. Difference between overflow and carry. Overflow detection and the 8080. Multiple precision numbers. Double precision addition. Double precision subtraction. Use of MVI. INR and DCR. The JMP instruction. Conditional jumps. Loops. Use of CMP. Pseudo-ops JEQ – JNE – JAL – JGE. Algebraic comparison. RLC and RRC. RAL and RAR. Pseudo-ops LLA and LRA.

Chapter 6 Multiplication and Division. 50

Multiplication by successive addition. Process of software multiplication. A multiplication loop. Use of CMA and the TCA pseudo-op. Double precision negation. Multiplication by special factors. Process of software division. Definitions of dividend, divisor, quotient and remainder. The divide fault condition. A software division loop. Multiple precision multiplication and division. Division by special factors.

Chapter 7 Using the Stack Pointer. 63

Stack pointer as a data access pointer. Loading the stack pointer. Entering data into the stack – PUSH. The F register. Program

Status Word. Retrieving Stack data – POP. Use of PUSH and POP to swap register pair contents. Use of POP for direct loading of double precision data. SPHL instruction. Stack overflow problem.

Chapter 8 Subroutines. 69

Commonly required instruction sequences. Use of LXI and PCHL as subroutine call. The CALL instruction. The RET instruction. A multiplication subroutine. A division subroutine. Subroutine argument transfer. Arguments directly in calling sequence. Addresses of arguments in calling sequence – ROM requirements.

Chapter 9 Arrays and tables. 80

Definition of an array. Use of RES pseudo-op. Searching an array for maximum and minimum. Movement of an array. The EQU pseudo-op. Reversing an array in memory. The sorting process – bubble sort. A bubble sort subroutine. Searching an array of character strings. Use of ASC pseudo-op. More use of EQU. The table lookup process. Use of table lookup for mathematical functions – a trigonometric tangent subroutine.

Chapter 10 Converting to binary. 94

Decimal forms – BCD and ASCII. Isolating BCD from ASCII. Special multiplication by 10. Use of successive multiplication to convert a two digit decimal number to binary. A multiply and add subroutine. Validity checks. Check for legal decimal range. Checking for overflow during conversion. Double precision conversion. Conversion of hexadecimal to binary.

Chapter 11	Converting from Binary	102
	Conversion to decimal by division. Conversion to decimal by successive subtraction. Conversion of an unsigned magnitude by carry sense. Conversion of signed number by sign sense. Conversion of double precision numbers. A double precision conversion subroutine. Conversion to external hexadecimal. A binary-hexadecimal conversion subroutine. Conversion from binary to an external digit string.	
Chapter 12	Basic Input/Output – Communication with a Terminal	112
	Elementary I/O functions – control – sense – data transfer. The device address. The peripheral interface. Use of IN and OUT. The peripheral status word. Programmed delay using the ready/not ready flag. Use of EQU to define device addresses. A character printing subroutine. Saving A and the PSW on the stack. Printing strings. A string printing subroutine. Multi-line output. Order of carriage return and line feed. Input status. A character read subroutine. A string read subroutine. The wait loop problem. Concept of the interrupt system as ready flag monitor. Initialization of I/O device interface.	
Chapter 13	Controlling a Complex Peripheral – The Victor Matrix Printer	126
	Physical description of the Victor Matrix Printer. Necessary elements of control. Description of function of 6820 peripheral interface adapter. Using the 6820 for an LED display. Initialization of the I/O port. Setting up the port for the Victor printer. Driving the printer. English text and symbols. Vertical resolution limitations. The Cyrillic alphabet bit patterns – RUSKII code. The table lookup	

process for Cyrillic characters. Integration of subroutines into a complete printer driver. Sample text in the Cyrillic alphabet. The full Cyrillic alphabet printed by the Victor Matrix printer.

Chapter 14 Decimal Arithmetic on the 8080. 144

Decimal vs. binary arithmetic. Addition of BCD numbers. Conditions for BCD carries. The auxiliary carry bit. The DAA instruction. Adjustment conditions for BCD sums. Example of 2 digit BCD sum. Example of 4 digit BCD sum. BCD subtraction considerations. Nines complement arithmetic. Tens complement arithmetic. Example of BCD subtraction. Difficulties in decimal arithmetic. The "Playboy" effect in decimal arithmetic.

Chapter 15 Communication with the Physical World. 152

Definitions of digital and analog input and output. Reading digital data from a switch register. Detection of change in digital input data – logical differencing. Example of simultaneous digital input and output. Output to a light display. Use as diagnostic device. Description of the analog to digital converter. The multiplexer. The sample and hold buffer. Controlling the multiplexer. Conversion time. Converting A/D readings to engineering units. An A/D reading subroutine. Example of physical measurement – temperature measured with a thermistor. Arithmetic manipulation of readings.

Chapter 16 Interrupt Driven Processes – the Real Time Clock. 163

Interruptable processes. The nature of an interrupt. Scheme of task priorities for allowing interrupts. Preservation of status of

the interrupted task. **Event** counting interrupts. The real time **clock**. Meaning of interrupt enablement. **Meaning** of device arming. Circumstances **under** which an interrupt can occur. The **interrupt** instruction. Function of the **88-VI**. Save sequence for 8080 registers and flags. Arming the clock. A ten second timeout example. A software time of day clock. Constraints on copying the clock. Keeping sidereal time. Stack requirements for various types of interrupt schemes.

Chapter 17 Interrupt Driven Processes – Input and Output. 178

Separation of input and output interrupts. Conditions for an interrupt. Clearing of ready flag by data transfer. Uncertainty in arm/disarm state on power up. Example of monitoring keyboard for special character. Clearing of VI ready flag. Priority interrupt systems. Output interrupts. An output interrupt service routine for character strings. Simultaneous input and output under interrupt control. Constraints on simultaneous two way data transfer. Conflict of interrupts from separate devices.

Chapter 18 Debugging Programs 188

Programming precautions to ease debugging. Some bad practices. The debug program. Inspection and modification of memory with debug. Jumping into a program from debug. The debug pseudoregisters. Setting the pseudoregisters. Bit specification of the F register. Setting register pairs – the stack pointer – H and L, B and C, D and E. Setting the interrupt enable/disable flag. Register and flag display with the R command. Nature of the return from the breakpoint. Binary instruction specifications for debug purposes.

Stepping through a program by
breakpointing. Example — correcting an
erroneous instruction. Cautions in the use of
breakpointing. Errors caused by the
breakpoint return instructions. Patching in a
missing instruction using a trap instruction.
Review of RST. Economics of debugging.
Some general procedures for estimating
source of error.

Appendices **204**

- A** Source listing of the debug program.
- B** Source listings of the LSI/2 cross assembler
and the ALTAIR object loader, with
assembler manual. **215**
- C** Bit pattern and equivalence table listings for
the Cyrillic alphabet **292**
- Index** **304**

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 2^2 position, and ones in the 2^1 and 2^0 positions. Its value is therefore $2^1 + 2^0 = 4 + 2 = 6$. 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

<i>binary</i>	<i>decimal</i>	<i>symbol</i>
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