Programming Microprocessors

By M. W. Newman

Programming Microprocessors

Programming Microprocessors

By M. W. Memurran

TAB BOOKS
Blue Ridge Summit, Pa. 17214

FIRST EDITION

FIRST PRINTING—MARCH 1977

Printed in the United States
of America

Library of Congress Cataloging in Publication Data

McMurran, Marshall
Programming Microprocessors

Bibliography: p.
Includes index.
1. Microprocessors-Programming. I. Title.
QA76.6.M326 001.6'42 77-3736
ISBN 0-8306-7985-5
ISBN 0-8306-6985-X pbk.

ACKNOWLEDGMENTS

I wish to thank the engineers and management of Rockwell International Electronics Operations, who provided valuable source information, and in particular, colleagues from the Microelectronic Device Division, who prepared the programs from which some of the AMP examples were adapted. I wish also to acknowledge the use of literature from Motorola and Intel Corporations, which was taken as source data for the discussions of the M6800, MP/L, and the 8080. And I thank Shirley Pairish, Ihla Crowley, and Jane Hornback for their invaluable aid in preparing the manuscript.

Contents

Introduction 9

1 Basic Microprocessor Organization and Functions 13
Microprocessor Configuration

2 Number Systems 26
Binary Integers—Binary Arithmetic—Complementation—Binary Multiplication—Precision—Octal and Hexadecimal Systems—Number Conversions—Binary-Coded Decimal Numbers

3 Basic Microprocessor Logic 52
Introduction to Boolean Algebra—Veitch Diagrams—Binary Adders

4 Fixed-Point Arithmetic 66
Conventions—Scaling Rules for Fixed-Point Arithmetic Operations

5 Basic Microprocessor Programming 76
Flow Charts—Memory Addressing and Paging—Instruction Set—Introduction to Assemblers

6 Programs and Subroutines 107
Initialization Routine (ST)—Zero RAM Subroutine (ZORM)—Word Shift Subroutine (WRS)—Bit Designation Subroutine (BDS)—Skip on Carry Zero Subroutine (SKCZ)—Binary Left Shift (BLS) Subroutine—Fixed-Point BCD Addition Subroutine (FXA)—Fixed-Point BCD Subtraction Subroutine (FXS)—Return Address Save Subroutine (PSHP)—Fixed-Point Binary Addition Subroutine (FBA)—Fixed-Point Binary Subtraction Subroutine (FBS)

7 Floating-Point Arithmetic 125
Conventions—Arithmetic—Floating-Point Addition and Subtraction—Floating-Point Multiplication

8 Programming Aids 136
Loaders—The Microprocessor as an Assembler Host—Diagnostics—The Simulator or Emulator—The Text-Editor—Time-Shared Systems—The Assemulator

9 Data Exchange and Use of Peripherals 148
Data Format Standards—Parity Check—Check Sum—Keyboard Interface—Display Drive—Auxiliary Storage Media

10 Compilers 163
Basic Compiler Configuration—FORTRAN—MPL Compiler Language—PL/M Compiler Language

11 Microprocessor Configurations 181
Motorola M6800 Microprocessor—Intel 8080 Microprocessor—Rockwell PPS-8 Microprocessor—PPS-8 Central Processor

12 Special Programming Techniques 217
Non-Integral Power-of-Two Scaling—Push-Down/Pop-Up Stacks—Interrupt Handling—Direct Memory Access—Pooled Data—Non-Binary Counters

13 Characteristics and Fabrication of Microprocessors 229
PMOS and NMOS Devices—Complementary MOS (CMOS)—Integrated Injection Logic (I^2L)

Bibliography 243

Appendices

A Single-Digit Hexadecimal (Hex) Multiplication Table 247

B A Proof of the Hartmann Conversion System 248

C Powers and Conversions 250

D Minimum Binary Scales 264

E AMP Instruction Set 266

F RAM Work Sheet 272

Index 275

Introduction

It is the intent of this book to impart a basic understanding of microprocessor configurations and functions, and to describe the requirements and techniques of microprocessor programming in sufficient detail that any user may, in conjunction with the literature for a specific microprocessor, define a useful system and prepare working programs with little difficulty.

Emphasis has been placed here on describing the interaction of the hardware and software systems, the fundamentals of processor arithmetic, and numerical conversions both to and from readable decimal numbers and their equivalent machine representations. Also discussed in some detail are scaling techniques for magnitude control of fixed-point processor data, the basics of floating-point arithmetic, and the efficient use of instruction and data storage. Such important elements are often under-emphasized today by programmers and programming instructors who have come to depend upon powerful compilers using near-English or near-algebraic programming languages. But many would-be programmers do not have unlimited access to a large-scale computer system with almost unlimited memory. Though such compilers can and often do play an important role in later stages of microprocessor programming and system design, it is the purpose of this book to bridge the gap between the elementary microprocessor programming

techniques and the more sophisticated techniques that are becoming available. A good understanding of a microprocessor's features and limitations, as well as established programming techniques, will make it much easier to write simple programs and to make best use of advanced computer-oriented programming systems.

The evolution of microprocessors and programming techniques goes back over many years, and it might prove helpful to gain a brief overview to place things in the proper perspective. Digital computational hardware has been in use since the days of the clever Chinese mathematicians who used simple beads as memory devices, although in fact all arithmetic operations were done in the head of the operator. Actually, the first successful attempt to mechanize arithmetic processes was made by Charles Babbage, who, in 1830, designed and built a "calculating engine" to compute artillery tables. This "engine" consisted of sets of gears and counters that approximated the generation of table entries by use of finite differences. The earliest *electronic* digital computer, the ENIAC, was developed in 1945 by Eckert and Mauckley at the Moore School of Engineering. This machine consisted of an arithmetic center and a memory that stored information in flip-flop bistable registers for rapid retrieval and use by the arithmetic and control portion. This was not a stored-program machine, however. The first truly general-purpose machine, called the EDSAC, was built by a group headed by M. V. Wilkes at Cambridge University in England. The Wilkes team was the first to use assemblers and subroutine libraries, as well as making other contributions to programming techniques and procedures.

The development of other digital computers came rapidly and can be thought of as occurring in four "rounds." The first round included the development of large and very expensive scientific and business computer systems such as the IBM 700 series, Univac 1103, etc. The second round included medium- or small-scale machines that were intended for use by scientific and engineering organizations; these sold in the price range of $50,000 to $100,000. The third round resulted from engineering efforts directed toward increasing inherent reliability, reducing size, and providing "building blocks" or modules that could be purchased separately and combined to form a single system.

Coincident with the latter part of the first round was the development of compilation programs to ease the programming burden. And by the mid 1960s, the old 256-word cathode-ray-tube memory in the Bureau of Standards' Western Automatic Computer (SWAC), an impressive machine built in the early 1950s, gave way to a 128,000-word high-speed core memory.

The fourth round was driven by the ability of industry to provide small, reliable, and inexpensive semiconductor devices, each containing thousands of transistors and diodes (so-called large scale integration, or LSI). This then paved the way for the powerful but small minicomputers.

The microprocessor (or microcomputer) is a very recent configuration based on the organization of the minicomputer, but using only LSI for memory and logic electronics. A minimally configured microprocessor system consists of a power source, a central processing unit, two memories and an input/output mechanism with interfacing electronics. Either the central processor or a memory device with well over 1000 storage elements can be contained on one silicon chip approximately 0.2 inch square.

The technology permitting this high circuit density was initially spurred on by the requirements of military and space programs during the 1960s, and has grown phenomenally in the last five years. With the broadening production base, primarily in the United States and the Far East, the unit cost of a useful set of microcomputer devices had dropped from around $1500 in 1972 to approximately $150 for equivalent computational capability by late 1975. Even more recently, several central processor units have dropped in price from $60 to $20 each for small quantities. The result of this kind of price erosion is that the computational power derived from a roomful of hardware in the 1950s is now available to a user on a few silicon chips for about $100, with several complete single-chip microprocessor systems already beginning to reach the marketplace.

The result of this relatively inexpensive computational tool is a tremendously expanding interest in designing and programming microprocessor systems for an unbelievably wide variety of applications. And, of course, experimenters and hobbyists have quickly welcomed the microprocessor as yet another sophisticated toy evolving from our technological

society. But whether a mere curiosity or an important component in an industrial system, the microprocessor has found solid footing in today's world, and both the serious-minded experimenter and the modern engineer need to know the essential architectural characteristics and programming techniques of these microprocessors in order to achieve the computational functions they desire.

M. W. McMurran

Chapter 1
Basic Microprocessor Organization and Functions

The systems described here are called *microcomputers* about as often as they are called *microprocessors*. Traditionally, the term *microprocessor* was reserved to describe the *central processing unit* (CPU), which contains various registers used to store and manipulate numbers and instructions, thereby performing arithmetic and logic operations. Present usage, however, tends toward calling an entire assembly of devices a microprocessor since the CPU cannot by itself do useful work. Thus the CPU is considered just a part of a larger microprocessor system.

Microprocessors are assembled from a wide variety of integrated circuit (IC) building blocks and so can assume an even wider variety of useful configurations. This variety, however, does not lend itself to an introductory discussion of microprocessors, so it is desirable to first select a representative system as the basis of discussion. Variations of this basic system, as they occur in real systems, will then be treated as modifications of the central theme.

The basic system we will use is based on the Rockwell PPS-4 system. We will call this configuration *AMP*—standing for *A Micro-Processor*. The AMP system illustrated in Fig. 1-1 is made up of a central processor unit (CPU) that is a large-scale integration (LSI) device performing the arithmetic and logic functions, a metal-oxide semiconductor (MOS)

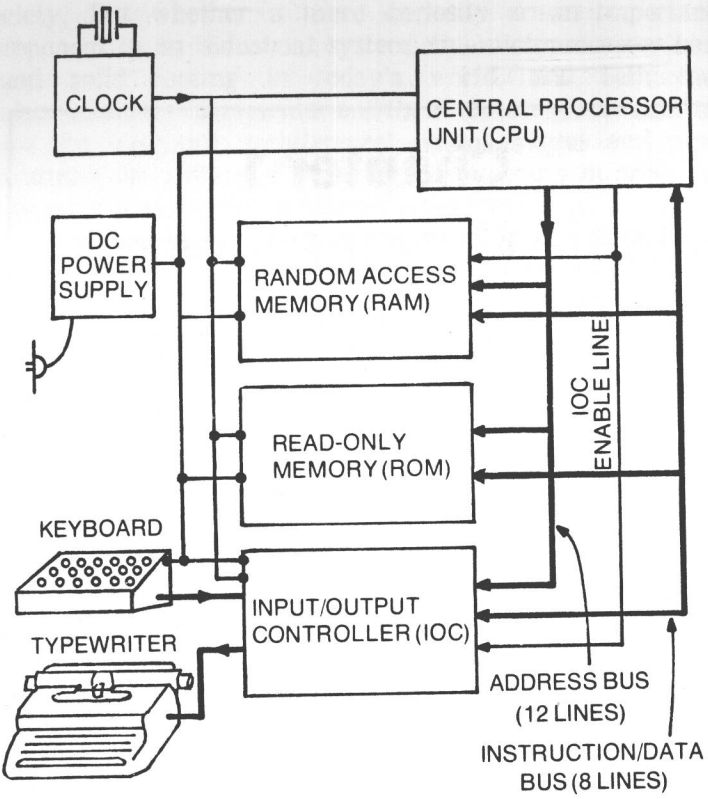

Fig. 1-1. AMP organization and data flow.

read-only memory (ROM) used for permanently storing programs and data, a MOS random-access memory (RAM) used as a "scratchpad" memory for temporarily storing modified instructions and results from calculations, and an input/output controller (IOC) used to direct and format data traveling between the CPU and the outside world.

The processor also needs an appropriate power supply, so the AMP will use +5 and −12 DC supplies. A crystal-controlled clock provides the basic timing source and completes the electrical components. The basic clock rate is 200 kHz and is separated into four phases for control of the internal MOS logic. For input/output the AMP uses a keyboard and a simple pointer.

Data flow in this system is via two information channels, or *buses*. This configuration requires individual devices to be smart enough to talk only during proper time slots. Since the

clock signals are common to all devices, system synchronization is relatively easy to achieve. We will consider all devices to be PMOS (P-channel MOS), though other options would be NMOS, CMOS, or I^2L. A description of these various technologies is given in Chapter 13.

The microprocessor obeys a set of operating rules similar to those of any stored-program digital computer. Processor functions are determined by the CPU under control of a set of instruction patterns, most of which are permanently stored in the ROM. These patterns are decoded by the CPU, resulting in the execution of an operation belonging to one of the following instruction classes or groups, which include some examples of the instruction types and characteristics of each group.

Arithmetic and Logic Instructions. Examples: Add two binary numbers. Logically combine two binary numbers (see Chapter 3).

Data Transfer Instructions. Examples: Load data from memory into an internal CPU register. Store data from an internal CPU register into memory. Exchange data between two registers. Load data from instruction memory (ROM) into a CPU register. (These instructions are termed *immediates*.)

Control Transfer Instructions. Examples: Take next instruction out of normal sequence if content of selected register is less than zero. Skip one instruction if a particular bit equals one.

Input/Output Instructions. These instructions are strongly dependent upon the type of processor used. Examples: Load *n* bits (binary digits) from an external device into a CPU register, or directly into memory. Read single-bit input lines and transfer the data into a CPU register. Transfer *n* bits of output data with the appropriate timing control to an external device.

The various permitted or recognized instructions that a microprocessor will accept is referred to as the *instruction set*. The PPS instruction set, which is the basis for the AMP instructions, is quite flexible and often complex. For example, the execution of some of these instructions will cause as many as four independent operations to be performed simultaneously.

To reduce the work of decoding the bit patterns that represent these instructions, the CPU will normally first determine the class, and then the specific instruction type

within the class. As a result, the bit patterns defining instructions within classes generally have something in common. In addition to the instruction groups introduced here, microprocessors more sophisticated than the AMP have instructions that further extend the processor's capability. Notable are:

Shift Instructions. Most microprocessors have the capability of shifting the binary contents of a register, usually the accumulator, either right a fixed number of spaces or left a fixed or variable number of spaces to provide for the arithmetic operations discussed in Chapters 5 and 6. The simple instruction list chosen for the AMP contains only a right shift command, and this makes programming the AMP a bit more involved than programming some other processors.

Stacking Instructions. A stack is a series of registers connected together so that data enters and leaves through only the first, or top, register. Stacks are generally used for storing reference addresses, usually the next address of an instruction occuring before the microprocessor was interrupted for some reason. Stacking instructions are usually of two types: *push* instructions store a new address on the top of the stack, pushing any other addresses deeper into the stack registers, while *pop* instructions raise the addresses in the stack registers to expose the next in order at the top. This procedure permits the orderly storing and removal of address data, and it is discussed further in Chapters 11 and 12.

Multiplication and Division. These functions are now being built into microprocessor hardware as new processors are introduced, thus significantly adding to the effective speed of these machines. Multiplication and division operations are performed by automatically creating strings of instructions (subroutines). Each time the function is needed, control is transferred to these subroutines. Multiplication is then mechanized by a controlled set of successive additions, and division by a controlled set of subtractions. This permits 20 to 100 instructions to be automatically executed for each multiply or divide instruction, depending upon the precision needed and the idiosyncrasies of the processor used. The resulting trade-off penalty in the excecution time of this method when compared with a separate hardware mechanization is often quite severe, but of course the approach is simpler and cheaper.